December 2017

ESATAN-TMS Thermal User Manual

Prepared by:
ITP Engines UK Ltd.
Whetstone, Leicester, UK

Contents

1. PREFACE

ESATAN^{TM} is a software package for the prediction of temperature distributions in engineering components and systems using the thermal network analysis technique. It enables the user to specify his problem in the thermal network quantities of nodes, conductances and material properties, together with the sequences of solutions required to obtain the required steady-state or transient temperature distributions. It also has a batch edit facility which permits previously defined thermal/fluid networks model to be incorporated into a model and changes to be carried out if required.

The development of ESATAN was undertaken by ITP Engines UK Ltd. under the European Space Agency contract 4791/81/NL/DK(SC). The need for the development was identified by ESA during a review of their analysis methods for spacecraft thermal control purposes. This had concluded that while the network modelling technique employed was satisfactory, the existing software package, SINDA, had significant shortcomings in several areas. These included the checking of models for correctness of purpose, the validation of modifications to existing models, and the merging of several independently developed models of system components to represent the complete system. Additionally there was a need to improve upon the numerical solution algorithms contained in SINDA.

It was further concluded that the incorporation of these extra features into the SINDA package was not feasible, as the existing software structure did not lend itself to such extensions. The decision was thus taken to develop a new program to replace SINDA which would include all of these additional features.

In spite of the problems mentioned above the fundamental approach of SINDA, which separates the topology of the thermal model from the solutions performed on it, was still felt to be attractive and thus was retained in the ESATAN specification. Existing SINDA users will therefore see many familiar features in ESATAN and will find little difficulty in using it. Input is block structured, with free format within each block, and there are blocks for specifying various model data and solution operations to be performed on it.

The major new features introduced with ESATAN are:

i) A sub-model structure within the input enabling a model to be described as consisting of many sub-models. Each sub-model defined is itself a complete and valid model suitable for separate analysis.

ii) Techniques for combining submodels by node merging or by inter-model conductance links.

iii) Implicit submodel definitions by the inclusion of previously defined submodels.

iv) A Mortran language to enable Fortran-like commands to be written either in the model data or operations input blocks.

ESATAN is written entirely in Fortran 77 to ANSI Standard X-3.9 1978 in a strictly top-down structured fashion. It was explicitly designed to run on any computer supporting a Fortran 77 compiler with only minor changes to allow for different run-time characteristics such as file naming rules.

Version 4.1 of ESATAN introduced the FHTS (Fluid Heat Transport System) extension to the thermal analyser. This enables the combined thermal/hydraulic solution of piped fluid networks. Use of the FHTS facilities is an option which can be taken up by an ESATAN installation site.

The remainder of this User Manual contains all the information required to use ESATAN to specify, edit, and solve thermal network models.

2. INTRODUCTION

2.1 Function of ESATAN

ESATAN is a computer program intended to perform thermal analysis by the thermal network technique. It will accept models of components or systems presented to it in thermal network terms, together with instructions for their solution. All models are checked during input for self consistency and translated into a database which is held permanently on the computer storage system until the user chooses to remove it.

Either steady-state or transient analyses can be handled, for problems with any degree of dimensional complexity. Any mode of heat transfer can be accommodated: explicit facilities exist for conduction, radiation, and convective heat transfer, any of which may be non-linear with respect to temperature or time. Additional features permit the handling of phase-change phenomena such as boiling, condensation, melting, solidification, ablation and gas-solid transitions. Material properties, such as specific heat, may also vary with time, temperature or any other independent or dependent quantity.

It should be recognised that ESATAN does not derive network models from specified geometries: that task must be performed by the user. The role of ESATAN is to handle, check, and solve the network models that the user has devised to represent his problem.

2.2 Function of FHTS

The Fluid Heat Transport System (FHTS) is a major extension to the ESATAN thermal analyser. It allows the prediction of fluid pressure, mass flow rate and temperature in a one-dimensional piped network to be made. Thermal-hydraulic solutions may be obtained for single- or two-phase piped fluid systems under steady state or transient conditions.

Users are able to construct piped fluid systems from basic node and conductance data, and also incorporate predefined elements which are used to simulate hardware such as pumps and heat exchangers. A comprehensive library of fluid property data is available which the solution routines continually access during solution time. If the user so wishes existing fluid properties can be redefined or data for totally new fluids added.

Analysis may be all-fluid, all-thermal or both fluid and thermal simultaneously.

2.3 Modes of Operation

A useful precursor to this section is a description of the program input and output terms, which fall under the heading of input, input/output and output.

Input

Source File - user created file for a batch preprocessor run consisting of ESATAN input data.

Input/Output

Model Database (MDB)- file containing numerical and character data derived from the source file; these are not directly accessible by the user.

Fortran Program - a Fortran 77 source program derived by ESAFOR from the MDB file as part of the solution process. Being part of the solution process it is not normally accessed by the user.

Analysis Case Definition (ACD) file – alternative representation of (some) data used for solution, bypassing the model source file. Compact, platform-independent binary format produced by ESATAN Workbench (see Section 3.10 for more information on ACD file).

Output

Log File - ASCII file produced by the preprocessor which echoes the source file, interspersed with any warning or error messages. A log file is also produced by ESAFOR if there are any warnings or errors during its execution.

Output File - ASCII file of analysis results, as requested by the user in the input file.

ESATAN has two principal modes of operation:

i. PREPROCESSING

Preprocessing is the initial definition of a model. It is a batch-mode operation on the computer and consists of the program reading a source file containing the definition of the model topology, associated tables of information relating to material properties, variations of model parameters with time etc, and instructions on solution procedures and sequences. The result of preprocessing is the establishment of a MDB, update of the main model ACD file and the generation of the log file.

ii. SOLUTION

Solution is the processing of the MDB and ACD file contents so as to produce predicted temperatures throughout the model according to the solution instructions contained within it. A Fortran program is generated from the MDB, which is then compiled, linked and run to produce the solution. At the end of solution the MDB and ACD files are unchanged. Output of printed results is entirely under user control.

2.4 Overall Structure

ESATAN has three major components: the Preprocessor, the Fortran generation and the Library. The way in which these components relate to each other is illustrated in figure 2-1.

2.4.1 Preprocessor

The Preprocessor accepts input from a source file prepared by the user through the standard file editing facilities of his computer. The source file contains the model definition in the standard ESATAN format defined later in this manual. It is sufficient here to understand that this input consists of a sequence of blocks of data defining different aspects of the model. (Later it will be explained that it can be a nested sequence so as to define submodels, but for simplicity we assume here that only a single model is concerned).

There are two categories of these blocks of data:

i. **Data Blocks** - these contain the definition of the model topology and parameters such as nodes, conductances, properties etc.

ii. **Operations Blocks** - these contain the operations to be performed on the model defined in the Data Blocks during solution. They contain instructions written in a language called Mortran, a superset of Fortran 77.

When the Preprocessor reads the source file it first encounters the Data Blocks. It processes the data, checks it for consistency and then, if all is well, stores it in a defined internal format on a newly created Model Database (MDB). Most of this data consists simply of numbers and values, and therefore the part of the MDB where such data is stored is called the Numeric Database.

Next the Preprocessor encounters the Operations Blocks containing the Mortran instructions. These it also checks for consistency (e.g. it will ensure the user does not refer to a non-existent node) before storing them in the MDB.

At this point all of the user's source file has been read and the Preprocessor performs some final operations:

i. A copy of the source file is stored in the MDB, but with the input re-sequenced and re-written into a standard form which is detailed later. This copy will be referred to as the MDB file.

ii. The Log File is produced.

iii. Some internal re-organisation of the MDB is carried out so as to produce a compact storage form. The contents are unaltered and the user need not be concerned with the detail as there is no effect upon his model.

An ESATAN source file can optionally reference an Analysis Case Definition (ACD) file; see Section 3.10 for more information on ACD file.

2.4.2 Fortran Generation

The Fortran generation process reads the Operations Block data and translates each instruction into a corresponding Fortran 77 statement or statements. Each Operations Block becomes a Fortran 77 subroutine and the process automatically generates appropriate SUBROUTINE and COMMON statements at the head of each, as well as RETURN and END statements at the end. When all Operations Blocks are processed a MAIN program is also generated, the contents of which are explained later.

2.4.3 Library

To solve for temperatures within a model the user initiates a solution run, and at this point the Library comes into play. As part of his Mortran instructions in the Operations Blocks the user will probably have made reference to a number of elements from the Library. These are modules which perform specific tasks, such as solving the network, calculating temperature dependent properties, or producing formatted output. There are many of them and each is fully described later in this manual. The Library contains each module, in a compiled form as object code.

2.4.4 Solution

Whilst the Fortran generation and solution can be run separately by the user, in most cases these are executed in a integrated manner via a solution run. The sequence of operations in a solution run is also shown in figure 2-1. Firstly the Fortran 77 is generated by calling ESAFOR, the Fortran generation process. This takes the Operations Block data from the Model Database (MDB) to create a Fortran file which is compiled by the system Fortran compiler. This Fortran will contain calls for a number of Library elements, so it is then linked first with the ESATAN Library from which those modules required are extracted, and then with the system Fortran 77 Library to obtain any necessary system specific routines. The result is an executable task which is run on the computer. As a result of statements automatically generated in the main program by the Preprocessor the first action of this task is to access the numeric Database and read all the data relating to the model parameters. This is placed into the common blocks in the solution program, which are also generated by the Preprocessor. Static data is then read from the Analysis Case Definition (ACD) file and assigned to the appropriate parameters. See Section 3.10 for more information on the ACD file. The user’s solution instructions from the original Operations Blocks are then performed.

Output from the run is totally controlled by the user, and can be formatted results or data written to disc storage.

At the end of the solution run the MDB is unchanged.

2.5 Organisation of this Manual

The remainder of this manual consists mainly of detailed explanations of the use of ESATAN. Chapter 3 covers the preparation of input file for the Preprocessor, beginning with a general description of submodel structuring, definition and linking, followed by the global data format rules in Section 3.2. The remaining sections deal with the form of input data for the various data blocks.

Chapter 4 considers aspects specific to fluid-loop modelling with FHTS, including a discussion of fluid properties.

Chapter 5 covers the description of each ESATAN/FHTS element supplied in the library with the installation.

A full explanation of all modules and functions contained in the ESATAN library is given in Chapter 6. The chapter is subdivided into functional sections which can be referred to in the index.

Finally, examples of use of ESATAN appear in Chapter 7, and show at least one instance of each particular feature.

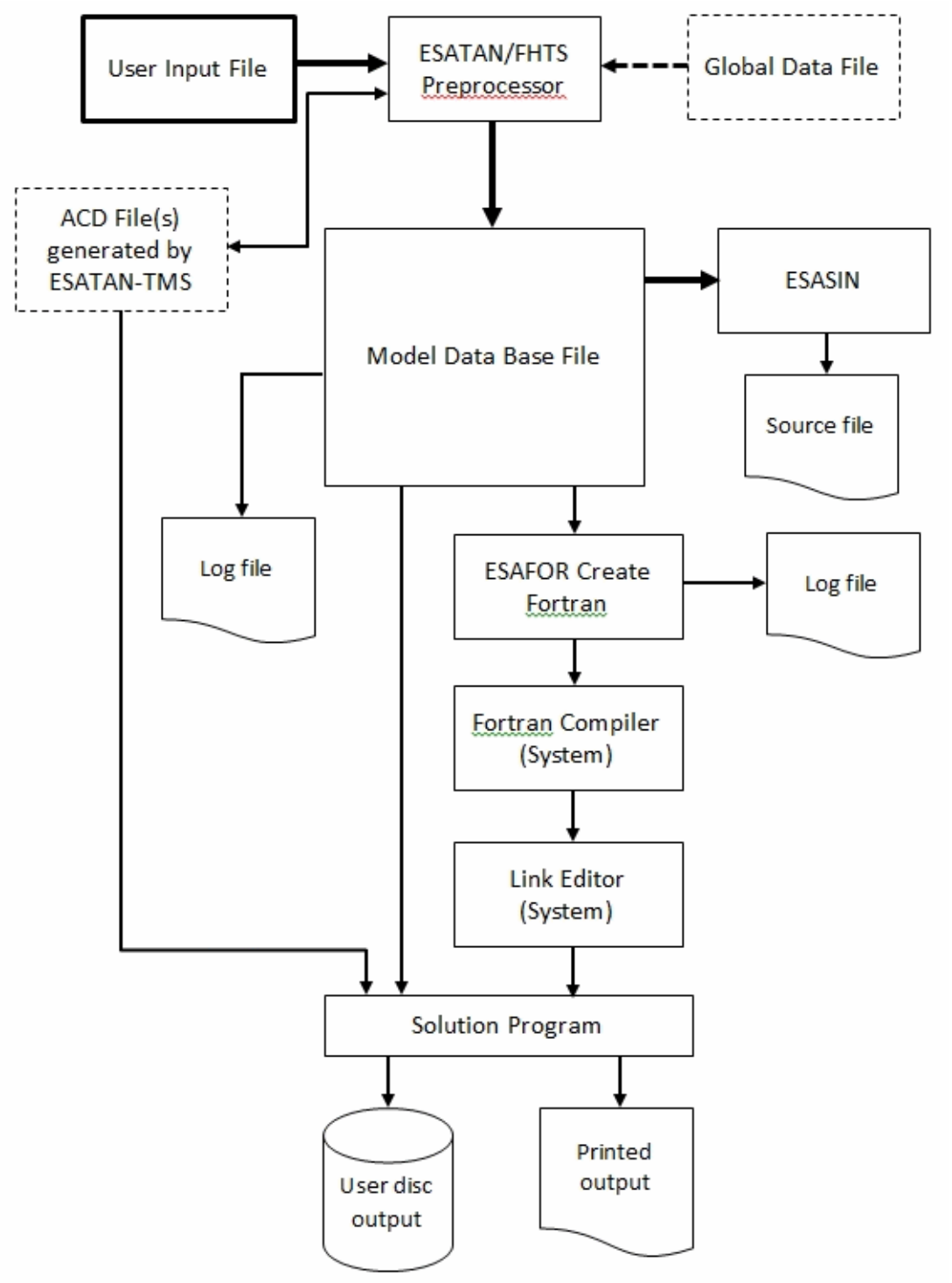


Figure 2-1 The basic ESATAN system

3. ESATAN MODEL DEFINITION

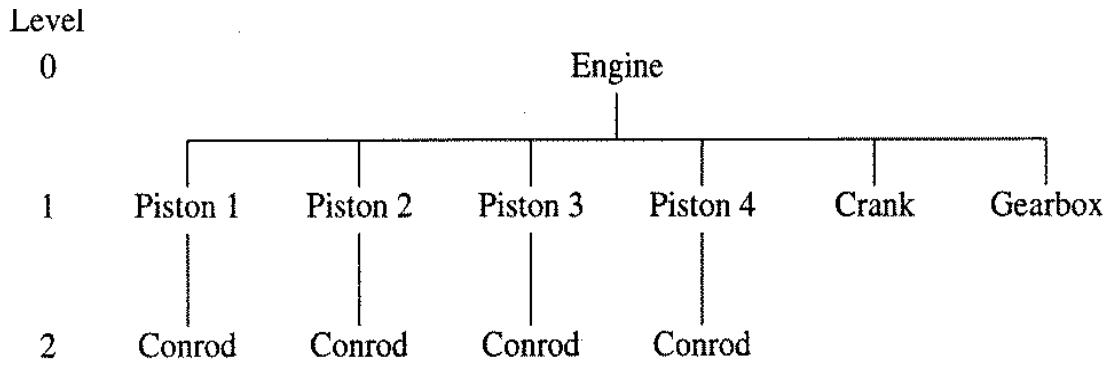
3.1 General Description

This section concentrates on the overall concept of an ESATAN model in terms of submodel structuring and the methods available to define individual submodels.

3.1.1 Model Hierarchy

The concept of submodel structuring is that any ESATAN model may contain a number of submodels, each being individually named and existing as entities in their own right. Each of these submodels may also contain submodels, and so on, to a theoretically infinite extent. In practice a finite limit of 99 is set upon submodel nesting level, for the purposes of internal array dimensioning within the program. The model may, however, contain more submodels than this provided that any one "branch" of the submodel structure does not contain more than 99 submodels. It is also possible that the overall "main" model will itself become a submodel of a much larger model at some future time.

A practical example may be taken as the automobile engine. The "top level" or main model contains the physical description of the crankcase and cylinder head, i.e. the outer shell of the engine. A submodel describing a piston could then be included (four times for a four cylinder engine), each piston submodel containing a submodel describing the connecting rod. Two further submodels, describing the crankshaft and gear box, are also included within the main model. Let us assume that this completes the model of the engine. Schematically, this model would look like so:-



It is not essential that the connecting rod is a submodel of the piston submodel, it could have been included as a direct submodel of the main model.

The concept of model level may be introduced at this point. The top level contains the main model, this is level 0. Submodels contained directly within this are said to be at level 1, and within level 1 submodels level 2 submodels are defined, etc.

To complete the above model, a description of how the submodels join or interface with each other is necessary. Methods of achieving this are described in Section 3.1.5. Having successfully developed the model of the engine, this could then be included in a much larger model describing a complete vehicle. The model "ENGINE" would then become a submodel of this larger model.

Figure 3-1 shows a larger submodel structure. The main model **MAIN** contains 13 submodels in total. Models **E** and **N** are the two level 1 submodels of **MAIN**, **E** contains **C**, **C** contains **A** and **B**. Similarly **N** contains **H** and **M** etc. The way in which this model would be described to the ESATAN preprocessor is as in figure 3-2. This involves the introduction of the keywords $MODEL and $ENDMODEL.

The $MODEL keyword denotes the start of a model or submodel definition, and is followed by a user chosen name for the model or submodel (see Section 3.3.1). Correspondingly, the $ENDMODEL keyword marks the end of the definition, again followed by the model name, although this is optional at this point. We will not concern ourselves with the information contained between the $MODEL and $ENDMODEL lines for the time being.

The general principle behind the structuring concept is that a model is not complete until all of its included submodels have been defined. For example, considering figures 3-1 and 3-2 **MAIN** is not complete until **E** and **N** are defined. **E** must have **C** defined, and **C** must have **A** and **B** defined. Consequently, **A** is the first submodel to be completely defined, since it is the first submodel in the input order having no included submodels. **B** is then defined, allowing **C** to be completed, and then **E** completing the left hand branch of the structure. A similar procedure with the right hand branch results in **N** being defined, and finally **MAIN**. The last line in ESATAN input is always $ENDMODEL for the main, level 0, model.

Referencing items within one submodel from another is allowed, subject to certain restrictions. The referenced submodel must be defined within the referencing submodel. Therefore, model **E** could reference elements of model A. Similarly model **N** may reference model **J** but note that model **G** cannot reference model **J** as **J** is not defined within **G**. Obviously, **MAIN** may reference any of the submodels shown.

Referencing of submodels is achieved by concatenating submodel names. The names are separated by colons (the precise usage will be given later). For example, if addressing model **J** from **MAIN**, the full form is:-

N : M : L : J

However, the full concatenation is not necessary, only sufficient to give an unambiguous reference is required. In this case, the reference **J** would have been sufficient since there is no other model **J** within the entire model.

The last statement implies that a model may contain submodels of the same name. Consider figures 3-3 and 3-4. A detail (the right-hand branch) of our previous examples is shown,

with two amended versions. In both, two submodels named **J** exist. In figure 3-3 they bear the full concatenation:-

N : H : J (i)

N : M : L : J (ii)

and can therefore be addressed uniquely from a model which contains both of them. However, in figure 3-4 both submodels **J** have the concatenation:-

N : M : L : J

and are thus not unique, this submodel structure is clearly incorrect. In order that all submodels may be referenced uniquely, it is essential that submodel names are not duplicated within a model which directly defines the submodels in question.

When referencing a submodel within the current model, only submodels that have been completely defined may be named. It is therefore an error to refer to the top (level zero) model from within the model at any time.

The content of the submodel is either an explicit or implicit definition. These are described in the following sections.

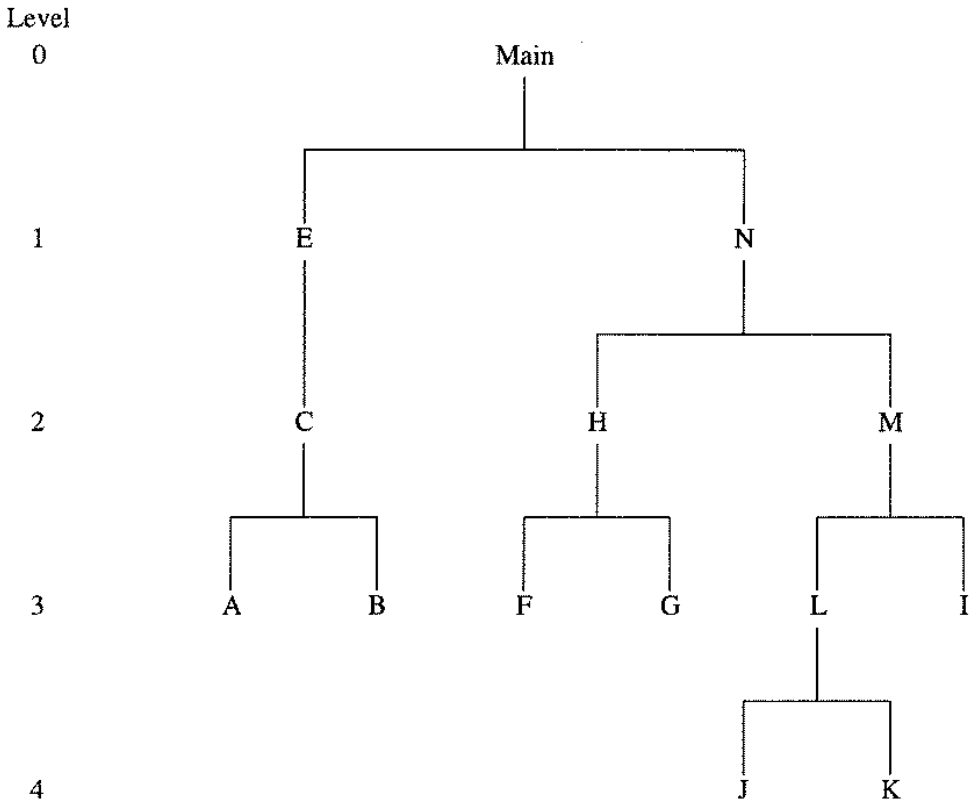


Figure 3-1. Submodel Structure Map

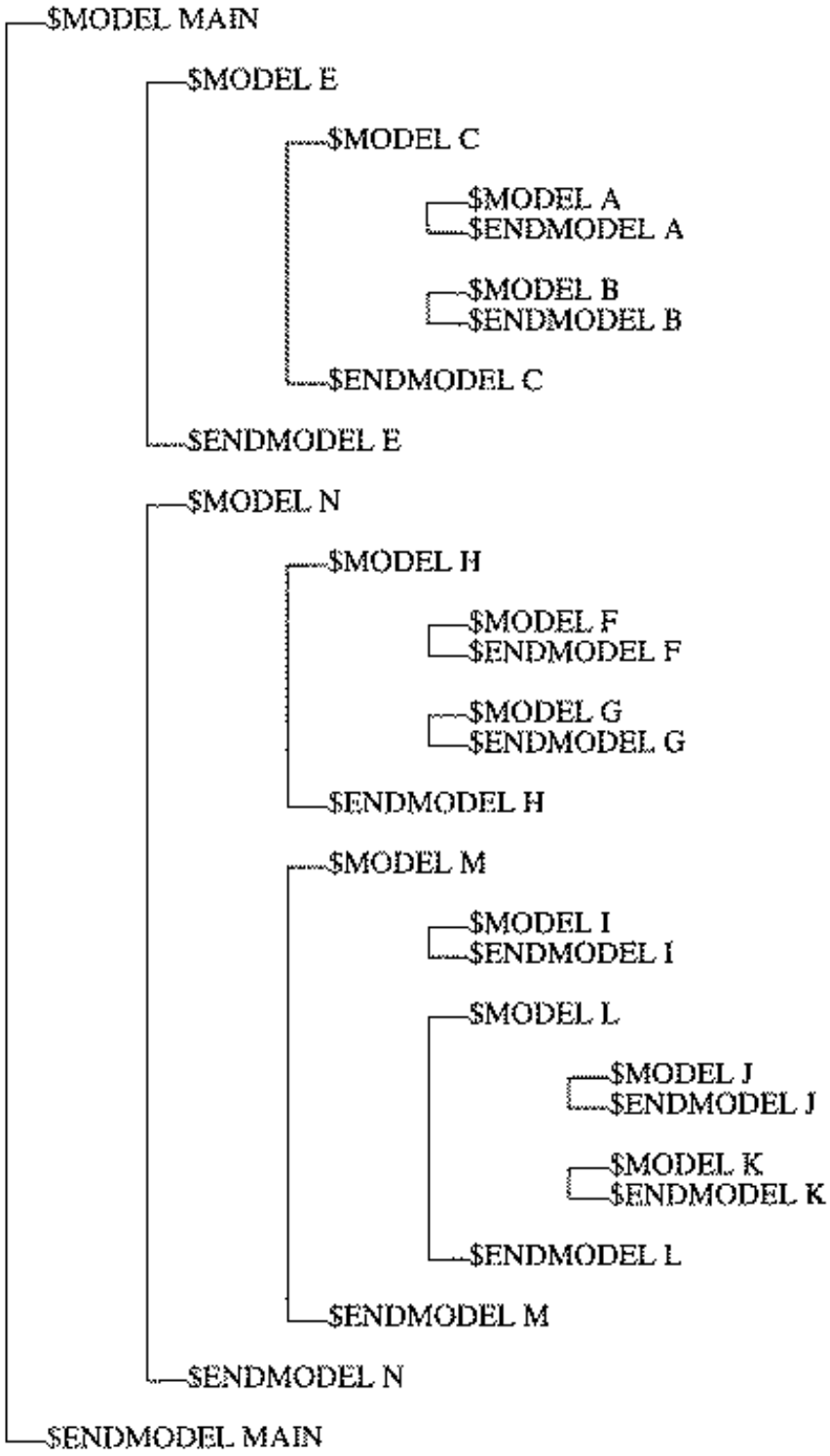


Figure 3-2. ESATAN Submodel Input Method

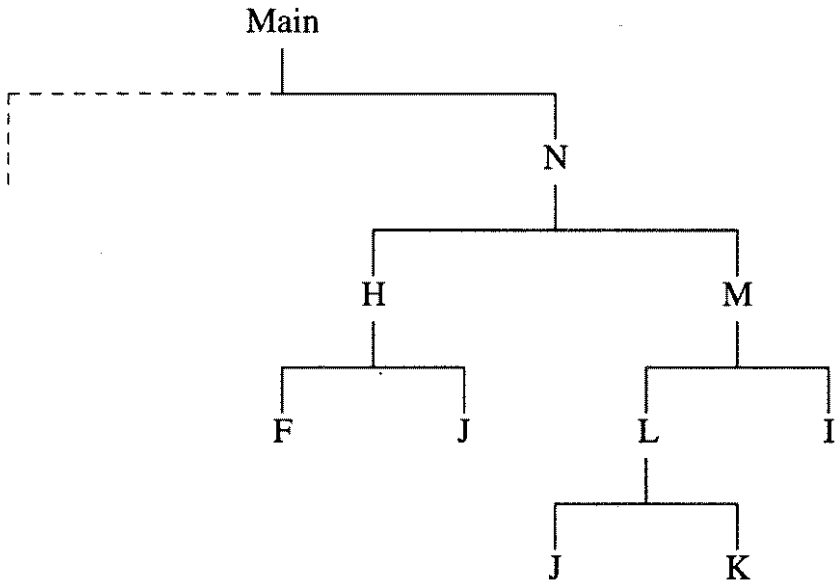


Figure 3-3. Correct Submodel Structure

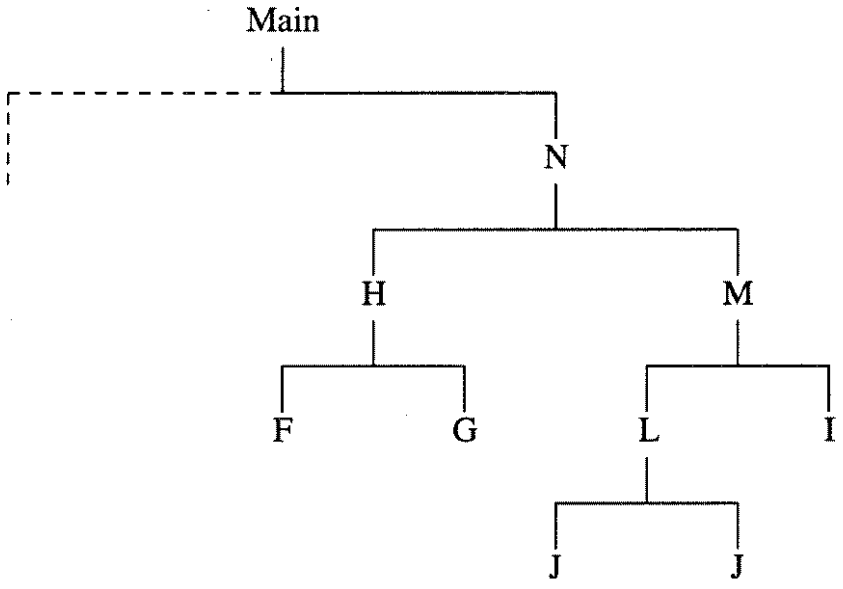


Figure 3-4. Incorrect Submodel Structure

3.1.2 Submodel Structure

A submodel, as we have seen in the previous section, may contain further submodels. In addition to further submodels, it may also contain information regarding the physical structure of part of the overall model. Such information is known as an explicit definition, and will be described in detail in the next section.

The submodel structure takes the form:-

```
$MODEL name
      Submodel definitions
      Explicit definition
$ENDMODEL name
```

Any submodel definitions must be included before the explicit definition; any number of submodels may be included. The included submodels are each contained within $MODEL and $ENDMODEL delimiters, and each of these may be implicitly or explicitly defined as required.

3.1.3 Explicit Submodel Definitions

While it is assumed that the reader of this manual is familiar with the network method of thermal analysis, a brief description of the general principles will be given here.

The component in question is divided into a finite number of cells, with imaginary points usually at the centres of the cells called nodes. All physical properties are considered constant across the cell at any instant in time, and the node is assumed representative of the cell. The next step is to describe the initial properties of the node, i.e. temperature, capacitance, heat sources, area, absorptivity, and emissivity. The node is also given an integer number and character label for referencing purposes.

The linking of the nodes is then described; such a link is known as a conductor. This describes the resistance to heat flow between the nodes, and may be due to conduction, convection, radiation, or any combination of the three.

The above is a very simple overview of the network method. An ESATAN input file must contain the above information in order to construct a database. Optionally it may also include user-defined constants and arrays, and a section describing the type and control of the solution procedure.

An explicit submodel definition is one where the user "manually" follows the above procedure, entering information into the submodel **data** and **operations blocks**.

Data blocks consist of:-

```
$LOCALS
$NODES
$ALIAS
$CONDUCTORS
```

$CONSTANTS
$ARRAYS
$EVENTS

Operations blocks are:-

$SUBROUTINES
$INITIAL
$VARIABLES1
$VARIABLES2
$EXECUTION
$OUTPUTS

These two sets of blocks are the subject of detailed discussion in Section 3.5 and Section 3.6.

The data blocks contain the numeric and topological model details. $NODES contains the nodal information, i.e. temperature, capacitance, heat sources, absorptivity, emissivity, and area, for each node defined within the submodel. $ALIAS contains node aliases - associating names with node numbers. $CONDUCTORS describes how the nodes are connected, and the value of the conductance between nodes. Data in $NODES and $CONDUCTORS may be defined as literal values, variables, variable expressions, or indeed user-defined functions. $CONSTANTS and $ARRAYS are provided in order to define variables and arrays. The contents of these blocks are available to the user such that values may be changed within the operations blocks hence effecting non-linear or time dependent features during solution. $LOCALS, however, is intended for constants which are local to the data blocks only, and therefore cannot be changed during solution. Their principal usage is in the assignment of repetitive initial data. $EVENTS enable the user to specify exact times during solution at which time step or output intervals are required.

The operations blocks describe solution and control information. $SUBROUTINES contains user-defined subroutines which may be called from other operations blocks. As the name suggests, $INITIAL is performed once, at the start of solution, whereas $VARIABLES1 is performed at each iteration, and $VARIABLES2 at each time step. $EXECUTION contains information regarding solution control, i.e. calls to subroutines providing specific solution methods, $OUTPUTS contains calls to routines provided for the printing of results. The information that the user places in the operations blocks is in the form of Mortran statements – a language very similar to Fortran 77. Reference to data block entities may be made, in particular node properties, conductances, constants and arrays may be given reassigned values or referenced in expressions.

3.1.4 Implicit Submodel Definitions

There are two forms of implicit submodels; Non-Preprocessed Models and Previously Preprocessed Models. Note that implicit models cannot contain reference to an Analysis Case Definition (ACD) file.

Non-Preprocessed Models.

Here an explicit model definition, contained within the user's file base, is copied by the preprocessor. The model acts as a template, but with the substitution of actual values for particular variables in the template.

This type of definition takes the form:-

```
$MODEL name
      $ELEMENT element name
      $SUBSTITUTIONS
      substitution values
$ENDMODEL [name]
```

The $ELEMENT keyword is used to copy an external input file, which must be contained in either the file ELEMSYS.DAT or in the global data file (See Section 3.8.3 for details). These files have to be contained within the user's file base. The substitution of explicit values in the element is carried out within the $SUBSTITUTIONS block, which immediately follows the $ELEMENT keyword.

Previously Preprocessed Models.

This form of implicitly defined model involves the copying of an already preprocessed model. The definition of this type of implicitly defined model takes the form:-

```
$MODEL name
      $implicit-keyword name
$ENDMODEL [name]
```

In this type of definition there are three forms of the implicit-keyword, depending on where the model is to be copied from.

a. **$EXTERNAL**

   The $EXTERNAL keyword is used to copy models from an external source, i.e. external to the current model. Such models must be resident within the user's file base as a preprocessed Model Database (MDB). It must be noted that it is only the MDB file (*name*.MDB) that is copied, the generated Fortran file not being required. Copying of a submodel from an external model is allowed, in which case the full submodel concatenation must be given, including the top level model name.

b. **$REPEAT**

   $REPEAT is used to copy a submodel which is already within the model being processed. The repeated model must have been completely defined at the time of use in a $REPEAT command, i.e. its $ENDMODEL line must have been processed. If the model to be copied does not have a unique name within the model, then a concatenated name must be given in sufficient detail to indicate the intended submodel correctly.

c. $VIRTUAL

The third implicit including command is $VIRTUAL. This is used to include one of a class of models known as virtual models. These are models which are defined within the current input file, and are regarded as part of the overall model. The method of defining a virtual model is described in Section 3.4.2. Such models have no position within the submodel hierarchy, and are not included in the solution. They are used to define the other submodels through the $VIRTUAL keyword, the submodels thus defined being active within the solution. A virtual model referenced in this way must be defined within the current model.

It should be clear that the copying of a model to another in an implicit submodel definition in no way affects the source model, it may be used again as a model in its own right or as the basis of some other implicit submodel definition.

The submodel resulting from an implicit definition is known by the name given on the $MODEL line. The name of the submodel from which it is derived disappears. Any subsequent changes which may take place upon the original included model will have no effect upon the implicitly defined model.

3.1.5 Linking of Submodels

The submodel structure facility of ESATAN and methods of defining submodels have been described, but each submodel is as yet physically independent of all others. Two methods of joining submodels are provided:-

Supernodes.

These are described in the $NODES block of a model that contains the submodels to be joined. The user specifies one or more nodes in each submodel as a constituent of the supernode, the effect being to merge the nodes together. The user may define the properties of the node in the same way as conventional node definitions. The default upon non-definition of properties is the same as that for conventional nodes, i.e. zero. Nodal properties of constituent nodes are ignored and in effect these nodes are removed from the solution. The temperature **must always** be defined.

Supernodes are most commonly defined with two constituent nodes, although any number may be used, even one. Conductors previously connected to the constituent nodes are connected to the supernode before the start of the execution of the operations block.

The user cannot refer to the constituents of a supernode at a level higher than that of the supernode definition in the model hierarchy, as the act of supernoding essentially causes them to disappear from the model.

Note that fluid nodes may not be used as constituents of a supernode.

Inter-model conductors.

These are defined in the $CONDUCTORS block of a model, and can either connect two submodels contained within the current model or the current model to a contained submodel. The conductor is defined in the usual way, but with one or both of the node pair being preceded by a concatenated submodel name.

3.1.6 The Log File

The log file is produced at the conclusion of a preprocessor run. It exists as an ASCII file and contains an echo of the source file, interspersed with any warning or error messages.

Each of the submodels has summary comments generated by the preprocessor appended, giving details of number of nodes, conductors, etc. as appropriate.

Summary information at the end of the log file includes a model hierarchy map.

3.2 Data Format Rules

This section describes the general principle of ESATAN data input. For a detailed description of specific items turn to the appropriate section.

3.2.1 General

a. The allowable character set within ESATAN is an extension of the ANSI X3.9 Fortran 77 character set:-

   i. All numeric and upper case alphabetic characters.

   ii. Special characters:-

	Blank
=	Equals
+	Plus
-	Minus
*	Asterisk
/	Slash
(	Left parenthesis
)	Right parenthesis
,	Comma
.	Decimal point
$	Currency symbol
’	Single quote (apostrophe)
:	Colon

iii. Non-Fortran 77 characters:-

a-z	Lower case alphabetic characters
_	Underscore
;	Semi colon
?	Question mark
#	Hash
@	"at"
!	Exclamation mark
%	Percent
&	Ampersand
"	Double quote
Tab	Tabulation

d. Input consists of lines containing:-

Keywords - these are a set of words, each signifying the declaration of a model, block, or end of model. Keywords start with a $ character.

Comments - comments commence with a # character.

Data - all input other than the above two items.

e. Blanks are treated as terminators of $keywords, $keyword parameters, and numbers, and are significant within character strings.

f. Any lines of input provided between a $MODEL keyword and the next keyword are preserved as the model title block.

g. A hash (#) anywhere on a line indicates that the rest of the line is a comment.

h. Termination of $keyword blocks is signified by the start of the next $block, i.e. there is no specific block termination instruction.

3.2.2 Data Blocks

a. The width of the input line is 255 characters.

b. Definitions are terminated by semicolons (;).

c. Entities within definitions are separated by commas (,).

d. Continuation of definitions on to new lines is permitted without any form of continuation symbol, except in the case of literal character strings which require a double slash (//). This is permitted at the end of the first line or beginning of the second.

e. Free format is allowed in the assignment of numbers.

f. The $NODES block if given must always precede the $ALIAS block.

g. Expressions involving arithmetic operations follow strictly the rules of Fortran 77. In particular, exponents are denoted by an upper-case "E" or "D" (e.g. 1.23E4), and it is incorrect syntax to have two consecutive operators (e.g. 4.56 ** -1.0 is not allowed - brackets should be used as: 4.56** (-1.0)).

h. Only one $LOCALS, $ALIAS, $NODES and $CONDUCTORS block may be specified in each submodel, but multiple $CONSTANTS and $ARRAYS blocks are allowed.

3.2.3 Operations Blocks

a. The width of the input line is 255 characters, and a Fortran format is used, i.e. columns 1 to 5 reserved for statement labels, column 6 for a continuation character, and 7 to 255 for a statement.

b. To be consistent with Fortran 77 the maximum allowable Mortran statement length is 20 lines (i.e. start line plus 19 continuation lines).

c. Tabulation ('tab') characters are replaced by eight spaces in order to ensure consistency with the Fortran 77 standard.

d. The $SUBROUTINES block, if given, must be the first of the operations blocks.

e. Expressions involving arithmetic operations should follow the rules of Fortran 77. In particular, exponents are denoted by an upper-case "E" or "D" (e.g. 1.23E4). Although both the pre-processor and the compiler will allow two consecutive operators (e.g. 4.56 ** -1.0), the result is compiler-dependent and such expressions should be avoided by using brackets (e.g. 4.56 ** (-1.0)).

3.2.4 Input Features

The following features are incorporated into ESATAN in order to aid data input.

a. **FOR...DO loops**

These are available in the $NODES and $CONDUCTORS blocks for automatic generation of nodes and conductors.

This format is:-

FOR KL*n* = *i-exp* TO *i-exp* [STEP *i-exp*] DO
statement
.
.
.
END DO

KL*n* signifies a local index. These are described in Section 3.5.1

The term *i-exp* signifies "integer expression", and in the present context means an arithmetic expression involving literal integer values and predefined integer local constants.

The loop is particularly useful for the generation of nodes and/or conductors with identical definitions.

FOR...DO loops may also be used to assign integer array element values.

Note that the terminating END DO must be the last item on a line.

b. **Functions**

Single-statement functions may be defined and referenced at any point in the data blocks.

The format is:-

[*Type* *] FUNCTION * *uname* [(*arg*[,*arg*...])] = *statement*;

Type is optional and can be DOUBLE PRECISION, INTEGER, or CHARACTER. If the latter, then a length may also be given, i.e.

CHARACTER [**length*] * FUNCTION * *uname*

If the length is not given, the default is taken from the character string returned. *Uname* is a user chosen name, following the Fortran variable naming convention, i.e. it has a maximum of six characters, the first character must be alphabetic, the remaining characters may be a mixture of alphabetic and numeric characters. It may not be one of the ESATAN "reserved" names, an attempt to use one of these will generate an error at the definition stage. Reserved names are itemised in Appendix F.

One-line function definitions are translated by the preprocessor into complete function definitions and are available in all operations blocks.

Due to the conversion of the ESATAN library to double precision from version 4.1, all one-line REAL function definitions are converted to DOUBLE PRECISION by the preprocessor.

Functions may also be defined in the $SUBROUTINES block, in which case they are not restricted to single statements.

The content of a function statement can be any Mortran expression, but must not contain any local constants. If reference to other functions is made within the statement then they must be predefined functions.

The default type for a function is taken from the name given, and follows the Fortran default rule for integers (I-N) whilst all others are implicitly defined as double precision. This rule also applies to function arguments.

c. **Variables of Undefined Value**

Any ESATAN entity within the data blocks may be set as undefined by assigning it equal to the character U. Thus:-

```
D20, T = U, C = U, QI = 30.5;
```

is a valid diffusion node definition with both temperature and capacitance undefined (see Section 3.5.2 for node definitions).

d. **Unit Designators**

The user may ensure the consistency of his models by constructing them with the default values for TABS, STEFAN, GRAVX, GRAVY, GRAVZ and PABS and storing and calculating all temperatures in °C.

In order to ease the effects of this restraint on users wishing to make use of this facility a system of unit designators has been introduced for the specification of temperatures. Any literal value can have a temperature unit designator appended to it to enable values to be specified in Kelvin, Rankine or Fahrenheit. Hence:

```
D10 , T = 72.5[K] ;
```

would lead to the value 72.5K being read, and stored in °C as -200.65. The available designators are:

[C] Celsius
[F] Fahrenheit
[K] Kelvin
[R] Rankine

These may be appended to literal values throughout the input file.

Two additions to the ESATAN syntax further ease the task of specifying unit designators:

i. $MODEL card-defaulting:

```
$MODEL MODELNAME , TEMP=K
```

implies that all literal temperature definitions in the $NODES block are in Kelvin

ii. Array definitions of x,y-pairs:

```
CONDXX(2,2)[K,.] = 0.0 , 100.0 , 100.0 , 50.0 ;
```

implies that all x-values be interpreted as being in Kelvin. Similarly, the unit designator [.,R] would imply that the y-values are in Rankine.

Two important points should be noted:

i. The temperature units determined by TABS and STEFAN must be the default of °C for unit designators to be used. If either TABS or STEFAN have been reset in the $CONSTANTS block, and unit designators used, then an error message will be provided and preprocessing will fail.

ii. When unit designators are used, the literal value is converted to °C and stored in °C. All calculations and results are similarly calculated and output in °C.

c. **$INCLUDE**

The $INCLUDE facility enables the contents of an external file to be included in a user's input file - analogous to the #include directive in the C programming language. The format is

```
$INCLUDE "filename"
```

3.3 Model Delimiters

The model-delimiting $keywords $MODEL and $ENDMODEL were introduced in Section 3.1.1 as the introduction and termination of a submodel definition.

3.3.1 $MODEL

Thermal Models

The complete $MODEL line is:-

```
$MODEL name    [,REAL | VIRTUAL][,GLOBALFILE = myglobalfile]
               [,STANDARD | VCHP][,TEMP = tdesig]
               [,ACDFILE]
```

where *name* is a user chosen model name of up to 24 alphanumeric characters. The alphabetic characters can be upper or lower case, and the underscore character is allowed. The first character should be an alphabetic character. Two significant points should be noted:

i. In order to generate the model files, ESATAN uses the first 8 characters of the model name. Hence, in order to avoid potential file name clashes the user should ensure that model names are unique in the first 8 characters.

ii. The model name is considered to be case insensitive – thus, whilst users can use upper- and lower-case characters, internally ESATAN stores the model name in upper case. This arises from the fact that some systems have case-insensitive file naming, and hence two model names which differed only in the case of the characters would generate clashing model files.

Examples of model names are

```
$MODEL SOHO_Pay_Load_Module, GLOBALFILE = soho.gbl
$MODEL StarTracker
```

Five further optional parameters are available. The first specifies the model type, which may be real or virtual. The usage of virtual models is discussed in Section 3.1.4. The definition of a virtual model is made by setting the parameter VIRTUAL following the model name. Note that the default parameter is REAL.

The second parameter specifies the name of the global file. The global file contains data which is common to all submodels in the model - user defined node entities, user library routines and user elements. It is discussed fully in Section 3.8.

The third parameter describes the type of model. Those currently available are STANDARD and VCHP. The latter is used to define variable conductance heat pipes; a full description is given in Section 3.9. The default model type is STANDARD.

The fourth parameter denotes the temperature units to be taken by default in the model: *tdesig* may take the value C, F, K, or R for Celsius, Fahrenheit, Kelvin, or Rankine, respectively; see the previous section.

The final parameter, ACDFILE, indicates that there is an Analysis Case Definition (ACD) file associated with this model, the name of which is assumed to be NAME.acd. See Section 3.10 for more information on the ACD file.

A space must separate $MODEL from the model name, and a comma must follow the model name if parameters follow. The optional parameters can be given in any sequence.

Fluid Models

The full definition of the $MODEL card for FHTS models is:-

```
$MODEL name [,REAL | VIRTUAL][,GLOBALFILE = myglobalfile]
          [,TEMP = tdesig][,ACDFILE][,FLUID = fname][,STATE =
          = fstate]
```

The first four parameters are as described for thermal models. Options for the FLUID parameter, which defines the fluid type, are presented in Section 4.2; if the fluid type is not specified for the current model on the $MODEL line, then it must be given for each fluid node with the entity FT (Section 3.5.2). The final parameter, STATE, defines which variables are used to describe the initial state for fluid nodes in this model/submodel. The options depend upon the state of the fluid (single- or two-phase), the full list available being:-

P&T	-	Pressure and temperature (default)
P&FE	-	Pressure and enthalpy
P&VQ	-	Pressure and vapour quality
T&VQ	-	Temperature and vapour quality

(The order of the variables is unimportant; e.g. STATE=T&P is also valid.) The state definition specified by STATE can be overridden via the nodal entity FST (Section 3.5.2). If STATE is not given then P&T is assumed.

3.3.2 $ENDMODEL

The complete $ENDMODEL line is:-

```
$ENDMODEL [name]
```

The model *name* is optional and would only be given to aid source file clarity. If given, the model name must be separated from $ENDMODEL by a space and should be identical to the name given on the $MODEL card. If a different name is detected, the preprocessor corrects the name and issues a warning message.

3.4 Implicit Definitions

The implicit model definition was introduced in Section 3.1.4 which described two conceptually different forms: the copying of non-preprocessed models and the copying of previously preprocessed models. Note that an Analysis Case Definition (ACD) file cannot be used in conjunction with an implicit model definition.

3.4.1 Non-preprocessed Models

$ELEMENT

The $ELEMENT keyword definition is:-

```
$ELEMENT name
```

A space must separate $ELEMENT and the model name. As explained in Section 3.1.4 the $ELEMENT form of implicit submodel definition copies an explicit submodel definition which is then preprocessed. The model definition must be present in either the file ELEMSYS.DAT or the global data file specified on the $MODEL line. The model is parametrised through the $SUBSTITUTIONS keyword.

Various common thermal models are provided for use in this way as standard, and are contained in the file ELEMSYS.DAT. See Chapter 5 for details.

Users may describe thermal or fluid models of their own to be included with the $ELEMENT keyword and these should be contained in the global data file. See Section 3.8.3 for details.

3.4.2 Previously Preprocessed Models

The $keywords for this form of implicit model definition were introduced in Section 3.1.4 and describe three methods of defining a submodel by copying all or part of a previously preprocessed model.

$EXTERNAL

The $EXTERNAL keyword definition is:-

```
$EXTERNAL name
```

A space must separate $EXTERNAL and the model *name*. The model referred to must exist as a preprocessed Model Database (MDB) in the user's file base, but not contained within the current model.

For example:-

```
$MODEL SPACECRAFT
$EXTERNAL SPACESHUTTLE
$ENDMODEL SPACECRAFT
```

This creates a model, named SPACECRAFT, using the existing, previously preprocessed model, SPACESHUTTLE as a basis. Note that the new model, SPACECRAFT, has a completely separate existence from SPACESHUTTLE, and furthermore, SPACESHUTTLE is unaffected by its usage in the definition of the new model.

It is possible to copy a submodel of some larger external model using this keyword. In that case, the full concatenated submodel name must be given, from the main model name (level zero) down to the required submodel.

An example is:-

```
$MODEL HEATPIPE
$EXTERNAL MODULE:SECTION2:RACK10:FRONT:HEATSINK
$ENDMODEL HEATPIPE
```

In this example, model HEATPIPE is created from a submodel of the external model MODULE. The heatpipe model within MODULE is known as HEATSINK, this is contained within FRONT, within RACK10, within SECTION2, the latter being a direct submodel of MODULE.

$REPEAT

The complete $REPEAT line is:-

$REPEAT *name*

where a space must separate $REPEAT and the model *name*. The model referenced must be a submodel within the current model, and its definition must be complete at the time it is referenced. Concatenated submodel names may be used if necessary in

order to identify the required submodel uniquely. Only sufficient of the concatenation to identify the submodel is necessary, although it is not an error to give a longer concatenation (provided that all submodels within the concatenation are defined).

$VIRTUAL

The $VIRTUAL definition is:-

$VIRTUAL *name*

A space must separate $VIRTUAL and the model *name*. The model referenced must have been defined as a virtual model within the current model. It is not possible to access external virtual models.

Submodels within virtual models may be referenced, in which case the full concatenation from the main (level zero) model down to the required submodel must be given.

The $VIRTUAL keyword may be used within the definition of either real or virtual models.

3.5 Data Blocks

The data blocks provide the physical description of the model; its numerical characteristics and topology are described to ESATAN through these blocks.

The data blocks consist of:-

$LOCALS
$NODES
$ALIAS
$CONDUCTORS
$CONSTANTS
$ARRAYS
$EVENTS

The user may input them in any order, provided that $NODES precedes $ALIAS and $CONDUCTORS. Only one $LOCALS, $NODES, $ALIAS, $CONDUCTORS and $EVENTS block may appear in a submodel, but multiple $CONSTANTS and $ARRAYS blocks are allowed.

3.5.1 $LOCALS

This block is used to define **local constants**. These are of type integer, real, or character, and are provided to simplify input. They are available to the data blocks and to the solution program.

Local constants may be given any alphanumeric name of up to eighteen characters, optionally including underscores (_). The type may be defined in one of two ways:-

a. Individual type definition:-

   This is of the form

   *type*uname = exp*;

   where *type* is either REAL, INTEGER, or CHARACTER. The default upon non-declaration of type is to assume the type of the value assigned to the constant. *exp* is an expression of literal values or local constants.

   The user should not rely on this default, however, since a warning message will be issued by the preprocessor at each occurrence.

b. Block definition:-

   Within the $LOCALS block, type blocks can be defined using the keywords $REAL, $INTEGER, and $CHARACTER. These keywords must appear on a new line which contains no other input. All constants defined following such a keyword are presumed to be of that type, unless given an individual type definition. The end of the type block is signalled by another $keyword line.

Local constants may be assigned to expressions, in which case mixed type expressions are not allowed. A mismatch of constant name and expression will cause the defined type to be taken, and a warning message to be issued.

Rules to be observed in defining and using local constants are:-

a. They may be defined only in the $LOCALS blocks.

b. They must be defined before they can be referenced.

c. They may be defined equal to expressions involving literal values and other local constants of the same type, provided that the latter have already been defined themselves.

d. They may be referenced in the data blocks and in the operation blocks. The log file retains all references to local constants, except for character local constants when used to reference a concatenated submodel name and when an integer local constant is used to replace a node number. In these cases, the reference is not replaced by the literal character string or integer.

A typical user-supplied $LOCALS block may take the form:-

```
$LOCALS
#
$INTEGER
```

```
I_am_an_integer = 16 ; IL15 = 19 ;
SUM = (I_am_an_integer + IL15) / 2 ;
REAL * I_am_a_real_no = 200.0;
$CHARACTER
CMODEL = 'ENGINE : PISTON : CONROD' ;
#
```

Note that the resulting value of SUM would be integer 17, as per Fortran conventions.

The local constants are not retained after preprocessing the model.

Another kind of local constant is available, specifically to perform the function of DO-loop indexing. This is known as a **local index**, and has the form KL*n* where *n* is an integer. Rules for defining local indices are:-

a. They may be defined anywhere in the data blocks, but retain such definitions only until the end of the block in which they are defined. After this point, they assume the undefined status.

b. They must be defined before they can be referenced.

c. They may be defined equal to expressions involving any combination of literal integer values, integer local constants, and other local indices, provided that the latter two have already been defined.

d. They may be redefined within the data blocks.

e. A maximum of 50 may be defined in a model.

Typical usage is:-

```
FOR KL1 = 1 TO 10 DO
KL2 = KL1 + 1 ;
GL(KL1 , KL2) = 5.6 ;
END DO
```

After preprocessing, all references to local indices are removed.

3.5.2 $NODES

The $NODES block contains information regarding network parameters derived from physical properties and dimensions of each of the finite cells of which the model is comprised.

Thermal Nodes

A thermal node definition takes the form:-

D|B|X*n* [= *string*], [T =] *r-exp* [,[C =] *r-exp*] [,[QA =] *r-exp*]
[,[QE =] *r-exp*] [,[QI =] *r-exp*] [,[QS =] *r-exp*] [,[QR =] *r-exp*]
[,[A =] *r-exp*] [,[ALP=] *r-exp*] [,[EPS=] *r-exp*] [,[QAI=] *r-exp*]

[,[QEI=] *r-exp*] [,[QSI=] *r-exp*] [,[FX=] *r-exp*] [,[FY=] *r-exp*] [,[FZ=] *r-exp*];

where the node number *n* is a positive integer up to 10 digits long.

Three thermal node types are available within ESATAN. These are diffusion, boundary, and inactive, symbolised as **D**, **B**, and **X** respectively.

A boundary node is used to represent a mathematical boundary to the problem, the temperature being prescribed by the user as a fixed value, or one varying with respect to time, or possibly some other quantity. Boundary node temperatures are not changed by the solution.

A diffusion node is one whose temperature will be calculated during solution. However, the temperature must be given an initial value. This may be a true value (e.g. at the start of a transient solution), or an estimated value (e.g. at the start of a steady-state solution).

Inactive nodes are ignored by solution routines, but may be re-activated at any point during the solution by library functions. Similarly, "active" nodes (i.e. diffusion or boundary) may be made inactive during solution. An inactive node is effectively removed from the model, as far as the thermal solution is concerned, although all of the nodal properties may still be referenced.

Nodes are defined by the appropriate type symbol (D, B, or X), followed by a user-chosen reference integer. This may be either a literal value or a local index (see Section 3.5.1). The node can optionally be assigned a literal character string, known as the node label, having a maximum length of 24 characters.

The temperature must always be given, either as a literal value or as an expression.

Parameters following the temperature definition are all optional, and are:-

C	=	capacitance
QA	=	total albedo heat source
QE	=	total earth heat source
QI	=	total internal heat source
QS	=	total solar heat source
QR	=	total rest, or other, heat source
A	=	area
ALP	=	solar absorptivity
EPS	=	infrared emissivity
QAI	=	incident albedo heat source
QEI	=	incident earth heat source
QSI	=	incident solar heat source

FX
FY
FZ } cartesian coordinates

These quantities may be given as literal values or variable expressions written in Mortran. The default value upon non-declaration is zero.

Node definitions may be given in any order, and the preprocessor will sort them into ascending reference number order, regardless of type. Since nodes are identified by their reference numbers, this must be unique within a particular submodel. (Note that since node types are not regarded separately, a diffusion node and a boundary node cannot have the same reference number.)

Parameters within node definitions may be defined without each full assignment statement if desired. The left-hand side of the assignment may be omitted for each entity, leaving a string of values or expressions separated by commas. In this case, the order of entities within a definition must be given as above. It is also allowable to commence a definition in this implicit fashion, and then revert to an explicit definition part of the way through the node definition. In this case, the rest of the definition must be given explicitly. For explicit definitions, entities may be given in any order.

The linking of submodels by supernode definition was described in Section 3.1.5. The supernode definition is similar to the standard node definition, except that the nodes to be merged are shown as a sum immediately before the temperature definition. The definition of a supernode is therefore:-

D|B|X*n* [= *string*] = *cname:n1* [+ *cname:n2* ...], [T =]*r-exp*, etc.

Nodes from lower-level submodels are used as constituents of the supernode. Any number of constituent nodes may be merged. The constituent nodes must be defined in submodels contained within the current model. The type of the supernode is determined by its definition, and is irrespective of the types of its constituents. The temperature must be defined. All other nodal quantities are optional, and will default to zero if not given explicitly.

A typical $NODES block might appear as:-

```
#
#
$NODES
#
D10= 'TELESCOPE', T=20.0, C=0.5,  QA=0.0, QE=0.0, QI=5.0,
     QS=0.0, QR=1.2, A=0.01, ALP=0.6, EPS=0.6,
     QAI=0.6, QEI=1.2, QSI=0.88;        #DIFFUSION NODE
#
D20= 'LOUVRE', T=0.5, C=CAPFN(T20), QS=1.3,
          QR=0.6;                       #VARIABLE CAPACITANCE
#
D23, T=26.67, C=(RHOX*VOLNOD/2.0)*NODFNC(1,SPECHT,1);
```

```
                                        #NODAL ENTITY FUNCTION
#
D25= 'INSTRUMENT PANEL', 50.0, 0.75, 0.5, 0.05, 5.3,
                         5.1, 1.05, 0.12, 4.4E-01, 0.55;
#
B30= 'NOSE CONE', TCONE;                #BOUNDARY NODE
#
D15, 29.6, 28.0E-02, 0.98, QI=INTHT(JREG, T15);
     #
D99= 'SUPERNODE N15' = EXOSAT:SUB1:10 +
          TELSAT:SUB1:GRXZ:56,T=20.,
          QI=56.0, QR=3.78;              #SUPERNODE LINK
#
#
```

The diffusion node D10 is an example of a fully explicit node definition. Node D20 is also explicit, but not all entities are defined. Note that the capacitance of this node has been assigned to a function, CAPFN, giving variable capacitance with temperature. Definition of such functions is described in Section 3.2.4.

Node D23 is an example of a node definition where the capacitance is defined by the nodal function NODFNC (see Section 6.4.9). This function provides the user with a short-hand way to interpolate over, say, a specific-heat array SPECHT, without having to specify the nodal temperature T23 in the equation.

Node D25 is an example of a node definition without full assignment statements. B30 is a boundary node, with the temperature assigned to a user constant called TCONE. Node 15 commences with an implicit input as far as QA and then switches to explicit with a definition of QI assigned to a function.

D99 is a supernode definition, combining node 10 of submodel SUB1 with node 56 of submodel GRXZ. Note that T, QI and QR are given explicitly, whilst all other entities will assume the default values of zero.

Fluid Nodes

The full fluid-node definition is:-

F|J|K|R|H|*n* [= *string*] [, [A =] *r-exp*], [FD =] *r-exp* , [FL =] *r-exp*, [P =] *r-exp*,
[[FE =] *r-exp*,] [T =] *r-exp* [,[FF =] *r-exp*] [,[FQ =] *r-exp*]
[,[FM =] *r-exp*] [,[FH =] *r-exp*] [,[FR =] *r-exp*] [,[FX =] *r-val*]
[,[FY =] *r-val*] [,[FZ =] *r-val*] [,[FT =] *z-val*] [,[VQ =] *r-exp*]
[,[FRG =]*z-exp*][,[FLA =] *r-exp*] [,[VOL =] *r-exp*] [,[PHI =] *r-exp*]
[,[FW =] *r-exp*][,[CMP =]*r-exp*] [,[VDT =]*r-exp*][,[FST =]*z-val*];

where *n* is a positive integer up to 10 digits long.

The 'status' of a fluid node is symbolised by a character **F**, **J**, **K**, **R** or **H**. F-nodes are the general, diffusive fluid nodes, whilst J-, R- and K-nodes are different types of boundary node. Nodes of type H are specialised diffusive cells, known as arithmetic nodes.

F-nodes are solved for both thermally and hydraulically at solution time. For the single-phase solvers, temperature and pressure are the primary variables; for the general, two-phase solvers, enthalpy and pressure are primary (see Section 6.15).

A J-node has a user-prescribed pressure, either constant or varying with some quantity such as time. A node of type K provides a thermal boundary, i.e. fixed temperature or enthalpy, as appropriate. An R-node has both pressure and temperature/enthalpy fixed.

The arithmetic or H-node status is similar to F-type but the node is assumed to be of zero volume and hence is always thermally and hydraulically in balance; i.e. a steady-state solution is performed on the node.

A fluid node may not have the same node number as any other node within the same $NODES block. The user may enter the nodes in any order, and thermal and fluid nodes may be freely interspersed. The preprocessor does, however, separate the two types of node and sort them into ascending node-number order within each type.

Fluid-node status may be changed during solution (Section 6.4.30), but it is illegal to change a fluid node status to a thermal node status, or vice versa.

Each node has an optional label of up to 24 characters, being assigned immediately after the node number. The remaining nodal entities are as follows:-

A	-	heat transfer area [m^2]
FD	-	hydraulic diameter [m]
FL	-	length [m]
P	-	static pressure [Pa]
FE	-	specific enthalpy [J/kg]
T	-	temperature [°C, K, °F, R]
FF	-	wall surface roughness [m]
FQ	-	internal heat source [W]
FM	-	mass source rate [kg/s]
FH	-	specific enthalpy of mass source [J/kg]
FR	-	temperature of mass source [°C, K, °F, R]
FX, FY, FZ	-	cartesian coordinates [m]
FT	-	fluid type [-]
VQ	-	vapour quality [-]
FRG	-	predicted flow regime [-]
FLA	-	flow area [m^2]
VOL	-	nodal volume [m^3]
PHI	-	relative humidity (AIRW only) [-]
FW	-	vapour quality of mass source [-]; specific humidity of mass source (AIRW only) [-]
CMP	-	compliance—rate of change of unit volume w.r.t. pressure [Pa^{-1}]

VDT - rate of change of volume with respect to time [m^3/s]
FST - fluid state descriptor [-]

Entities may be defined implicitly, as with thermal nodes, in which case the order is as given in the above list. In general, each entity may be assigned a literal value, Mortran references, or expressions involving either or both of these.

The initial fluid state **must** be defined. Normally this is done via the entities P and T (pressure and temperature). However, it may be more appropriate—especially for a two-phase model—to give the state in terms of enthalpy (FE) or vapour quality (VQ); in this case either the fluid state descriptor FST, described below, should be used, or else the STATE parameter on the $MODEL line should be set (Section 3.3.1). During simulation all fluid-state entities are updated from the primary variables. When using a single-phase solution module the user should ensure that the fluid is not initialised in a two-phase state.

Two other entities **must** be specified for each node, relating to the geometry. They are: hydraulic diameter, FD; and length, FL.

The hydraulic diameter, FD, is used internally to evaluate pressure drops and heat transfer coefficients. It is also used in the default calculation of flow area, FLA (see below). The preprocessor can only calculate this default value when the hydraulic diameter is defined explicitly, i.e. not using Mortran; thus, **FD must be given using a literal expression and/or local constants**. The relation between flow area and hydraulic diameter is given by FD=4×FLA/(wetted perimeter); for non-circular pipes, FD should be defined accordingly.

The nodal length, FL, determines the coarseness of discretisation of the model; it is also used in the computation of frictional pressure losses.

The remaining fluid-node entities are optional, defaulting to zero unless otherwise stated.

Heat transfer area, A, is used only in conjunction with the intrinsic calculation of heat-transfer coefficients (Section 3.5.4). By default, A is set to 4×FLA×FL/FD i.e. whole surface area, allowing for non-circular cross section.

The wall surface roughness, FF, is expressed in units of length. It represents the actual surface irregularity (absolute, not relative) of the internal surface of the pipe wall, and is used in the calculation of the friction factor to give the correct pressure drop in the momentum equation.

A fixed heat source in the fluid is defined by the entity FQ. This may be negative, indicating the extraction of heat. Mathematically speaking, FQ provides a thermal boundary condition to the model.

A mass source—a fixed injection of fluid—is specified by the entity FM, defining a hydraulic boundary condition. It is assumed that the injected fluid is at the same pressure as the node itself. The temperature is given via FR; enthalpy (FH) or

vapour quality (FW) of the source may be specified instead, in conjunction with the fluid state descriptor, FST. The value of FM may be negative, indicating the extraction of fluid at the nodal conditions; in this case, FH, FR and FW are ignored.

The cartesian coordinates of the node—FX, FY, FZ—may be given if gravitational effects on pressure and energy are to be modelled (the gravity vector being specified through the control constants GRAVX, GRAVY, GRAVZ; see Appendix E). Note that within the general or two-phase solution routines, gravity is currently assumed to act in the z-direction only; hence FX and FY are ignored by these solvers.

The fluid type, FT, must be given if FLUID has not been specified on the $MODEL card (Section 3.3.1). If both are given, FT takes precedence. See Section 4.2.1 for the fluid types available in FHTS.

Vapour quality, VQ, may be used in combination with the fluid state descriptor, FST, to define the initial fluid state. In the simulation of moist air (Section 4.4), vapour quality is redundant and so this entity is used instead to hold specific humidity.

The character entity FRG represents the predicted flow regime. It is set during solution to be 'LIQ' (liquid), 'VAP' (vapour) or 'HEM' (homogeneous equilibrium model) depending upon the vapour quality. The user should not alter this value.

The flow area, FLA, is used mainly to determine the fluid velocity. If FLA is not given it is assumed that the pipe is circular; hence FLA defaults to $\pi \times FD^2/4$.

VOL stores the volume of the fluid node and is calculated during solution. Thus, users should only *reference* this entity and not define its value. The volume of the node can be modified during solution as a function of time or pressure via the entities VDT and CMP, respectively. By default the flow area, FLA, is updated if the volume changes. The function VOLST (Section 6.4.33) can be used if it is required that the length, FL, should be updated instead.

The entity PHI, representing relative humidity, is provided for use with the fluid type AIRW, i.e. wet or moist air. See Section 4.4.

FW specifies the vapour quality of a mass source (see FM, above), and is used in conjunction with the fluid state descriptor, FST. In the simulation of moist air (Section 4.4), vapour quality is redundant and so this entity is used instead to hold specific humidity of the mass source.

As mentioned above, the volume of a node can be modified by the entities CMP and VDT. CMP is the compliance, defined mathematically as

$$CMP = \frac{1}{V}\frac{\partial V}{\partial P}$$

where V is the volume of the node and P the pressure. Compliance causes a change in volume depending on the change in pressure:

$$\delta V = CMP \cdot V \delta P.$$

VDT, or "V-dot", is defined as

$$VDT = \frac{\partial V}{\partial t},$$

where t is time, and gives direct control over the volume change:

$$\delta V = VDT \cdot \delta t .$$

VDT has no effect in steady-state solutions. In transients, CMP and VDT give a combined volume variation, thus:

$$\frac{dV}{dt} = \frac{\partial V}{\partial t} + \frac{\partial V}{\partial P}\frac{dP}{dt} = VDT + CMP \cdot V \frac{dP}{dt}$$

As discussed above, the initial fluid state is defined by default via pressure and temperature. However, for two-phase conditions these variables do not uniquely specify the state. To increase the flexibility, therefore, the entity FST is provided to allow the definition to be made via one of the following combinations:-

’P&T’	-	Pressure and temperature
’P&FE’	-	Pressure and enthalpy
’P&VQ’	-	Pressure and vapour quality
’T&VQ’	-	Temperature and vapour quality

(The order of the variables is unimportant; e.g. FST=’T&P’ is also valid.) Saturated conditions can be specified by using either 'P&VQ' or 'T&VQ' and setting the vapour quality to 0.0 or 1.0 as appropriate. FST is also applied to the state definition of a mass source; that is, FR (temperature), FH (enthalpy) or FW (vapour quality), along with the nodal pressure P, will be used accordingly to define the state of the fluid injected at a rate given by FM. A global default for FST can be defined via the $MODEL-card parameter, STATE (Section 3.1.1). If both STATE and FST are defined then the nodal FST takes precedence; if neither, then ’P&T’ is assumed.

3.5.3 $ALIAS

The $ALIAS block enables named aliases to nodes to be defined. These named aliases can be used in node entity references wherever a node number would otherwise be given. This capability has two main purposes:

a. Increase the readability of the input - instead of control logic being expressed with reference to node numbers, it can be expressed in terms of physically meaningful names.

b. Make the control logic less sensitive to changes in the model - if the node number of a component is changed, then rather than having to change all references to that node throughout the model, only the alias definition needs to be changed.

An alias definition takes the form:-

alias = D|F*n*;

where *alias* is the alias name and conforms to standard ESATAN rules for identifiers (up to 18 case-sensitive characters - alphanumeric and the underscore - starting with an letter), and *n* is a node number. The prefix D must be used for thermal nodes, and the prefix F for fluid nodes.

A typical $ALIAS block might appear as:-

```
#
#
$ALIAS
#
thermocouple1 = D103;
pump_inlet = F5;
#
```

These aliases could then be used as follows:-

```
$CONDUCTORS
...
GL(5, thermocouple1) = ...;
...
M(pump_inlet, 27) = ...;
...
$VARIABLES1
...
IF(T:thermocouple1 .GT. 50.0D0)THEN
...
$OUTPUTS
      CALL PRNDBL('#10-20,thermocouple1', 'T', CURRENT)
```

3.5.4 $CONDUCTORS

The topology of the model is described through the $CONDUCTORS block, in describing which nodes are physically linked.

Thermal Conductors

Conductor definitions describe the heat conductance value and are of the form:-

GL|GF|GR|GV *(n1, n2)* = *r-exp*;

For conductor definitions, *r-exp* is a real expression of literal, local constant, user constant, or function references.

Four conductor types are available, these being linear, fluidic, radiative, or view factor, symbolised by GL, GF, GR and GV respectively.

Linear conductors link two nodes in which the heat flow may be in either direction. Practical examples are conduction through a solid or convection between a solid and a fluid.

Fluidic conductors permit the realistic modelling of heat transfer by fluid (mass) flow, and their conductances are always computed as $\dot{m}c_p$. The node described first in the link is the upstream node, the other node being the downstream node. The upstream node will not be allowed to lose or gain heat through the conductor, even though the temperature of the node will be used to calculate the heat flow. In other words, the solution routine will compute the heat transferred through a fluidic conductor as though it were an ordinary conductor. This heat will not, however, be allowed to enter or leave the upstream node; it will be allowed to leave (or enter) the downstream node.

Radiative conductors are used to define radiation heat transfer; the solution thus treats heat flow within radiative conductors as being proportional to the difference between the fourth power of the absolute temperatures. The user should not include the Stefan-Boltzmann constant in his conductance value: the Preprocessor incorporates this automatically.

View factor conductors may be specified in place of radiative conductors. The user must specify the view factor only as his view factor conductance value, and all view factors describing "line of sight" connections must be given. Note that only one such link between any particular node pair should be specified, repeated definitions will result in the final value being used. Radiative conductors, GR's, are calculated internally (under user control) from the view factors, GV's, by the ESATAN library routine VFAC. This routine calculates all direct radiative conductances and also further conductances due to reflected radiation. View factor submodels are regarded as isolated enclosures by the preprocessor and library. Values of area and emissivity (A and EPS) must be given to nodes used in view factor conductors. Assignment of an area value to a node representing deep space must be to a suitably large value, 1.0E20 is recommended.

The definition of inter-model links is:-

GL|GF|GR *(cname : n1, cname : n2) = r-exp*;

Note that it is not possible to specify a view factor conductor between two submodels. The submodels specified must be contained within the present model, and it is allowable to specify a link between the current model and a contained submodel.

Conductors may be specified between nodes of any type, the nodes of any one conductor not necessarily being of the same type.

More than one conductor may be specified between the same two nodes. These may be of different types, or the same type. In the latter case, the user may make reference to such conductors by inserting a sequence number after the node pair, i.e.

GL|GF|GR *(n1, n2, n3)*

where *n3* indicates the conductor between nodes *n1* and *n2* in order of appearance on the log file.

An example of a $CONDUCTORS block is:-

```
$CONDUCTORS
#
GL (10,11)= 0.35;
GL (10,12)= 0.26;
GF (30,15)= 6.80E-03;
GL (11,12)= CONSS*0.1/0.25; #STAINLESS STEEL CONDUCTIVITY
GR (11,999)= RL500*0.05;
GR (12,999)= RL500*0.05;
GR (30,999)= RL500*0.17;
GL (ENGINE:50,NOZZLE:43)= 1.5;  #JOIN ENGINE TO NOZZLE
```

The first three conductors are conventionally defined linear and fluidic conductors. The fourth is a linear conductor employing a user constant giving the value of the conductivity of stainless steel. Three conductors are then defined giving radiative links between nodes 11, 12 and 30 and node 999. Local constant RL500 is used to aid input and the current value of RL500 will be used to calculate the conductance at this point. For instance, if the value of RL500 is 100.0, GR (1,999) will be stored with value 5.0. Finally, an inter-model link is given, joining the submodels shown with a linear conductor.

The following example demonstrates how a linear conductor is often defined as a function of conductivity vs. temperature. The value is obtained by interpolating over a conductivity array at the average nodal temperature.

```
#
# Use conductivity array CONDX
#
GL (3,4)= INTRP1((T3+T4)/2.0, CONDX, 1)*CSA*DX;
```

The next example enables the user to achieve the same result by calling the conductance function CNDFNC (see Section 6.4.8), but without the need to specify the nodal temperatures T3 and T4. This method aids the user considerably in building a model with many linear conductors.

```
#
# Use conductivity function CNDFNC
#
GL (3,4)= CNDFNC(1, CONDX, 1)*CSA*DX
```

Mass Flow Links

The specification of mass flow links in ESATAN/FHTS serves two purposes. Firstly, it describes the topology of the fluid network in terms of node connections; and, secondly, it provides first-guess values of mass flow rate for steady-state solutions or initial values for transient solutions.

The FHTS numerical solution schemes solve for mass flux or mass flow rate between fluid nodes, and hence the mass flow rate is associated with the fluid link between nodes rather than the fluid node itself. Fluid links must therefore be given between all connected nodes in the loop and are expressed as follows:-

M (*n1, n2*) = *r-exp*;

where *r-exp* is a real literal value or an expression of real literal values.

The specification of the node pair in a particular order does not imply that mass flow is restricted to any particular direction; but the user-defined order is preserved and any subsequent reference to mass flow rate in the link will be positive for flow from *n1* to *n2*, or negative if reversed. When referencing mass flow links node pair order is important and must be as defined otherwise the entity will not be recognised by the preprocessor. Referencing of FHTS quantities, including mass flow rate, is described in more detail in Section 3.6.1.

Fluid Conductances

A fluid conductance determines the resistance to flow between two nodes and refers to the 'fitting loss', the irrecoverable pressure loss as a result of turbulence produced by typical pipe fittings such as sudden expansions/contractions, bends, orifices and valves. This should not be confused with the pipe-wall frictional loss, this latter quantity being taken from the nodal information.

For example, consider the following:-

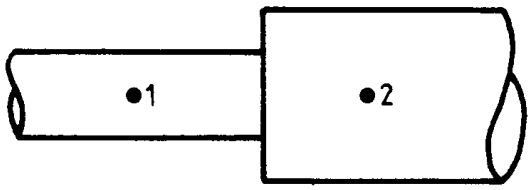


Figure 3-5. Pipe showing sudden expansion

This shows flow between two nodes, *n1* and *n2*. The total irrecoverable pressure loss between these nodes consists of the pipe wall frictional loss and a sudden expansion (i.e. fitting) loss. The latter is described in the fluid conductance value, such that

$$\frac{1}{2}\rho u^2 = G\Delta p$$

where ρ is the density, u is the fluid velocity, G is the fluid conductance value for the fitting and Δp is the fluid pressure drop due to the fitting loss alone. This means that a large conductance value refers to a low resistance and hence a low pressure drop.

Fitting losses are normally given by an expression of the form:

$$\Delta p = \frac{1}{2}k_f \rho u^2$$

where k_f is a tabulated fitting loss coefficient for different types of fittings. So, if

$$G = \frac{1}{2}\rho u^2 / \Delta p$$

then the conductance is $G = 1/k_f$.

For valves, the characteristic is often expressed as:-

$$Q = k_v A\sqrt{2 \cdot \Delta p \cdot \rho}\ ;$$

the conductance then can be shown to be $G = k_v^2$, where k_v is the valve flow coefficient, and a function of the valve opening, and A is the nominal valve flow area. This applies to valves as described in Section 5.3, k_v being expressed as a variable function in order to simulate a control action.

The fluid conductance is expressed as

GP(*n1,n2*) = *r-exp*

where *r-exp* is a real literal, an expression of real literal values and/or Mortran references. Library modules are provided to evaluate pressure loss coefficients for the following:-

i. Expansions/contractions - (ACLOSS)

ii. Pipe bends - (PLBEPI)

iii. Nozzles - (PLNOZZ)

iv. Diffusers - (PLDIFF)

v. Orifices - (PLORIF)

vi. Butterfly valves - (PLBUVA)

vii. Slide valves - (PLSLVA)

See Section 6.4.1 for details.

The fluid conductance is optional; if it is not given, then this implies that the pressure drop is calculated from frictional losses only (if any), which are calculated

by the program using the surface roughness (FF) given in the $NODES block for each node. If it is included, then the pressure drop is calculated from the combined effects of frictional loss and fitting loss.

The actual topology of the loop (i.e. the node connection sequence) is determined by the mass flow link: a GP cannot be defined between two nodes if there is no mass flow link.

Thermal Conductances Between Thermal and Fluid Nodes

The interfacing of FHTS fluid nodes to other nodes of the conventional solid model is naturally done by thermal conductances. Such interfacing is indicated in figure 4-2 where the fluid nodes are shown linked to adjacent wall nodes by thermal conductors. Such links will be convective, allowing heat flow in the direction of the temperature gradient, and therefore linear conductors of the type 'GL' are normally used for this purpose. However, fluid nodes may be linked to solid nodes by other conductors, GR and GF, allowing the description of radiative and 'one-way' heat transport respectively. In the modelling of air/vapour loops GL's are used to define condensing surfaces (see Section 4.4). The pipe-wall nodes may be diffusion, boundary, or inactive in type.

Special FHTS features are available for use with GL conductors which ensure a sound link between the thermal and hydraulic solution. Linear conductances may be defined as:-

$$\mathrm{GL}(n1, n2) = \begin{cases} r\text{-}exp \\ * \\ \mathrm{NUVRE}(n, \mathrm{ARRNAM}) \\ \mathrm{STVRE}(n, \mathrm{ARRNAM}) \end{cases}$$

These four options, in the order shown, are:-

i. User-supplied real literal values or an expression of real literal values and/or Mortran references.

ii. Internally calculated using heat transfer coefficient correlations appropriate to the fluid conditions; see Appendix K.

iii. Interpolated on the user-defined array ARRNAM (containing Reynolds numbers × Nusselt numbers), with properties from node *n*.

iv. As iii., but ARRNAM contains Reynolds number × Stanton number.

The above input syntaxes are available for linear conductors (GLs) only. For radiative and 'one-way' conductors, only the first form, i.e.

GR|GF(*n*1,*n*2) = *r-exp* ,

is available.

The usage of the ESATAN submodelling facility is entirely at the discretion of the user, as with conventional ESATAN models. Either node, both nodes, or neither node, may be in the submodel where the conductance definition is made. In the case of reference to nodes outside of the current submodel in such a definition, the conventional ESATAN rule applies allowing only references to nodes in contained submodels.

No thermal conductance of any type (GL/GR/GF) may be defined between two fluid nodes.

The FHTS thermal conductors are entered by way of the $CONDUCTORS block.

Example:-

```
$CONDUCTORS
#
# Simple fluid loop
#
M(1, 2) = 0.36;
M(2, 10) = 9.4;
M(11, 9) = 9.2;
GP(1, 2) = 2.3;
GP(11, 9) = ACON * 6.3;
GL(100, 11) = 0.36;
GL(200, 9) = M(11, 9) *0.023;
M(10, 11) = 0.42;
```

The mass flow links above define the fluid topology. Note that they also define the 'positive' direction of flow rate. Fitting loss links (GP) are given for two flow links. Linear conductors (GL) are shown connecting thermal to fluid nodes.

3.5.5 $CONSTANTS

This block contains the user and control constants. Multiple $CONSTANTS blocks may appear in a single submodel. User constants are of real, integer, and character type, and the basic difference between them and the local constants is that the user constants are active only in the solution program. User constants may be referenced and/or reassigned in all operations blocks, and referenced in $NODES and $CONDUCTORS data blocks. Hence the user may assign, for example, a boundary temperature to a user constant, and then change the value of the user constant during the solution run in order to change the boundary temperature.

Constant names may be of up to eighteen characters, and include lower and upper case alphabetic characters, digits and the underscore character. The first character must be an alphabetic character. Constant names are case sensitive. The type may be defined in one of two ways:-

a. Individual type definition:-

   This is of the form

*type*uname = exp;*

where *type* is either REAL, INTEGER, or CHARACTER and *exp* is an expression of literals, local constants, user constants and/or function calls. Character user constants can only be defined in the $CONSTANTS block in terms of literal character strings. Expressions cannot be used. The default upon non-declaration of type is to use that of the expression (a warning message will be issued by the preprocessor).

b. Block definition:-

Within the $CONSTANTS block, type blocks can be defined using the keywords $REAL, $INTEGER, and $CHARACTER. These keywords must appear on a new line which contains no other input. All constants defined following such a keyword are presumed to be of that type, unless given an individual type definition. The end of the type block is signalled by another $keyword line. Character user constants can only be defined in terms of literal character strings - expressions cannot be used.

User constants may be assigned to expressions, in which case mixed-type expressions are not allowed. A mismatch of constant name and expression will cause the defined type to be taken, and a warning message to be issued.

ESATAN control constants may be assigned within the $CONSTANTS block in a sub-block headed $CONTROL. These are of predefined type, and a mismatch in assignment will generate a warning message and correction of the value assigned. Control constants are of either global or local type. Globals may be defined in the top-level model only, and are ignored if found elsewhere. Local control constants, if not defined in a submodel, assume the value given in the nearest containing model. Control constants of real, integer, and character type exist, although only real and integer types may be assigned in the $CONSTANTS block. Character-type constants should be assigned within one of the operations blocks.

A full list of control constants is given in Appendix E, together with type, default value, and purpose.

A typical $CONSTANTS block is given below:-

```
#
$CONSTANTS
#
INTEGER*ILNRST= 99;REAL*QWERTY= 15.62;
REAL*AB23= 4.56E03;CHARACTER*ZPROCESS= 'HELLO';
#
$INTEGER
JSTRT= 26; Anint =  56; LOOPMAX= 9999;
a_MAXIMUM= 10000;NewInt = JSTRT * Anint;
#
$REAL    # The following are all real user constants
AB46= 19.3E-06; ASTARTUP= 34.51;
```

```
UB_XSTP =  23.9; htcoef= 22.0;ht2= 35.98;
VaryingFlux = INTRP1(TIMEM, Qarray, 1);
#
$CHARACTER
JTC= 'TIME STEP'; FRXSTP= 'MAXIMUM STEP';
XMXLPS= 'PREVIOUS TEMPERATURE VALUE';
```

In the above example, several definitions are made by the explicit type declaration immediately after the $CONSTANTS line.

Note that if one constant is used to define another in the $CONSTANTS block, then changing the value of the first during a solution run will cause that of the second to be recalculated accordingly. Also, the second one will not have the correct value until $VARIABLES1 has been executed.

3.5.6 $ARRAYS

ESATAN arrays are essentially tables of quantities, stored in a manner closely similar to Fortran array storage, to be used during solution mainly for interpolation. They can be of any size and can be assigned dimensions by the user to give control over the storage form. In common with the convention for the $CONSTANTS block, multiple $ARRAYS blocks may appear in a single submodel.

Arrays follow the same general principle as the user constants in that they are active only within the solution program. In addition they obey the same naming convention as $CONSTANTS. That is; names may be of up to eighteen characters, and include lower and upper case alphabetic characters, digits and the underscore character. A common usage of arrays is to store variable physical properties which are then used during the solution as time and temperature change.

Precisely the same rules are obeyed regarding type declaration for arrays as those for user constant types, i.e. the provision of $INTEGER, $REAL, and $CHARACTER blocks, and the facility to override the block declaration with an explicit definition.

The explicit array definition form is:-

[*type**] *uname* [(*dim* [, *dim* ...])] [/SIZE = *i-val*/] = *val*, ...;

The *type* may be INTEGER, REAL, or CHARACTER, and is optional. If not given, the type will be implied from the current type block declaration, or in the event of no block declaration having been made, from the type of the first data value given. Dimensions are also optional, the default being to regard the array as one dimensional with a length implied from the SIZE parameter or number of elements given, whichever is the larger. SIZE is optional, if not given it will be implied from the dimensions or number of elements given, whichever is the larger. The preprocessor always reserves sufficient space for an array according to the largest space implied by dimensions, SIZE, or number of elements. The maximum allowed number of dimensions is ten. In the case of character arrays, reserved elements are always 24 characters in length.

The elements of the array must be given as a list of literal values separated by commas, and terminated by a semi-colon. The comma may be omitted at the end of a line, however, its presence being implied.

The data fill the array in the Fortran manner, i.e. the left-most elements of the array in the dimension list increment first. A "shorthand" method exists for filling several values of an array with the same constant value. This is:-

m@value

When given at any point within the element list, the next *m* elements will be given the value *value*.

Array elements can be referenced explicitly in the operations blocks and anywhere Mortran expressions are allowed in the data blocks. For example referencing the array MASS_FLOW within conductor definitions:-

```
$CONDUCTORS
 GF(1 , 2) = MASS_FLOW(1) * 0.273 ;
 GF(2 , 3) = MASS_FLOW(2) * 0.173 ;
#
$ARRAYS
$REAL
 MASS_FLOW(5) = 5@1.0;
```

Array elements can also be redefined anywhere in the operations blocks, see Section 3.6.1.

TABLE arrays are provided to give values of a single dependent variable "tabled" against several independent variables, such as a function *V*(*X*, *Y*, *Z*) where the value of *V* is dependent upon the values of *X*, *Y*, and *Z*. Table arrays are placed in a $TABLE sub-block within the arrays block, and **can only be accessed through ESATAN interpolation routines**. TABLE arrays always contain real values. If integer values are given, they are converted to real values by the preprocessor. The SIZE parameter, available in other forms of arrays, is not applicable to table arrays, and no more than three independent variables may be defined.

Three forms of input, illustrated below, are allowed. Depending upon the form of the data input, one of the three choices should be the most efficient in terms of database storage and manual input. Note that X, Y, Z are not the dimensions of the array, they are reference names of the independent variables.

i.

```
V(X, Y, Z)
X = X1,   Y = Y1,  Z,  V,  Z,  V,  Z,  V,  ...,  Z,  V,
          Y = Y2,  Z,  V,  Z,  V,
X = X2,   Y = Y1,  Z,  V,  Z,  V,  Z,  V,  ...,  Z,  V,
   .                              .
   .                              .
   .                              .
X = XN,           Z,  V,  Z,  V,  Z,  V,  ...,  Z,  V;
```

Thus for a range of values with fixed X and Y, pairs of Z and V can be given. When X or Y change, only the changed variable is given, e.g. in the second row, when Y = Y2, X still has the value X1.

NB: This input format may only be used with table arrays which have three independent variables, since the input becomes ambiguous with the other formats if a definition is made for an array with one or two independent variables.

ii.

```
V(X, Y, Z)
Z = Z1,  Z2,  Z3,  ...,  ZM,      # M values
X = X1,  Y = Y1,  V1,  V2,  V3,  ...,  VM,
X = X2,           V1,  V2,  V3,  ...,  VM,
   .                       .
.                     .
Z = Z1,  Z2,  Z3,  ...,  ZN,      # N values
X = X5,  Y = Y5,  V1,  V2,  V3,  ...,  VN;
```

The range of Z values persist here until changed, at which time X and Y must also be given. Until the next setting of Z, X and Y may be changed individually, as in (i) above.

Note that this form of input is a shortened version of the previous one, and is valid for the case when one independent variable has the same range of values for several sets of values in the other independent variables.

iii.

```
V(X, Y, Z)
Z = Z1,  Z2,  Z3,  ...,  ZM,
Y = Y1,  Y2,  Y3,  ...,  YN,
X = X1,  V1,  V2,  V3, ..., VMN     # M*N values
X = X2,  V1,  V2, ..., VMN          # M*N values
Z = Z1,  Z2,  Z3,  ..., ZL,
X = X3,  V1,  V2,  ..., VLN         # L*N values
.
.
Z = Z1,  Z2,  ...,  ZA,
Y = Y1,  Y2,  ...,  YB,
X = X1,  V1,  V2,  ..., VAB;        # A*B values
```

Ranges of Z and Y are given for fixed X, resulting values being given for the product of the numbers of Z's and Y's.

The order of the independent variables is arbitrary, not necessarily as shown above; e.g. in (i) the following would be perfectly valid:

```
X = X1,  Z = Z1,  Y,  V,  Y,  V,  Y,  V,  ...  Y,  V,
```

However, the values of the independent variables must be given in increasing order only. The maximum number of independent variables is 3, and their names may be up to 6 characters long.

Table arrays with one independent variable are defined thus:

```
V(X)
 X1, V1, X2, V2, ..., XN, VN;
```

As can be seen, this format is equivalent to that of a standard 2-dimensional real array; since the latter is more flexible, its use is recommended over the 1-dimensional table array.

The format rules for table arrays are somewhat stricter than those for ordinary arrays. Firstly, the array name and independent variables must be given on the first line of the array definition **without** any array data following on the same line, and **not** followed by an equals sign (=). Secondly, the rule concerning an implied comma at the end of each line is **not** observed, therefore the user must end each line (apart from the last one) with a comma. The table array shown in the example below should make this clear.

A typical $ARRAYS block takes the form:-

```
$ARRAYS
$REAL
     VISC_WATER(2, 19) /SIZE=40/ =
          10.0,  1.3E-04,   20.0, 1.002E-03,  30.0, 7.973E-04,
          40.0, 6.51E-04,   50.0, 5.44E-04,   60.0, 4.63E-04,
          70.0,  4.0E-04,   80.0, 3.51E-04,   90.0, 3.11E-04,
          100.0,  2.79E-04;
#
$INTEGER
 ANGLES(100)=10, 15, 20, 25, 30, 35, 40, 45;
#
$CHARACTER
 Title_names(2)/SIZE=20/='TEMPERATURE',  'TIME';
#
$TABLE
 Enthalpy_Water(P, T)
           P =   1.0,    5.0,   10.0,   15.0,   20.0,
     T=100.,   2676.,   419.,   419.,   420.,   420.,
     T=150.,   2776.,   632.,   632.,   633.,   633.,
     T=200.,   2875.,  2855.,  2826.,  2795.,   853.,
           P =   30.,    40.,    50.,    60.,    70.,
     T=150.,    634.,   634.,   635.,   636.,   636.,
     T=200.,    853.,   853.,   854.,   854.,   854.,
     T=300.,   2995.,  2962.,  2925.,  2885.,  2839.;
```

The real array VISC_WATER in this example contains the dynamic viscosity of water stored paired with temperature in steps of 10 °C from 10 °C to 100 °C. Note that although 20 elements are used, the array is given a size of 40, to allow for future expansion. The integer array ANGLES contains only 8 elements, but 100 are allocated due to the array dimension used. Similarly, the character array Title_names reserves a space for 20 elements through the SIZE parameter.

The table array shown, Enthalpy_Water, is a two-dimensional example: enthalpy of water is given as a function of independent variables pressure and temperature.

2-D Real and 2- and 3-D Table arrays may be defined as deferred. Deferred arrays cannot be defined directly by the user, but are generated by ESATAN-TMS Workbench, and the data values for the array are stored in the Workbench-generated ACD file (See Section 3.10) rather than in the input file. The format for these in the input file is as follows:

[REAL *] *uname(dim, dim)* / DEFERRED / ;

[TABLE *] *uname(X, Y)* / DEFERRED /;

[TABLE *] *uname(X,Y,Z)* /DEFERRED / ;

A deferred array can be used during solution in exactly the same way as a standard array, and can be accessed by all the array library routines.

3.5.7 $EVENTS

This block enables the definition of both time step and output events. A time step event forces the end of a transient analysis time step to coincide with the event time, whilst an output event forces an output interval to occur at the event time. Both time step and output events can be periodic. Periodic events allow an event to repeat at regular intervals.

A typical $EVENTS block takes the form:-

```
$EVENTS
#
$TIMESTEP
My_event = 100.0;           # simple event at 100 seconds
AnotherEvent = 50.0,100.0; # periodic event at 50, 150, 250, ...
                            # seconds
#
$OUTPUT
OutputEvent = 125.0;        # event at 125 seconds causing $OUTPUTS
                            # to be run
```

Events can be given as literal values, local constants, or expressions of literal values and local constants.

Note that events can only be referenced via the functions BEFORE, AT, AFTER and BTWEEN. Events cannot be referenced directly in Mortran.

3.6 Operations Blocks

The operations blocks provide specification and control information to the solution program. The organisation of the execution program is described in Section 6.15. It is very important to note that the complete library is written in double precision and special care is needed when using explicit values in subroutine/functions arguments. Precise information regarding the function and content of each block is given in the following sections.

The operations blocks consist of:-

$SUBROUTINES
$INITIAL
$VARIABLES1
$VARIABLES2
$EXECUTION
$OUTPUTS

Each $keyword may be placed anywhere on the line, providing the $ is the first non-blank character.

Any block may be omitted from the source file if desired, and will have the same effect as supplying an empty block.

The $SUBROUTINES blocks, if given, must be the first of the operations blocks. The relative order of the other blocks is unimportant.

The $EXECUTION and $OUTPUTS blocks are valid in the main model only. To include them in a submodel is not an error, but they will be ignored.

3.6.1 Mortran Language

The language used in the operations blocks is Mortran, which is based upon the Fortran 77 language. Using Mortran, the user refers directly to the entities of the model by the reference integers or names supplied in the data blocks; the preprocessor detects these entities and translates them into Fortran references to the ESATAN numeric database for use in the solution program.

The user may reference "local variables" within an operations block, i.e. a variable not known to the Model Database (MDB), and therefore having no corresponding Fortran reference. Such variables should, however, be type-declared at the start of the operations block in which they are referenced, otherwise the preprocessor will detect them as "unknown" and issue a warning message at each such occurrence. This feature provides additional error checking. The type of variables or functions not type-declared will be taken from the name given. This follows the Fortran default rule for integers (I-N) whilst all others are implicitly defined as double precision.

The standard list of ESATAN reference forms is given in Appendix C. Of these, all are allowed in the operations blocks except for local indices (KL*n*). Character local constants are allowed in order to simplify the input of concatenated model names.

The following entities are associated with thermal node definitions:-

L	-	label
T	-	temperature
C	-	capacitance
QA	-	heat source (albedo)
QE	-	heat source (earth)
QI	-	heat source (internal)

QS	-	heat source (solar)
QR	-	heat source (rest)
A	-	area
ALP	-	solar absorptivity
EPS	-	infrared emissivity
QAI	-	incident heat source (albedo)
QEI	-	incident heat source (earth)
QSI	-	incident heat source (solar)
FX FY FZ	-	cartesian coordinates

with thermal conductors:-

GL	-	conductor (linear)
GR	-	conductor (radiative)
GF	-	conductor (fluidic)
GV	-	conductor (view factor)

with fluid nodes:-

A	-	heat transfer area
FD	-	hydraulic diameter
FL	-	length
P	-	static pressure
FE	-	specific enthalpy
T	-	temperature
FF	-	wall surface roughness
FQ	-	internal heat source
FM	-	mass source rate
FH	-	specific enthalpy of mass source
FR	-	temperature of mass source
FX FY FZ	-	cartesian coordinates
FT	-	fluid type
VQ	-	vapour quality
FRG	-	predicted flow regime
FLA	-	flow area
VOL	-	nodal volume
PHI	-	relative humidity (AIRW only)
FW	-	vapour quality of mass source; specific humidity of mass source (AIRW only)
CMP	-	compliance – rate of change of unit volume w.r.t. pressure
VDT	-	rate of change of volume with respect to time
FST	-	fluid state descriptor

and with fluid conductors:-

M	-	mass flow link
GP	-	fluid conductance

References to the above entities comprise simply the entity followed by its reference number, e.g.

The temperature of node 56 = T56
The internal heat source of node 587 = QI587
The linear conductance between nodes 10 and 12 = GL(10, 12)

and similarly for FHTS entities:

The length of fluid node 17 = FL17
The internal heat source of fluid node 32 = FQ32
The mass flow link between fluid nodes 32 and 33 = M(32 , 33)

Referencing entities in included submodels is done by using the concatenated submodel name. E.g.

The capacitance of node 19 in submodel E:S:A = C:E:S:A:19

User constants and arrays are referenced by the name given by the user, and thus have no standard reference form. E.g.:- The user constant USERCN in submodel ENGINE:PISTON:CONROD is referenced as:-

```
ENGINE:PISTON:CONROD:USERCN
```

Array RXZF in submodel TELESCOPE:PLATFORM is referenced as:-

```
TELESCOPE:PLATFORM:RXZF
```

Fluid properties (such as density or viscosity) may be referenced through a set of library functions provided for this purpose; these are described in Section 6.4.

The $VARIABLES1 block of any included submodel may be called from any operations block, for example:

```
$EXECUTION
#
     CALL ENGINE:VARIABLES1
     CALL CSGDMP(CURRENT)
```

Elements of all arrays except table arrays may be referenced individually in a Fortran-like manner. Element references may be literal values or variables, or expressions involving either or both of these. If array elements are redefined then the preprocessor generates a call to a subroutine with the defined value becoming one of the arguments. As mentioned all the library is in double precision which means the assigned value must be in double precision format. The following are valid examples:-

```
QINPUT(10) = 9.9D0
```

```
ARR1(11) = 0.1D0 * USRCON
T10 = TVALS(3, 2)
XYZ = 10.2 * ABC(I * J, 10)
```

In Mortran the user may call subroutines or functions of three types:-

a. Those present in the ESATAN library. See Chapter 6.

b. Those defined in the $SUBROUTINES block of the current, or another, submodel; so, for example,

```
CALL ENGINE:PISTON:SUB1(...)
```

would be a valid call to subroutine SUB1 defined in sub-model ENGINE:PISTON.

c. Those written by the user included in the link prior to solution. In this case, the names of the subroutines should be listed, one per line, in the $USER_LIBRARY section of the global data file (see Section 3.8.1). Functions only need to be listed here if they are called from a data block. A function listed in $USER_LIBRARY cannot be explicitly declared in an operations block, but this is unnecessary if the function name follows the implicit typing rule (A-H, O-Z are double precision, I-P are integer). It is recommended that external user routines start with the letter U, to avoid conflict with ESATAN library routines.

An attempt to call a subroutine or function which is not recognised as one of these types is treated as an error.

The Mortran language is a superset of Fortran 77, in that any statements may be used which are allowable on the particular machine, plus several features which would not normally be possible under a Fortran compiler. These are:-

a. **SELECT...CASE**

SELECT CASE (*exp*)
CASE *lower* [:*upper*] [,*lower* [:*upper*]] ... *statement*
.
.
.
[CASE ELSE]
[*statement*]
.
.
.
END SELECT

Any number of cases may be given in one construct. The expression (*exp*) is evaluated at solution time, and the first CASE encountered which satisfies the evaluation of the expression is executed. Cases may be identified by point values and/or ranges, in which case the lower and upper limit of each range are separated by a colon as shown. Each case may have several ranges or point values, the only

restriction being the maximum allowable statement length. Both expressions and values may contain ESATAN entities. Ranges are inclusive of both end points.

The user may specify a default case under the CASE ELSE statement, giving instructions to be executed if none of the cases are satisfied. This is optional however, and the preprocessor will assume that the default case is to take no action.

b. **WHILE...ENDWHILE**

WHILE (*l-exp*)
statement
.
.
.
ENDWHILE

This is an iterative loop, being executed while the logical expression (*l-exp*) is true. Note that the condition is tested before execution of the loop. A limit of 99000 loops is permitted.

c. **REPEAT...UNTIL**

REPEAT
statement
.
.
.
UNTIL (*l-exp*)

Another iterative loop, being executed until the logical expression (*l-exp*) is true. Note that the statement block in this case is always executed at least once. A limit of 99000 loops is permitted.

d. **Subscripted values**.

The syntax is shown in Appendix C. The integer expression enclosed in brackets following an entity reference is evaluated by the preprocessor and added to the reference to the database location used by the Fortran program.

For example, if the user had specified in his $NODES block the existence of nodes

```
1, 2, 3, 5, 10, 15, 16, 20, 22, 25, 30, 100, 150
```

then the reference

```
T10(5)
```

would indicate the temperature of node 25, which is the fifth node after node 10 in the numeric sequence.

Conductors may also be referenced in this manner. The preprocessor makes no attempt to ensure that such references are valid, therefore the user should exercise extreme caution when using this feature.

e. **Entity referencing**.

Entities of other submodels are referenced by the use of concatenated submodel names, as in the data blocks. For nodes and conductors, the format is:-

entity [:*cname*:] *n*

and for constants and arrays:-

[*cname*:] *entity*.

f. **Comments**.

A hash (#) may be used anywhere on a line to indicate that the rest of the line is a comment. A hash is however not permitted to appear within a statement, i.e. a continuation line may not follow a full-line comment.

The above features may all be used freely within the operations blocks, the only exception being within subroutines or functions defined in the $SUBROUTINES block which are declared to be Fortran. This point is explained in more detail in Section 3.6.2.

Mortran structured-programming statements (i.e. (a), (b) and (c) above) have a maximum nesting level of 99. Such statements may be freely mixed, i.e. statements within may contain further structured programming statements.

The input format of Mortran is similar to that of Fortran, i.e. columns 1-5 on each line are reserved for statement and format labels, and column 6 is reserved for a statement continuation character. Columns 7-255 are available for statements. A maximum of 19 continuation lines are accepted by the preprocessor, and the same number are allowed for the resulting generated Fortran statement.

Reserved variables contained within the database are available to the user through various library functions (see Chapter 6).

3.6.2 $SUBROUTINES

The $SUBROUTINES block is used to provide user-defined subroutines and functions associated with the model, for example to calculate temperature dependent physical properties. These may only be called from another operations block or subroutine. There is no limit to the number of subroutines contained within the $SUBROUTINES block.

The syntax of the subroutine definition line is:-

SUBROUTINE *name* [(*arg*[,*arg* ...])] [LANG = **MORTRAN** | FORTRAN]

where *name* has a maximum length of 18 characters.

This syntax is similar to the Fortran syntax apart from the optional language specification. Subroutines written in Fortran will be passed directly to the solution program by the preprocessor, and are therefore not allowed to use any of the Mortran facilities described in Section 3.6.1. The default language is Mortran.

In the interpretation of subroutines, the preprocessor renames each one in the generated Fortran program. This is because implicit submodel definitions could lead to duplication of subroutines and hence to ambiguities. In Mortran subroutines, references to other subroutines are correctly interpreted, but since no such checking is done in Fortran subroutines the references are not interpreted. Consequently, it is not possible to call other subroutines or functions, which are defined in the input file, from within a LANG=FORTRAN routine. The user should be aware of this point; it is advised that, if such usage is required, the external user library feature (see Section 3.8.1)should be considered.

Additionally, it should be noted that if one function or subroutine calls another function or subroutine, both being defined in the $SUBROUTINES block, then the definition of the calling routine must follow the definition of the called routine. That is, **it is not possible to reference a routine before defining it**.

Users should ensure that subroutines are correctly terminated, i.e. with an END statement, and also that at least one RETURN statement is contained within the subroutine. The preprocessor will attempt to correct omission of both or either of these statements, but it is advisable that the user includes them.

Users may also include a private library of subroutines, developed separately and stored in compiled form in a file format which is suitable for the system linker or loader. The names of these subroutines and functions should be made known to the preprocessor and it is therefore required that these names are stored in the global file after the $USER_LIBRARY card (see Section 3.8.1).

3.6.3 $INITIAL

The function of this block is to provide an area where the user can perform initialisation of data prior to the program solution. The section of the Fortran program resulting from the initial block is executed once only at the start of the solution.

A simple example of an $INITIAL block is shown below:-

```
$INITIAL
#
     INTEGER I
     I=0
     REPEAT
          T1(I) = 20.0D0
          I=I+1
     UNTIL (I .EQ. 20)        #SET INITIAL TEMPS.
     CALL TMPSAM
```

This example shows the use of the REPEAT ... UNTIL loop to set a band of nodes to an initial temperature, and also the subscript feature to increment through the nodes. Finally a call is made to a user-defined subroutine.

Any submodel at any level may have a $INITIAL block. During processing, the $INITIAL block for each submodel becomes an individual subroutine within the Fortran program. When formulating the Fortran for the main (top level) model, calls are made from the $INITIAL subroutine to all of the lower level $INITIAL subroutines, in the order in which they were encountered in the input file. Thus the $INITIAL block contents for the top level model are the last to be executed.

During processing of the data blocks, the preprocessor may generate Mortran as a result of node, conductor, or function definitions, supernodes etc. It may be appropriate that the generated Mortran is placed in the $INITIAL block instead of the $VARIABLES1 block. In this case the GENMOR instruction can be used. This statement may be inserted as the single six-character word "GENMOR" on a line with no other statements, and will result in generated Mortran pertaining to the operations block being inserted at that point in the block. E.g.:-

$INITIAL
statement
.
.
.
GENMOR
statement
.
.
.

The user should be aware that this feature will, in effect, allow the cancellation or modification of details given in the data blocks.

Mortran is generated in the $INITIAL block of the main model and each submodel for the initialisation of the state at fluid nodes (FHTS). The preprocessor creates a call to the subroutine FINITS (Section 6.4.13) which ensures that the state variables—pressure (P), temperature (T), enthalpy (FE) and vapour quality (VQ)—are all consistent at each fluid node.

3.6.4 $VARIABLES1

Instructions placed in the $VARIABLES1 block are executed at the start of each iteration or at the start of each time step in the solution. Such instructions should, therefore, be concerned with temperature- and time-dependent quantities. Examples are temperature-dependent material properties and time-varying boundary conditions.

The user is free to express such variations himself within $VARIABLES1. However, if he defines such variations for a particular entity in the data blocks (e.g. by setting node or

conductor data to functions and/or expressions) then the preprocessor generates appropriate Mortran statements and places them in $VARIABLES1. Such generated statements are normally placed in the block following any user-defined Mortran, thus overwriting any of the latter which refer to the same entity. However, a method is provided to avoid this, by use of the GENMOR statement. The usage of this statement is as described for $INITIAL (Section 3.6.3).

If the model has an associated Analysis Case Definition (ACD) file, the dynamic data is automatically read for the current time, and the data assigned to the appropriate parameters. The data is read at the start of $VARIABLES1 of the main model, prior to executing any generated user-defined logic or generated MORTRAN. See Section 3.10 for more information on the ACD file and the processing of data.

A typical $VARIABLES1 block would take the form:-

```
#
$VARIABLES1
#
     CALL TCAPCA(C10,T10)       #CALCULATE TEMPERATURE-
     CALL TCAPCB(C56,T56)       #DEPENDENT CAPACITANCES
     IF(T10 .LE. 10.0D0)THEN
          QI10 = 5.6D0
     ELSE
          QI10 = 0.0D0
     END IF
```

In this block, two calls are initially made to user-supplied routines for the calculation of temperature-dependent capacitances. Next a check is made in an IF ... END IF statement, adjusting the internal heat source of node 10 according to its current temperature.

A subroutine is created by the preprocessor corresponding to the $VARIABLES1 block. This subroutine can be executed from any operations block by the statement

```
CALL VARIABLES1
```

(the $-symbol is omitted). Any submodel may contain a $VARIABLES1 block, and the method by which the overall model's VARIABLES1 subroutine is constructed is identical to that used for the $INITIAL block.

The control constant SOLTYP (Appendix E) is related to the $VARIABLES1 block and is provided for use with the FHTS thermohydraulic solvers (Sections 6.14.11–6.14.16). It allows code to be executed only at the appropriate point in the solution, for instance during the thermal or the hydraulic iterations. An example is given below:

```
$VARIABLES1
#
IF (SOLTYP .EQ. 'THERMAL') THEN
     CALL TCAPCA(C10,T10)       #Calculate temperature-
     CALL TCAPCB(C56,T56)       #dependent capacitances
     IF(T10 .LE. 10.0D0)THEN
          QI10 = 5.6D0
```

```
            ELSE
                 QI10 = 0.0D0
            END IF
        END IF
#
        IF (SOLTYP .EQ. 'FLUID') THEN
#           Update fluid conductance
            GP(61,62) = GPCALC(M(61,62) , T61 , T62)
        END IF
#
```

NB: The $keywords $THERMAL, $FLUID and $BOTH, which test the value of SOLTYP in a similar manner, are a deprecated feature, and should be considered obsolete.

3.6.5 $VARIABLES2

The $VARIABLES2 instructions are executed at the end of each time step in the solution. Since the current solution time is known, $VARIABLES2 is the appropriate block for instructions concerning metering and integrating quantities, to rearrange the model as a result of some time-dependent event, or for comparison of latest results with test data. A simple $VARIABLES2 block could take the form:-

```
$VARIABLES2
      CALL ROTATE          #SET LATEST POSITION
      HX=FLUXML(CURRENT,LOUVRE:PIPE)
      HY=FLUXL(D10,D56,D:HEATPIPE:CAPILLARY:999,
     &                 D:HEATPIPE2:CAPILLARY:1023)
      HZ=HX+HY
#
```

This example shows a user-defined subroutine, ROTATE, being called, to set a time-dependent position for the model. The library functions FLUXML and FLUXL are used, the former to obtain linear heat flow between the current submodel and submodel PIPE, the latter to obtain linear heat flow between certain nodes in the current submodel and submodel CAPILLARY. The final statement sums the two heat flows.

The $VARIABLES2 block may be included in any submodel and the method by which the overall model's VARIABLES2 subroutine is constructed is identical to that for the $INITIAL and $VARIABLES1 blocks. Again, the GENMOR card may be used to position any generated Mortran within the block, as described in the $INITIAL block section.

3.6.6 $EXECUTION

The $EXECUTION block is called once only after the $INITIAL block is executed and usually requests the library solution routines necessary to solve the problem. These will be listed as a series of subroutine calls, and a typical $EXECUTION block may appear as follows:-

```
#
$EXECUTION DYSTOR=2500000
```

```
#
     CALL SOLVIT                  # Solve network
     CALL SAVET(CURRENT)          # Store steady state
     CALL PRQNOD(' ', CURRENT)    # Print heat flows
#
```

In this example, SOLVIT is called to perform a steady state solution, SAVET is called to write out the steady-state temperatures to an unformatted file, and finally heat flows along every conductor are printed out via PRQNOD.

The $EXECUTION block is significant only in the top-level model: the preprocessor will ignore any such blocks encountered at lower levels.

The line introducing this block may contain a parameter, the full definition being:-

$EXECUTION [, DYSTOR = *n*]

DYSTOR sets the number of locations of dynamic storage, or program work space, to be allocated for the solution. The amount needed will depend on the solvers and output routines being called (see Sections 6.10 and 6.15). A minimum of 1,000,000 locations is always reserved by default.

3.6.7 $OUTPUTS

The contents of $OUTPUTS are performed automatically by the solution routines. The block may contain any Mortran the user desires, but is primarily concerned with instructions for the output of information. Only the main-model $OUTPUTS block is executed.

For example:-

```
#
$OUTPUTS
#
     CALL PRNDTB('LOUVRE','T',CURRENT)
#
```

This will result in a printout, in table format, of the temperatures of all nodes in the model containing the text "LOUVRE" in their label. For a full description of all the output routines, see Section 6.10.

Steady-state solution routines execute $OUTPUTS at the end of solution, when the steady state has been found, whereas transient solvers call $OUTPUTS at the following times: a) at the start of solution, i.e. before the first timestep; b) at each output interval (see below); and c) at the end of solution, i.e. after the last timestep. Note, however, that $VARIABLES1 is called before $OUTPUTS; hence, entities defined using Mortran may not appear in the first output with their initialisation values.

Two control constants are associated with this block. OUTINT specifies the interval between outputs during a transient solution (it is ignored by a steady-state solver); for

instance, if OUTINT = 10.0 then $OUTPUTS will be executed every 10 s of simulation time.

OUTIME is a character control constant which is used to suppress $OUTPUTS, typically during certain phases of a transient run. If OUTIME = 'ALL' (the default), the block is executed normally, but if OUTIME = 'NONE' then it is bypassed irrespective of the output interval defined by OUTINT.

3.7 Parameter Cases

ESATAN has a built-in facility for allowing various forms of parametric analysis, for example determining the sensitivity of predicted temperatures to the material and optical properties used in a model. The solution is performed repeatedly with certain parameters in the model being varied for each run, as prescribed by the user. Postprocessing is facilitated by recording the results either in comma-separated value format or as a TMD file, the latter format being suitable for loading into ESATAN-TMS ThermNV; ESATAN automatically directs the various kinds of output to systematically named files.

Parameter cases are defined in a separate file which is loaded at solution time, thus avoiding the need to re-preprocess the model. The default name for this file is *model*.PAR, where *model* is the name of the model being run: if a file so named is present in the working directory, a parametric solution will be initiated automatically. However, the ESATAN front-end user interface allows *any* file to be specified for a parametric solution.

Usually the model as defined in the input file—the so-called *nominal* case—will be run as normal prior to any parameter cases being executed, but this can be omitted if required.

3.7.1 $PARAMETERS Keyword

The parameter case file starts with the following keyword line:

$PARAMETERS [, PARMONLY] [, OUTPUT = **SINGLE**|MULTIPLE]
[, CSV_ENTITIES = (*entity_refs*)] [, CSV_OUTPUT = **SINGLE**|MULTIPLE]
[, TMD_ENTITIES = *string*] [, GFF_ENTITIES = *string*]
[, OUTPUT_FROM = **OUTPUTS**|VARIABLES2] [, CHECK]

(Note that, although for purposes of documentation it is shown here extending across several lines, this must in reality be on a **single line** of no more than **255 characters**.)

The options available for the $PARAMETERS statement, governing all subsequently defined parameter cases, are:

PARMONLY	-	Indicates that the nominal case is not to be run, only the parameter cases.
OUTPUT	-	Determines whether a single, aggregate standard output file is produced (= SINGLE), or a separate one for each parameter case (= MULTIPLE).
CSV_ENTITIES	-	A comma-separated list, enclosed in brackets, of entity references whose values are to be output to a CSV file(s), e.g. T5, GL(*), QI:'TOP'. The syntax of these entity references is described in Section 3.7.7.
CSV_OUTPUT	-	Determines whether a single, aggregate CSV file is produced (= SINGLE), or a separate one for each parameter case (= MULTIPLE).
TMD_ENTITIES	-	A string specifying the values for which kinds of entity are to be output to a TMD file; a comma-separated combination of NODES and CONDUCTORS. These may be qualified by listing the required entities inside brackets, as described for DMPTMD (Section 6.2.2).
GFF_ENTITIES	-	Deprecated. Use TMD_ENTITIES.
OUTPUT_FROM	-	Specifies whether CSV and TMD output is produced from the $OUTPUTS block (i.e. at intervals determined by control constant OUTINT), or from $VARIABLES2 (i.e. after every time step); = OUTPUTS or VARIABLES2, respectively. Only relevant for transient solution.
CHECK	-	Indicates a validation run only; parameter cases are parsed but the solution is not run.

3.7.2 Parameter Case Definition

Following the $PARAMETERS marker, each parameter case definition takes the form

!INITIAL | !FINAL [= *string*]
command
.
.
.

There may be any number of parameter cases, and any number of commands in each one.

Two types of parameter case are supported, *initial* and *final*. An *initial* parameter case first restores the state of the model to that defined by the data blocks, and then executes the contents of the $INITIAL block. Next the parameter case commands are enacted, after which the model's $EXECUTION block is obeyed.

Final parameter cases execute the commands and the $EXECUTION block only; i.e. the state of the model at the start of a final parameter case is its current state when the previous solution is completed.

Each parameter case may be assigned an optional literal character string, subsequently referenced by the character control constant PARNAM. The maximum length of this string is 24 characters.

There are four parameter case commands available, described in the following sections:

OFF *cond_ref*
ON *cond_ref*
CHANGE *entity_ref* = *r-val*
SCALE *entity_ref* * *r-val*

3.7.3 The CHANGE Command

CHANGE *entity_ref* = *val* [, *val*, ...]

This assigns the given value to the entities specified in *entity_ref* (see Section 3.7.7). Where *entity_ref* is an array reference, a comma-separated list of values can be given to be assigned to successive array elements starting at the element specified.

For example:

```
CHANGE QS:PANEL:*-ONLY = 5.67
```

changes the value of all solar heat sources in submodel PANEL to 5.67;

```
CHANGE RACK1:HLOAD = 50.0
```

sets the user constant HLOAD in submodel RACK1 to 50.0;

```
CHANGE EclipseTimes(5)= 410.0, 520.0, 630.0
```

sets the 5th, 6th and 7th elements of the array EclipseTimes to 410.0, 520.0 and 630.0, respectively;

```
CHANGE D10=X
```

changes status of node 10 from diffusion to inactive.

3.7.4 The SCALE Command

SCALE *entity_ref* * *r-val*

This multiplies the (real-valued) entities specified in *entity_ref* (see Section 3.7.7) by the given factor.

For example:

```
SCALE GL(*) * 0.95
```

multiplies all linear conductances in the model by 0.95;

```
SCALE C:RACK2:'#5-9' * 1.1
```

scales the capacitances of nodes 5 to 9 in submodel RACK2 by 1.1;

```
SCALE SPECHT(2, *) * 0.9
```

multiplies all elements in the second dimension of array SPECHT by 0.9.

3.7.5 The OFF Command

OFF *cond_ref*

This command turns off the conductors named in *cond_ref* (see Section 3.7.7), i.e. makes them inactive for solution.

For example:

```
OFF GL(10,15)
```

makes linear conductor GL(10, 15) inactive;

```
OFF GR:AXU:(*)
```

turns off all radiative conductors in submodel AXU and its children.

3.7.6 The ON Command

ON *cond_ref*

This turns on the conductors named in *cond_ref* (see Section 3.7.7), i.e. makes them active for solution.

For example:

```
ON GL('TOP', 'BOTTOM')
```

makes active all linear conductors between those nodes with 'TOP' in their label and those whose label contains 'BOTTOM'.

3.7.7 Entity References

CSV output is requested by specifying a list of entity references on the $PARAMETERS line. Also, all the parameter-case commands act on an entity reference specification. For CHANGE and SCALE (*entity_ref*) this can involve any of nodes, conductors, user and control constants, or arrays. For ON and OFF (*cond_ref*) this is restricted to conductors only.

For **node** entities, the reference can take one of the following forms:

nent[:*cname*:]*n* - e.g.
```
T4
QS:ARM:9
```

nent:[*cname*:]*uname* - where *uname* is a node alias; e.g.
```
T:thermocouple
QS:ARM:motor
```

nent:[*cname*:]*string* - where *string* is interpreted in the same way as ZLABEL in Section 6.1.3; e.g.
```
T:'TOP'
QA:'#10; PANEL:20-30'
```

nent:[*cname*:]* - to mean all nodes in the named submodel or its included submodels; e.g.
```
T:*
QR:ASSEMBLY:MIDARM:*
```

nent:[*cname*:]*-ONLY - to mean all nodes in the named submodel but not its included submodels; e.g.
```
T:*-ONLY
C:ENDCAP:*-ONLY
```

For **conductor** entities, the reference can take one of the following forms:

cent[:*cname*:](*node_ref*, *node_ref* [, *m*]) - to mean all conductors of the given type, defined in the named submodel, between the specified nodes; e.g.
```
GL(4, 5)
GL:ARM:(motor, 3)
GL(5, 'TOP')
GR(ASSEMBLY:MIDARM:1, 999, 2)
GR(ASSEMBLY:*-ONLY, space)
```

cent[:*cname*:](*) - to mean all conductors of the given type, defined in the named submodel or its included submodels; e.g.

```
GL(*)
GR:ASSEMBLY:(*)
```

cent[:*cname*:](*-ONLY) - to mean all conductors of the given type, defined in the named submodel but not its included submodels; e.g.

```
GL(*-ONLY)
GR:ASSEMBLY:(*-ONLY)
```

Here, *node_ref* is a node reference using the same syntax as described above for nodal entities and *seq_num* is the sequence number in the case of parallel conductors (if a sequence number is not given, then *all* parallel conductors between the specified nodes are addressed).

There are some subtleties to be observed in this. For instance, GL(*) means *all* linear conductors in the entire model (i.e. main model and submodels), whereas GL(*, *) refers to only those conductors that are themselves defined in the main model, even if the nodes they couple are actually in a submodel (hence this is equivalent to GL(*-ONLY)).

User and **control constant** references take the following simple form:

[*cname*:]*uname* - where *uname* is the constant name; e.g.

```
RELXCA
ASSEMBLY:MIDARM:Angle
```

Array references take the form of either a single array element or a range consisting of all elements in one dimension:

[*cname*:]*uname*(*subs*, ...) - where *uname* is the array name; e.g.

```
ARR(5)
ASSEMBLY:MIDARM:JCoef(2, 4)
```

[*cname*:]*uname*(*subs*, ..., *, *subs*, ...) - where *uname* is the array name; e.g.

```
ARR(*)
ASSEMBLY:MIDARM:JCoef(2, *)
```

3.7.8 Parameter Case Output Files

By default, the standard output from running a series of parameter cases on a model goes, as normal, to a file called *model*.out. A header is printed at the start of each parameter case giving summary information.

If the OUTPUT=MULTIPLE option is selected on the $PARAMETERS line, each parameter case will produce a separate standard output file named

*model*_PAR*nnnn_PARNAM*.out, where *PARNAM* is the parameter case name if one has been specified. The nominal case will be reported in *model*_PAR0000_ORIGINAL.out.

As stated previously, CSV output is requested by giving a CSV_ENTITIES specification on the $PARAMETERS line. If also CSV_OUTPUT=SINGLE then all the CSV results are written to a file called *model*_PAR.csv; alternatively, if CSV_OUTPUT=MULTIPLE then, similarly to the standard output, the results from each parameter case go to a separate file named *model*_PAR*nnnn_PARNAM*.csv.

TMD output—requested with a TMD_ENTITIES specification—is always written to a separate file for each parameter case. Following the convention above, these files are named *model*_PAR*nnnn_PARNAM*.TMD. Multiple solution calls can result in more than one TMD file being produced for each parameter case, in which event a numerical suffix is added to the subsequent files: *model*_PAR*nnnn_PARNAM*.TMD*n*.

Where the solution performed on the model is a transient, both CSV and TMD output is generated at intervals throughout the solution, either every time step (OUTPUT_FROM=VARIABLES2) to give a complete history, or at every standard output interval (OUTPUT_FROM=OUTPUTS) if regular ‘snapshots’ are sufficient.

3.7.9 Example Parameter Case Files

Suppose we have a model on which we have run a steady state solution. However, there is an uncertainty of ±10% in the value of conductivity used in calculating linear conductances and of ±5% for the emissivity used in calculating radiative conductances. To determine the sensitivity of the predicted temperatures to these uncertainties we use the following parameter cases to produce a CSV file, which we can then analyse using a spreadsheet program:

```
#
# Sensitivity analysis
#
$PARAMETERS, CSV_ENTITIES=(T:*)
#
!INITIAL = 'cond-1'
SCALE GL(*) * 0.9
#
!INITIAL = 'cond-2'
SCALE GL(*) * 1.1
#
!INITIAL = 'emiss-1'
SCALE GR(*) * 0.95
#
!INITIAL = 'emiss-2'
SCALE GR(*) * 1.05
#
```

The next parameter case file represents a design analysis to assess the effect of the internal heat dissipation of a particular component. We are interested in the impact on the

temperatures at certain nodes and also on the heat flow between two submodels, the latter being calculated in the model and stored in a user constant called Qtot. The solution performed is now a transient and so separate CSV files are requested. In addition, the more detailed TMD output is requested for analysis with the dedicated postprocessing tool ThermNV:

```
#
# Design analysis
#
$PARAMETERS, PARMONLY, CSV_ENTITIES=(T:'#5, 25, 30', Qtot),
     CSV_OUTPUT=MULTIPLE, TMD_ENTITIES='NODES, CONDUCTORS'
#
!INITIAL = 'EBOX-1'
CHANGE QI:'#EBOX:1-10' = 5.0
#
!INITIAL = 'EBOX-2'
CHANGE QI:'#EBOX:1-10' = 10.0
#
!INITIAL = 'EBOX-3'
CHANGE QI:'#EBOX:1-10' = 15.0
#
!INITIAL = 'EBOX-4'
CHANGE QI:'#EBOX:1-10' = 20.0
#
!INITIAL = 'EBOX-5'
CHANGE QI:'#EBOX:1-10' = 25.0
#
```

Finally, we illustrate how parameter cases can be used to control the solution and change boundary conditions at the same time. Suppose that our model has an integer user constant, MODE, and that the $EXECUTION block looks like this:

```
$EXECUTION
      SELECT CASE (MODE)
      CASE 0 # Steady state
         CALL SOLVFM
      CASE 1 # Transient
         CALL SLFWBK
      END SELECT
```

Then we can run a steady state solution followed by a transient and then another steady state, with a varying load, like so:

```
$PARAMETERS, PARMONLY
!INITIAL = 'Initial steady state'
CHANGE MODE = 0
CHANGE QI10 = 100.0 # Nominal heat load
#
!FINAL = 'Evolution with 50% increase'
CHANGE MODE = 1
SCALE QI10 * 1.5
#
```

```
!FINAL = 'Final steady state'
CHANGE MODE = 0
#
```

3.7.10 Remarks

It is important to understand that alterations to the model specified by parameter cases take effect *before* the solution is run. Hence any quantities calculated dynamically using Mortan—for instance, node capacitance defined by interpolation on temperature in the $NODES block, or conductances evaluated in $VARIABLES1—will not be directly amenable to parametric analysis using this facility. In such situations a user constant can be defined and applied as a multiplying factor to the quantity in question (with unity as the default value for the nominal case), and then this user constant can be varied using parameter cases.

The ESATAN Parametrics Manager is a simple graphical utility for generating multiple parameter cases where only single CHANGE or SCALE commands are needed, e.g. for sensitivity analysis. See Appendix O for more information.

Use of a $PARAMETERS block in a model input file, although still supported for backwards compatibility, is now deprecated, as is the ability to abbreviate commands. Also, the GET command is now considered redundant but is again retained for backwards compatibility. GFF_ENTITIES is obsolescent: TMD_ENTITIES should be used instead. For backward-compatibility reasons, a blank string may be used for TMD_ENTITIES as equivalent to 'NODES, CONDUCTORS'. It is likely that these features will be withdrawn in a future version of ESATAN.

3.8 Global Data File

The global data file contains data which is global to an ESATAN model, rather than specific to a submodel. At present this is the following:-

a. List of external subroutines and functions referenced within the ESATAN model (USRLIB)

b. List of user defined nodal entities (USRNOD)

c. User defined $ELEMENT models (ELEMUSER)

The format of the global data file is as follows:-

```
$USER_LIBRARY
...
$USER_NODE_ENTITIES
...
$USER_ELEMENTS
...
```

Any of these headers can be omitted if there is no associated data. However, those sections which do appear must be in the order given above.

3.8.1 User Library

The ESATAN Preprocessor checks all subroutine calls in the operations blocks and gives an error message if it does not recognise a name used. In order to allow users to link in external libraries of routines it is necessary to tell the Preprocessor the names of these external routines. This is done by specifying the routines names, one per line, in the $USER_LIBRARY section of the global data file. Note that only subroutines which are directly called from within the ESATAN Source File need be included - functions should not be included. The format is as follows:-

```
$USER_LIBRARY
SUB1
SUB2
```

3.8.2 User-Defined Nodal Entities

In order to give greater flexibility in accessing particular data within the operations blocks, the user may define his own nodal entities. These may be of type real, integer or character, with character entities having a value of up to 24 characters long.

User-defined nodal entities have to be declared in the $USER_NODE_ENTITIES section of the global file as illustrated below, following a similar syntax to the $CONSTANTS block but without assigning a value:-

```
$USER_NODE_ENTITIES
#
$REAL
Heat_exchange;
#
$INTEGER
Integer_entity_a;
Integer_entity_b;
Int_maximum_length; # max length of 18 characters
#
REAL* A_Real_one;
CHARACTER* CHAR_enta;
Int_entity_c;
#
$REAL
Another_Real;
#
$CHARACTER
CHAR_entb;
```

The name of each entity is supplied by the user, is case-sensitive and may be up to 18 characters in length. A name must not end in a digit. The file is restricted to one entity per record, and descriptive comments are allowed.

Although not available in the data blocks, user-defined nodal entities are used within the operations blocks in the same manner as true nodal entities. For example,

```
Heat_exchange456
```

refers to the value of the entity `Heat_exchange` at node 456. Similarly, they may be used in the subscripted form

```
Int_entity_b1(I)
```

where `I` is a Mortran loop counter.

Of course, it is for the user to define the value of these entities. For output purposes, the entity names may be included in the following ESATAN library routines: PRNDTB, PRNDBL, PRTNAV and PRTNSM (see Section 6.10).

User-defined nodal entity data may also be output using the data-dumping routine DMPGFS, which produces an output file that may be selectively viewed using the Data Extraction Utility (see Appendix M).

3.8.3 User Elements

Users may describe thermal or fluid models (user elements) of their own to be included within the ESATAN model using the $ELEMENT keyword and these user elements should be contained in the global data file. Element definitions should follow the $USER_ELEMENTS header. There is no limit to the number of elements that can be defined within the $USER_ELEMENTS block.

When an element is referenced in a user's source file, the $SUBSTITUTIONS block creates substitutions of literal text strings within the element submodel. Within the element submodel values to be substituted are parenthesised with the percent (%) character. Note that the substitution is a pure text substitution of the complete element definition and all occurrences of % will be treated as a substitution parameter. This means that the use of a % within a comment should be avoided (the use of a single % character will give rise to an error).

For example, the element definition in the global data file:-

```
$USER_ELEMENTS

  $MODEL PLATE
  #
  $NODES
  FOR KL1 = 1 TO %NNUM% DO
       DKL1, T = 22.6, C = %CAPP%;
  END DO
  $CONDUCTORS
  :
  :
  :
  :
```

```
$ENDMODEL PLATE
```

and the user's model in the input file:-

```
$MODEL BASE
#
# Uses the standard plate model
# contained in the ELEMUSER.DAT file
#
$ELEMENT PLATE
#
$SUBSTITUTIONS
#
NNUM = 10 ;  # Total number of nodes
CAPP = 5.6 ; # Capacitance of nodes
#
$ENDMODEL BASE
```

will result in the substitution of 10 for the total number of thermal nodes and 5.6 for the capacitance of each node. Note that each substitution definition must be terminated by a semi-colon.

It is assumed that the element is a complete model definition. That is, no further block can be added after the $SUBSTITUTIONS block. If there is a duplication of an element name between the global data file and the ELEMSYS.DAT files, the element definition in the global data file has priority.

To increase the flexibility of the $ELEMENT facility, and reduce user input, default values for substitution data can be specified in the element definition in the global file. Defaults are defined immediately after the $MODEL card with the $DEFAULTS block. Rules of format are the same as for other ESATAN data blocks. Looking at the previous example again, but this time introducing a $DEFAULTS block, we have:

```
$MODEL PLATE
#
$DEFAULTS
#
CAPP = 0.0 ;   # Arithmetic nodes assumed
#
$NODES
FOR KL1 = 1 TO %NNUM% DO
     DKL1, T = 22.6, C = %CAPP%;
END DO
#
$CONDUCTORS
:
:
:
$ENDMODEL PLATE
```

If no substitution for CAPP is given within the $SUBSTITUTIONS block of the input file then the value of 0.0 is assumed in this case. No error or warning message is generated by

the preprocessor so care needs to be taken to ensure substitution values have not been omitted by mistake. All substitution data appearing within the element and not in the $DEFAULTS blocks must be defined, otherwise an error is reported.

3.9 VCHP Submodels

A VCHP submodel, indicated by the VCHP parameter on the $MODEL card, is used to describe a variable conductance heatpipe. This is a heatpipe in which a non-condensing gas is used to dynamically modify the effective length of the pipe as the temperature varies. Two library routines are provided within ESATAN for use with VCHP's:- the design routine, VCHPD, which determines the amount of non-condensing gas required to maintain a particular temperature under given conditions; and the routine VCHP, which calculates the vapour/gas-front position during a thermal solution with the VCHP active. These routines are described in detail in Section 6.5.6.

A VCHP as modelled in ESATAN consists of four sections; an evaporator, an optional adiabatic section, a condenser and a reservoir (for the control gas). With the exception of the reservoir which is modelled by a single ESATAN node, each of these sections may be modelled by as many nodes as the user requires. In addition to these nodes, a single vapour node must be specified. A reserved 4×2 array, VCHPAL, must be specified listing the cross-sectional areas of each of the four sections in the pipe followed by the lengths of each node within each section of the pipe. This implies, of course, that each node within a particular section is of the same length. The nodes must be defined with increasing (although not necessarily consecutive) numbering as one progresses from the evaporator to the reservoir. The vapour node must have a number higher than that of the reservoir node. Identification of pipe sections is through the node label. This must contain the string 'EVAPORATOR', 'ADIABATIC', 'CONDENSER', 'RESERVOIR' or 'VAPOUR' (alternatively 'VAPOR') to denote the appropriate section. If two or more of these strings occur within a node label then the first occurrence is taken. A linear conductor must be specified from the vapour node to each of the nodes in the evaporator, adiabatic section and condenser. These may be constant values or expressions and will be dynamically adjusted to reflect the current vapour front position.

In addition to array VCHPAL, the two-dimensional array, SVPRES, of saturated vapour pressure against temperature must be specified, and user constants MOLESG and VAPLEN defined. Choice of values for these constants is discussed in the descriptions of the library routines VCHPD and VCHP. Nodes and conductors other than those defining the VCHP may be specified within a VCHP submodel if this is considered advantageous. However, all such nodes should be numbered higher than the number of the vapour node.

3.10 Analysis Case Definition (ACD) File

ESATAN-TMS Workbench provides the option to directly generate a platform-independent binary representation of some of the ESATAN Thermal data rather than adding it to the text-based input deck. This results in substantial savings in both disk space and CPU time.

The data is contained in the Analysis Case Definition (ACD) file, which uses the open-source Hierarchical Data Format (HDF). The ACD file is automatically read at solution time; data for static quantities is loaded at the start of solution (see Section 2.4.4), and data for dynamic quantities is read as necessary and used to update such quantities throughout the solution, viz. at the start of $VARIABLES1 (see Section 3.6.4).

The presence of a populated ACD file is denoted by the parameter ACDFILE on the $MODEL line (see Section 3.3.1). There must always be an ACD file present for the solution; if a model does not have one already, the preprocessor will create a 'skeleton' ACD file containing no data. A model with its own ACD file may be included into another model as a submodel; the preprocessor will incorporate all submodel ACD data into the main-model ACD file. A model's ACD file must be named <MODEL>.acd, with the basename in upper case.

Quantities currently supported by the ACD file are albedo, planet, solar and remainder heat fluxes (QA, QE, QS and QR), radiative conductances (GR), and real and table arrays. For the heat fluxes, both static (e.g. orbit-averaged) and dynamic (time-dependent) data can be stored. For the radiative conductances, only dynamic data (time- or wavelength-dependent) may be stored. Dynamic quantities are updated (or not) during solution according to the type of analysis case and the type of solver (steady-state or transient), using appropriate interpolations, etc. Analysis case types supported at present are: static-only data, simple cyclic, chained arc, and wavelength-dependent; there is also the 'null' analysis case type used for the skeleton ACD file mentioned above.

The library routine ACDDYU may be called by the user in an operations block to effect an additional, discrete update of dynamic quantities from data in the ACD file; see Section 6.4.47.

4. FLUID LOOP MODELLING

4.1 General Concepts

The FHTS fluid node and fluid conductor have been devised in order to describe fluid models. Consider a section of pipe containing flowing fluid, as shown in Fig. 4-1.

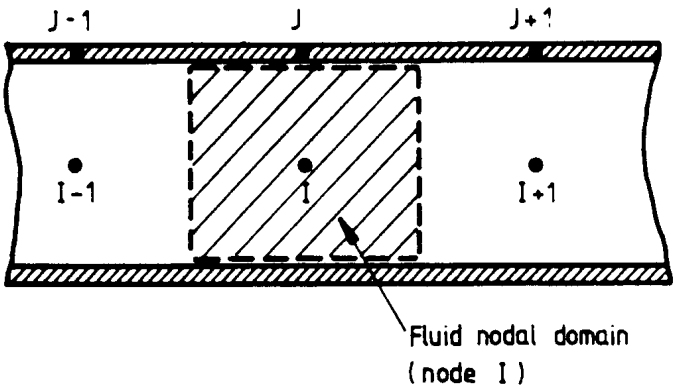


Figure 4-1. Pipe section showing nodal discretisation

Nodes representing the fluid are shown as I-1, I and I+1. The tube wall is represented by nodes adjacent to those in the fluid, and these are denoted J-1, J, J+1. The fluid nodal domain for node I is indicated, and its boundaries are the midpoints between it and the upstream and downstream fluid nodes, and the pipe walls. It is important to appreciate that the boundary between the fluid and the pipe wall is absolutely intimate, insomuch as that one of the attributes of the fluid node is the pipe wall surface roughness, as shown later.

The fluid cell as represented by the nodal domain for any given node is associated with the dependent variables of pressure and either enthalpy or temperature, i.e. the FHTS analyser solves for these quantities at each fluid node. (Whether enthalpy or temperature is solved for depends on the type of solution being performed: the single-phase solvers regard temperature as the primary variable, the two-phase solvers, enthalpy.) The third dependent variable, mass flow rate, is regarded as being associated with the link between the fluid nodes, as shown in Fig. 4-2.

During solution, the FHTS analyser assigns pressure and enthalpy/temperature to each node by solution of the mass and energy conservation equations. Mass flow rate is solved for each conductor through conservation of momentum. Mass-flow conductances defined between fluid nodes transport energy and momentum by advection, i.e. in the direction of flow only.

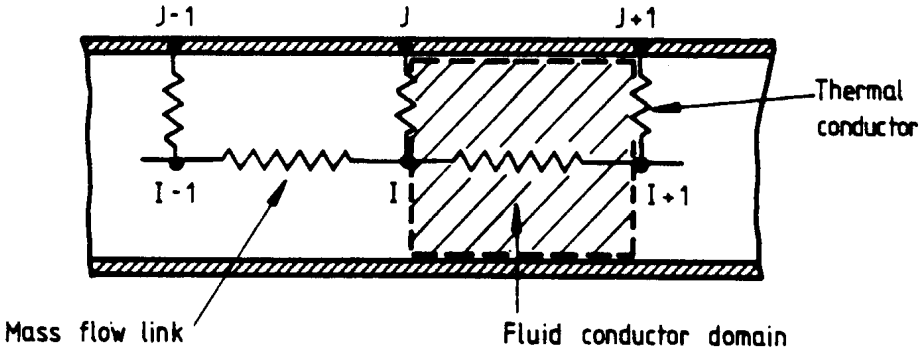


Figure 4-2. Pipe section showing conductor network

The fluid state at each node is initialised prior to solution from the entities defined by the user. Normally, pressure and temperature are specified in the $NODES block and from these enthalpy and vapour quality are calculated. If the user wishes, other combinations of state variable may be used, such as temperature and vapour quality; this is discussed in Section 3.5.2.

Values of temperature and pressure are relative to the control constants TABS and PABS, respectively. The defaults are TABS = 273.15 and PABS = 0.0; thus, unless the user redefines these control constants, temperatures will be in °C and pressures will be absolute values.

Gravitational effects, particularly on pressure, are modelled in FHTS. The components of the gravity vector are given by the control constants GRAVX, GRAVY, GRAVZ. By default, zero gravity is assumed: GRAVX = GRAVY = GRAVZ = 0.0

As with conventional ESATAN models, a coherent system of units must be employed. The library of standard fluid properties (Section 4.2.1) is given in the S.I. system (kg-m-s), but the user may employ any consistent system of units if user-defined fluid properties are given.

4.2 Fluid Property Description

A library of physical properties for a range of fluids is provided within FHTS, comprising air, water, ammonia, and certain freons. Alternatively the user may provide his or her own thermosphysical data.

The fluid required must be specified in the input file in one of two ways. The first method is through the $MODEL card, in which case the syntax is:-

```
$MODEL name, FLUID = fname
```

Alternatively, the fluid type may be specified within the node definition by the entity FT. In this case, FT must be a literal character string, in quotes. Such definitions override any fluid type definition on the $MODEL card.

Note that specification on the $MODEL card is mandatory in some of the fluid elements. There is no restriction upon the number of different fluid types defined in any one submodel.

4.2.1 Library of Fluid Properties

The fluid types recognised by FHTS are:-

WATER
AIR
AIRW
AMMONIA
R11
R12
R21
R22
R114
R502.

Fluid type AIRW is a two-component gas consisting of air and water vapour, combining the properties of both of these fluids. It is used for humidity simulation with the single-phase solution routines.

The units of the library fluid properties are S.I. (kilograms - metres - seconds), and the correlations are valid within the ranges given in Table 4-1. The pressures quoted are absolute. Also given are the enthalpy datum states.

It must be noted that for the calculation of subcooled and superheated properties of specific heat, conductivity and viscosity, saturation values are assumed (at the fluid temperature). This is true for all fluid types except R11, R12, R22, R114 and R502 where true superheated values of specific heat are evaluated. If any of the above assumptions introduces an unacceptable level of error into the solution then the user-defined property facility can be used to override the default property description; see Section 4.2.2, below.

Fluid Type	State*	Temperature Range / °C	Pressure Range /Pa	Enthalpy Datum State§
WATER	L V	0.01 – 350.0 0.01 –350.0	$610.0 – 20.0 \times 10^6$ $610.0 – 10.0 \times 10^6$	h_l = 0.61 J/kg @ 0.01°C, 0.0061 bar‡
AIR	V	0.0 – 1500.0	$610.0 – 20.0 \times 10^6$	h_v = 1111.6 J/kg @ 1 K†
AMMONIA	L, V	-77.7 – 90.0	$10.0 \times 10^3 – 11.0 \times 10^6$	h_l = 0 J/kg @ 25°C, 10.0 bar
R11	L V	-73.3 – 197.7 15.5 – 198.6	$610.0 – 4.1 \times 10^6$ $610.0 – 4.2 \times 10^6$	h_l = 0 J/kg @ -40.0°C, 0.05 bar
R12	L V	-95.5 – 112.2 -28.8 – 111.5	$610.0 – 4.0 \times 10^6$ $610.0 – 4.1 \times 10^6$	h_l = 0 J/kg @ -40.0°C, 0.64 bar
R21	L V	-85.0 – 178.9 15.0 – 178.9	$610.0 – 5.1 \times 10^6$ $610.0 – 5.1 \times 10^6$	h_l = 0 J/kg @ -36°C, 0.12 bar
R22	L V	-73.3 – 96.1 -40.0 – 95.9	$610.0 – 5.1 \times 10^6$ $610.0 – 4.8 \times 10^6$	h_l = 0 J/kg @ -46°C, 0.79 bar
R114	L V	-73.3 – 145.6 4.0 – 145.6	$610.0 – 3.1 \times 10^6$ $610.0 – 3.1 \times 10^6$	h_l = 0 J/kg @ -40°C, 0.13 bar
R502	L V	-74.0 – 82.2 -52.0 – 82.0	$610.0 – 3.9 \times 10^6$ $610.0 – 4.0 \times 10^6$	h_l = 0 J/kg @ -40°C, 1.3 bar

* L=liquid, V=vapour
§ Subscript l,v = liquid, vapour saturation, respectively
‡ By definition, $u_l = s_l = 0$ for water at the triple point
† Air is treated as an ideal gas with an artificial saturation line at 1 K

Table 4-1. Applicability of property data

4.2.2 User-defined Fluid Properties

Overview

The user may describe totally new fluids or override properties of fluids that are defined in the system library. User definition of fluid property data is by way of a text file using a

syntax specially designed for this purpose. The text file is converted by a utility provided with ESATAN into a set of Fortran functions; these are compiled into a library that can be accessed at solution time.

The start of a fluid definition is marked by the keyword `$FLUID` followed by the name of the fluid, and the end by `$END_FLUID`. Within a fluid each property is defined in a block marked by a specific keyword, for example `$RHO` for density or `$CP` for specific heat. The property is defined separately for each phase, i.e. liquid, vapour, two-phase, saturated liquid and saturated vapour, and the data may be given in a number of different ways such as tabulated values for interpolation. Pressure, temperature, enthalpy or vapour quality may be used as the independent variable(s).

As a simple example, the following defines liquid density for a fluid called MYFLUID:

```
$FLUID MYFLUID
$RHO
$LIQUID
FINTRP1(T, 1)
0.0, 540.0,   10.0, 535.0,   25.0, 528.0,   40.0, 521.0;
$END_FLUID
```

Here, the data is presented as (temperature, density) pairs for linear interpolation. This will be converted into a Fortran function which encapsulates the data and calls appropriate routines to perform the interpolation.

A complete fluid definition is shown in Appendix I. The system fluid property definitions can be found in the ESATAN installation in a subdirectory called flpDefs.

Format

Formally, the syntax of a fluid-property definition is as follows:

```
$FLUID fname
[# comment]
     $prop
          $phase
          [FPABS = r-val;]
          [FTABS = r-val;]
          dopt
          data

          $phase
          ...

     $prop
     ...

$END_FLUID
```

where:-

i) *fname* is an alphanumeric identifier of up to 24 characters
ii) $*prop* signifies the fluid property being defined, e.g. density or specific heat
iii) $*phase* denotes the fluid phase, i.e. liquid, vapour, etc.
iv) FPABS and FTABS specify the reference pressure and temperature, respectively, assumed in the property definition. If not given, FPABS defaults to 0.0 and FTABS to 273.15.
v) *dopt* specifies the form of the fluid property data: constant, interpolation, etc.
vi) *data* is the fluid property data represented in the form denoted by *dopt*.
vii) Whitespace and indentation are not significant.
viii) Comments, preceded by a '#', can be placed anywhere.

The structure is hierarchical: a phase block must be contained within a property block, which must be within a fluid block. Any number of fluids may be defined in a single file.

Fluid Block

The start of a fluid definition is marked by the keyword $FLUID followed by the name of the fluid (up to 24 characters). The end of the fluid definition is marked by $END_FLUID.

Property Block

Within a fluid each property is defined in a block marked by a specific keyword, for example $RHO for density.

The set of fluid properties supported by ESATAN/FHTS is given in Table 4-2, with the corresponding keywords. They may be defined in any order. Not all properties are essential for solution: for those that potentially may be left undefined, the accompanying remarks in the table indicate when they are needed. Generally, a property may be left undefined for phases not being modelled; however, saturation enthalpy and saturation temperature are always required. An error message will be issued at model solution time if a property required by the solution is undefined.

Phase Block

There are six possible phases for a fluid: *(subcooled) liquid*, *saturated liquid*, *two-phase (mixture)*, *saturated vapour*, *(superheated) vapour* and *saturation*. The corresponding keywords are given in Table 4-3. The special case of *saturation* refers to when the property has the same value for saturated liquid and saturated vapour (and, of course, all two-phase states in between); for example, saturation temperature depends on pressure only and is independent of vapour quality. The phases may be defined in any order.

Reference Values

By default, in defining the property data pressures are assumed to be absolute and temperatures to be on the Celsius scale. The optional keywords FPABS and FTABS, which follow the phase keyword, allow the user to change this in a way analogous to the ESATAN control constants PABS and TABS. In the following example, pressures are in gauge and temperatures absolute:

Property	Keyword	Units	Remarks
Density	`$RHO`	kg/m^3	Mandatory
Specific heat (at constant pressure)	`$CP`	J/kg K	Mandatory
Thermal conductivity	`$COND`	W/m K	Needed only for heat-transfer coefficients
Dynamic viscosity	`$VISC`	kg/m s	Mandatory, but used only for frictional pressure loss and heat-transfer coefficients
Surface tension	`$SIG`	N/m	Only saturated-liquid value is needed, for intrinsic two-phase heat-transfer coefficients. Physically meaningless for vapour.
Specific enthalpy	`$ENTH`	J/kg	Mandatory, but evaluated in single-phase solution for information only.
Temperature	`$TEMP`	°C or K [a]	Evaluation of liquid or vapour values not needed for single-phase solution. Value the same for saturated liquid and saturated vapour.
Pressure	`$PRES`	Pa (absolute or gauge[b])	Evaluation of liquid or vapour values not needed for solution. Value the same for saturated liquid and saturated vapour; required for two-phase heat-transfer coefficients and humidity solution.
Joule-Thompson (or Joule-Kelvin) coefficient	`$JT`	K/Pa	Needed for general transient solution and single-phase heat-transfer coefficient (laminar flow, free convection); evaluated in liquid and vapour phases. Saturation value only used for evaporative links.
Isothermal compressibility	`$KT`	1/Pa	Needed for general transient solution only; evaluated in liquid and vapour phases.

Table 4-2. Fluid properties

a. Depending on the reference value, `FTABS`

b. Depending on the reference value, `FPABS`

```
$VISC
$VAPOUR
FPABS = 1.01325D5;     # 1 atm
FTABS = 0.0;           # 0 K
...
```

Phase	Keyword
Liquid	$LIQUID
Saturated liquid	$SAT_LIQ
Two-phase	$TWO_PHASE
Saturated Vapour	$SAT_VAP
Saturation	$SAT
Vapour	$VAPOUR

Table 4-3. Fluid phases

The scope of FPABS and FTABS is the current phase block only. Note that they apply to both the independent variables P and T and the return value of the $PRES and $TEMP blocks.

Data Options

There are five options for presenting the data, described below. Where applicable, independent variables *var*, *var1*, etc., may be any one of P, T, H and X for pressure, temperature, enthalpy and vapour quality, respectively. (Note, however that enthalpy, H, is usually redundant and is set to zero: most properties are assumed to depend on pressure, temperature and quality. Only when the property being calculated is itself either pressure or temperature is H assigned a meaningful value.) As explained in "Reference Values", P and T are in the units scale determined by FPABS and FTABS, respectively. Note that, except for PROC, all data is terminated by a semi-colon.

i) CONSTANT
r-val;

The property has a constant value, e.g.

```
$CP
$SAT_LIQ
CONSTANT
123.4;
```

ii) FINTRP1(*var, i-val*)
r-val, r-val, r-val, r-val, ... ;

The property is evaluated by interpolation on the independent variable *var*, the order of interpolation being the second parameter after the FINTRP1 keyword. The data is given as pairs of values, the first of each pair being the independent variable and the second the value of the property at that point. For example, quadratic interpolation of saturated liquid enthalpy on pressure would look like this:

```
$ENTH
$SAT_LIQ
```

```
FINTRP1(P, 2)
0.5E5, 1.2E3,   1.0E5, 1.5E3,   2.0E5, 2.3E3,   3.0E5, 3.4E3;
```

iii) `FINTRPA(var, i-val)`

```
r-val, r-val,
r-val, r-val, r-val, ... ;
...
```

The property is evaluated by interpolation on the independent variable *var*, which is assumed to be at fixed intervals. The order of interpolation is the second parameter after the FINTRPA keyword. The data is given as a list of values, the first two of which are the base value and interval length of *var*, and the remainder are the property values. E.g.

```
$PRES
$SAT
FINTRPA(T, 2)
# 1st band: -10.0 to 30.0 deg C in steps of 10.0
-10.0, 10.0,
0.0015D5, 0.0120D5, 0.0282D5, 0.0472D5, 0.0788D5;
# 2nd band: 30.0 to 50.0 deg C in steps of 5.0
30.0, 5.0,
0.091D5, 0.122D5, 0.234D5, 0.366D5, 0.501D5;
```

Note that several bands of data can be defined, each using a different interval length.

iv) `FINTRP2(var1, var2, i-val)`

```
      var1 =     r-val, ... ,
var2 = r-val,    r-val, ... ,
...
var2 = r-val,    r-val, ... ;
```

The property is evaluated by bivariate interpolation on the independent variables *var1* and *var2*, the order of interpolation being the third parameter after the FINTRP2 keyword. The data is given in tabular form as illustrated in the following example:

```
$RHO
$VAPOUR
FTABS = 0.0; # Absolute temperatures
FINTRP2(P, T, 1)
   P =    0.1E5,   1.0E5,   5.0E5,  10.0E5,
T = 300.,  .123,    1.09,    3.89,    6.50,
T = 350.,  .094,    .841,    2.64,    5.12,
T = 375.,  .086,   . 711,    1.94,    3.70;
```

v) `PROC`

```
cseg
END_PROC
```

The property is evaluated by executing a Fortran 77 code segment, *cseg*, which is essentially a procedure without the SUBROUTINE or FUNCTION declaration and the final RETURN and END statements.

Double-precision variables P, T, H, X are predefined to give the state of the fluid. In addition, the integer variables FTI and NODE hold the fluid-type number corresponding to the name of the fluid, and the node reference (internal node number) when applicable, respectively; these are for use in calling relevant ESATAN/FHTS library routines.

On occasion it may be desired to convert a pressure or temperature to the same scale as used in the model, for example when using the fluid property interface functions (Section 6.4.39). For this purpose, the double precision variables PABS and TABS may be used, each one holding the value of the ESATAN control constant of the same name. Thus, (P - PABS) and (T - TABS) give pressure and temperature, respectively, in model units.

Two further variables are available, for use within the $TWO_PHASE, $SAT_LIQ and $SAT_VAP blocks only. Named after the **current** property, with either an L or a V appended, these contain the saturated-liquid and saturated-vapour values*, respectively; i.e. RHOL, CPV, etc. (As should be obvious, a saturation value referenced in this way must be defined in its own block. Also, the saturated liquid value of a property cannot be obtained by this method within its $SAT_LIQ block, nor similarly for the saturated vapour value.)

Any ESATAN/FHTS library routine or external user routine can be called from within a PROC definition.

The last line of the procedure should be the assignment of the property value, the left-hand side of which is the name of the current property.

The following example, defining enthalpy, should make the use of the PROC data option clear.

```
$ENTH
$SAT_LIQ
# Must be defined for ENTHL below
...

$TWO_PHASE
# Interpolate on vapour quality
PROC
      ENTH = ENTHL + X * (ENTHV - ENTHL)
END_PROC

$SAT_VAP
# Must be defined for ENTHV above
...

$VAPOUR
#
# h = h_s + Cp_s (T - T_s)     (approx.)
#
```

* Evaluated at the current pressure/temperature.

```
FPABS = 0.0;      # P absolute
FTABS = 0.0D0;    # T absolute
#
PROC
      DOUBLE PRECISION CPS, HS, TS, XCPS, XENTHS, XTEMPS
      TS = XTEMPS(P - PABS, FTI, NODE) + TABS
      HS = XENTHS(P - PABS, TS - TABS, X, FTI, NODE)
      CPS = XCPS(P - PABS, TS - TABS, X, FTI, NODE)
C
      ENTH = HS + CPS * (T - TS)
C
END_PROC
```

It is the user's responsibility to ensure that the code between `PROC` and `END_PROC` is valid Fortran 77—this will not be checked during the conversion of the fluid property definition file. In particular, note that the normal Fortran 6-space indentation must be maintained. Also, '`#`' does **not** denote a comment within a procedure.

The user should avoid declaring local variables in a procedure with the names given in Table 4-4.

COND	CONDL	CONDV	CP	CPL	CPV
ENTH	ENTHL	ENTHV	FTI	H	JT
JTL	JTV	KT	KTL	KTV	NODE
P	PABS	PRES	PRESL	PRESV	QPnnnn
RG	RHO	RHOL	RHOV	SIG	SIGL
SIGV	T	TABS	TEMP	TEMPL	TEMPV
VISC	VISCL	VISCV	X	ZFTI	ZH
ZNODE	ZP	ZT	ZX		

Table 4-4. `PROC` reserved names

Creating a Fluid Property Library

Once a set of fluid properties has been defined it is converted into a library archive file by choosing the appropriate option from the ESATAN user interface. Several fluid definitions can be incorporated into a single library, and the user has the option of including the ESATAN/FHTS system fluids.

It is quite valid for properties of the same fluid to be defined in more than one file. Where a property is defined twice for a given fluid, the fluid property file named first in the list has higher priority. System fluid properties, if selected, have lowest priority and hence can be overridden by the user.

The first step is to translate the property definitions into a set of Fortran functions. Each fluid is allocated a two-digit integer code in the range 01–99. Every phase block defined results in a separate Fortran routine being generated, called a property *data* function, and a family of property *interface* functions is produced which provides the link between the solution modules and the low-level data functions.

Let us expand the earlier MYFLUID example a little:

```
$FLUID MYFLUID
$RHO
$LIQUID
FINTRP1(T, 1)
0.0, 540.0,   10.0, 535.0,   25.0, 528.0,   40.0, 521.0;
$CP
$SAT_LIQ
CONSTANT
123.4;
$END_FLUID
```

On converting this fluid definition, two data functions will be produced: QP0101 and QP0107, corresponding to liquid density and saturated-liquid specific heat, respectively, for the fluid MYFLUID (the naming scheme of these functions need not concern the user). There will also be a full set of interface functions, amongst which will be XRHO and XCPS. Now, XRHO is called from the solution whenever single- or two-phase density needs to be evaluated; in this example, if the vapour quality is zero then control is passed to QP0101 to carry out the calculation, otherwise an 'undefined property' error is reported. Similarly, XCPS is called whenever saturation specific heat is required; again, only if the vapour quality is zero will a value be returned here, control being passed to QP0107 to carry out the calculation (trivial as it is in this instance).

It is important to realise that the property interface functions are also intended to be user-callable from within either an ESATAN/FHTS model's operations blocks, or a fluid property definition using the PROC option (as illustrated in "Data Options"), or the user's own external Fortran. (In the first case the functions do not need to be declared; in the latter two cases, they do). See Section 6.4.39 for details of each interface function.

A log file, flp.log, is produced in the current working directory as the fluid property definitions are converted. Diagnostic information, such as the name of the Fortran file generated from each phase block and any errors found in the fluid property definitions, is recorded. The log from the preceding example, assuming it is saved in a file called myfluid.flp, looks like this:

```
ESATAN/FHTS Fluid Property Convertor version 1.0
Run date: Tue Mar  8 13:09:39 2005


Parsing file:
myfluid.flp

Fluid: MYFLUID
----Property: RHO
```

```
--------Phase: LIQUID
        Data option: FINTRP1
        File generated: qp0101.f
----Property: CP
--------Phase: SAT_LIQ
        Data option: CONSTANT
        File generated: qp0107.f

No errors found
```

The generated Fortran itself is preserved in a sub-directory called flptmp; this can be safely deleted if desired.

The next step in the process is to compile the generated Fortran code and place the resulting object files into a single archive, the name of which is specified by the user*; this is known as the fluid property library. It may be used immediately with a model in the current working directory, or it may be saved for later use.

Using a Fluid Property Library

The ESATAN user interface allows any previously created library to be selected for use, either the system library (assumed by default) or a user-defined one. It does not have to be in the same directory as the model, so long as the correct path is given. The specified library will be recorded in the current working directory and hence maintained for each model between ESATAN sessions.

The required library must be selected prior to running a solution (more precisely, before the compile and link step). It is not relevant to the preprocessing of a model.

Restrictions

A maximum of 99 fluids can be defined in a single library.

The maximum order of interpolation for the `FINTRP1`, `FINTRPA` and `FINTRP2` data options is 10.

AIRW is a reserved name for a fluid type, and should only be used to describe the two-component fluid consisting of air and water vapour. Its usage with any single-phase solver invokes the humidity solution (Section 4.4), for which the fluid type WATER must also be defined.

A property interface function (see “Creating a Fluid Property Library”) cannot be called from the corresponding property definition, as this would lead to recursion in the generated Fortran. For example, function `XRHO` must not be used in `$LIQUID`, `$TWO_PHASE` or `$VAPOUR`

* On some platforms this must be given a specific extension; e.g. Compaq Alpha Unix requires archives to end with .a

sub-blocks of $RHO; it can, however, be used in $SAT_LIQ or $SAT_VAP for that property, or in any sub-block within the definition of any other property; specific heat ($CP), for example.

4.3 Modelling Techniques

Modelling of fluid systems is inherently more difficult than that of conventional 'solid' or thermal models. This is because three dependent variables must be considered simultaneously—pressure, mass flow rate, and temperature (or enthalpy for two-phase conditions)—whereas only the latter is present in thermal models. Calculation of these variables is made from the solution of three conservation equations (momentum, mass, and energy), and each of these equations is a function of two or more of the variables. This section presents guidelines for the successful solution of an FHTS model.

If a hydraulic steady state is being performed then each fluid circuit must contain a pressure-boundary (J- or R-node) to provide a datum for the system. This applies to all of the single-phase solvers and the two-phase steady state module FGENSS. When using the two-phase transient solver FGENFI it is not mandatory to define a pressure boundary, although one can be used to model an accumulator which, for two-phase simulation, is usually essential. The Engineering Manual explains this more fully.

For the single-phase solution routines, it is worth bearing in mind that initial conditions are not vitally important hydraulically. Specification of initially consistent conditions will lead to a faster solution, but inconsistencies are normally smoothed out by the end of the first time step or within a steady state solution. This may not however be true when using functions to calculate losses at tee-pieces, such as the formulas described in Appendix L: an inconsistent initial definition of flow distribution may lead to instability problems.

Also, the user should be careful with the thermal loading of the system. It is quite easy to construct a model in which condensing or boiling conditions will occur, due to incorrect heat exchanger sizing, boundary conditions, etc. Clearly, this is incompatible with a solution that assumes single phase. Consideration of the implications of the model in this respect is well worthwhile before submitting it for solution.

The specification of ’rough’ initial hydraulic conditions—sharp pressure drops or mismatched flow rates, for instance—in a full time-dependent simulation can be equivalent to a severe transient being imposed. Thus, except where the simulation is required to begin from a precise, prescribed state, it is recommended that the steady-state solver FGENSS be run prior to calling the transient FGENFI with QTRSOL = ’NO’.

For both classes of solver, the method of solution assumes weak coupling between the equations governing the thermal solutions (fluid and solid) and the hydraulic solution. Damping within each inner solution is generally provided by the control constants DAMPM and DAMPT (see the appropriate text on each solver in Section 6.15 for details). These can be important in obtaining stable solutions for larger fluid networks. A user experiencing such instability problems should consider under-relaxing the solution by specifying a value of less than unity for each of these (in fact, DAMPM defaults to 0.5).

The decoupling of the thermal and hydraulic solutions can lead to problems with models containing strong thermal-hydraulic links (e.g. temperature-control valves), and so greater management of the solution procedure can be achieved in two ways:-

1. Limiting the number of hydraulic and thermal inner iterations.
2. Distinguishing between code in $VARIABLES1 which is specific to the hydraulic solution, the thermal solution, or neither.

NLOOP defines the maximum number of outer iterations. NLOOPH and NLOOPT define the maximum number of inner iterations (hydraulic and thermal, respectively) allowed for each outer iteration. NLOOPH and NLOOPT default to NLOOP, but reducing them can aid overall convergence by in effect damping between the three inner solutions.

The user's code in $VARIABLES1 can be selectively executed by testing the value of the control constant SOLTYP. Generally speaking, this is set to ' ' (blank) at the start of each outer iteration, 'THERMAL' during the thermal solution (both fluid and solid), and 'FLUID' during the hydraulic solution. Again, see Section 6.15 for details.

The total number of outer iterations is recorded in the control constant LOOPCT (reset to zero at the start of each timestep in a transient). The total number of inner hydraulic and thermal iterations is recorded in SUMFLL and SUMTHL, respectively.

4.4 Air/water-vapour Loop Modelling

The fluid type AIRW—wet (or moist) air—allows the simulation of air/water-vapour loops, including calculation of humidity and condensation rates. This capability is supported by the single-phase solvers (FLTNSS, FLTNTF, FLTMTS and FLTNTS). This section gives a brief discussion of the concepts involved and their modelling in FHTS; for more detail, see the Engineering Manual.

Relative humidity, commonly denoted by the Greek letter ϕ (*phi*), is defined as the ratio of water vapour partial pressure to saturation pressure. It is thus independent of the amount of air present, and gives an indication of the ease with which the atmosphere will take up moisture, i.e. how close to saturation it is. A relative humidity of 100% means that the atmosphere is saturated. In theory supersaturation ($\phi > 1.0$) is possible if the air/vapour mixture is pure, but in practice impurities such as dust particles act as nucleation sites, and condensation within the bulk of the fluid takes place; that is to say, fog forms.

Specific humidity, on the other hand, provides a measure of the relative amounts of air and water vapour present; it is defined as the ratio of the mass of water vapour to the mass of air in a given volume, or, equivalently, the ratio of the respective mass flow rates.

In an FHTS model the nodal entity PHI represents the relative humidity, and is calculated by the solution for all fluid nodes except those of type 'R'. For R-nodes PHI is fixed and should be specified appropriately in the input file. The specific humidity of a node is available to the user via the function SHUM.

If a mass source (FM>0) is defined at a node then the proportions of water vapour and air being input must also be given. This is done via the entity FW, which in the special case of AIRW represents the specific humidity of the source. It can be shown that the rate of inflow of water vapour is then given by FM*FW/(1+FW), whilst that of air is given by FM/(1+FW). To specify pure air FW should be set to zero; for pure water vapour FW would in theory be infinite, but in practice a value of 10^{15} is sufficient.

A mass sink (FM<0) extracts air/vapour mixture at the specific humidity of the node, and so in this case the value of FW is irrelevant. To remove pure water vapour the subroutine WVSINK must be called.

Where a fluid node is linked to a solid node via a GL conductor there is the possibility of condensation occurring. This happens when the solid node temperature is below the dew-point of the fluid. The rate of condensation is calculated, being proportional to the GL value, and also the rate of liberation of the accompanying latent heat; the latter is added as a heat input to the solid node. There may be any number of condensation surfaces, i.e. solid nodes, linked to each fluid node, and vice versa.

Water vapour can also be removed from a network by use of the condensing heat exchanger element CHX. The output routine HUMDMP provides a summary of information relating to humidity and condensation for all fluid nodes in a model.

Note that the method used to calculate humidity at each node effectively precludes the modelling of closed loops in steady state, i.e. with solver FLTNSS.

4.5 Evaporative Links

As described in Section 4.1, a mass flow link normally satisfies conservation of momentum: essentially, the pressure difference across the link determines the flow rate through it. In certain circumstances, however, a flow rate may be induced in a quite different way. An example of this is in a capillary evaporator, where the rate of evaporation of fluid through the liquid/vapour interface is a function of the applied heat flux.

FHTS provides a way of modelling this, with the *evaporative link*. This is a specialised form of mass flow conductor, the normal form of which may be referred to as a *momentum link* for distinction. Any mass flow link M(i, j) may be designated as evaporative by use of the library routine MTYPST in an operations block (see Section 6.4.36). The solution will then calculate the flow rate according to the equation

$$W = \frac{Q}{h_{\mathrm{v}} - h_{\mathrm{in}}}$$

where Q is the heat flux, h_{in} the enthalpy at the first node and h_{v} the enthalpy of saturated vapour at the pressure of the second node. The link will supply saturated vapour to the downstream node. The heat flux is applied via the library routine EVLQST (Section 6.4.37).

A mass flow link may be changed from evaporative back to momentum type, whereupon the flow rate wil once more be computed according to the momentum equation.

An evaporative link can, of course, only be used with a two-phase solver. Currently only FGENSS and FGENFI support this facility.

4.6 Phase Separation

Modelling of certain devices, for instance a capillary filter, requires that only fluid of a certain phase (liquid or vapour) is allowed to leave a node. FHTS provides the facility of *phase separation* for this, whereby the user can specify the vapour quality of the fluid flowing in a link.

Consider a node containing two-phase fluid, as shown in Fig. 4-3; although the mixture is in fact modelled as homogeneous in FHTS, for our present purposes it can be imagined that the liquid and vapour phases are in separate layers, perhaps due to gravity. Suppose the node was initially full of liquid, but then the fluid in the upstream link started to flow at 50% vapour quality. It is required that the downstream mass-flow link shall allow the passage of the liquid phase only. Clearly, if two-phase fluid continues to enter at the same rate then the proportion of vapour in the node will rise.

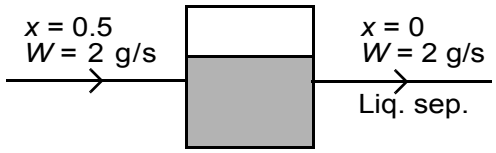


Figure 4-3. Liquid-phase separation

An example of a phase separation situation which has reached steady state is shown in Fig. 4-4. Here, a two-phase mixture of 50% vapour quality enters the node; liquid is drawn off along one exit link, leaving all of the vapour plus the remaining liquid to flow along the other. The vapour quality in the latter is therefore higher than on entry, and since this link simply draws off fluid at the mixture proportions it sees at its upstream end, the nodal vapour quality must also be more than 50%. This apparent paradox can be understood by

considering that the phase-separating link draws off as much liquid as it requires from the inflow *before* the inflow mixes with the fluid already in the node.

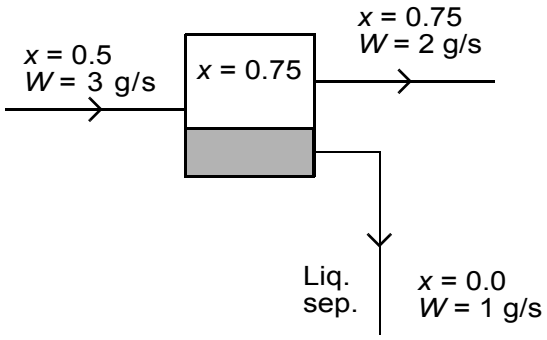


Figure 4-4. Phase separation: steady state example (1)

With a steady state simulation there is a complication in that there may not actually be a true solution. Consider again the arrangement in Fig. 4-3. In steady state the flow rate entering the node must equal the flow rate at exit (mass conservation); however, it is required that all of the fluid leaving be liquid, whereas only half of that entering is in this phase. A time-independent solution does not exist in this case.

In such circumstances, i.e. where there is insufficient liquid or vapour entering the node to satisfy the designated phase separation, a time-averaged solution is sought in which mass and energy are conserved. Heuristically, in the above example the node will be drained of liquid until it becomes pure saturated vapour, at which point the downstream link will begin to carry vapour. But during the next time interval a small amount of liquid will become available, allowing the phase-separation link to convey liquid-only once more. Thus, the fluid in this link will oscillate between liquid and vapour. In the limit as the time-slices get smaller and smaller, the node will contain only vapour while the downstream link will have a quality of 50%, indicating that it actually conveys vapour half of the time (Fig. 4-5).

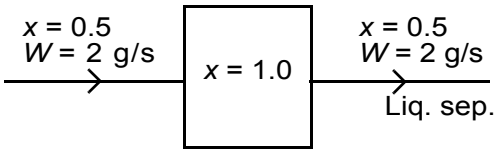


Figure 4-5. Phase separation: steady state example (2)

A more general example is shown in Fig. 4-6. Here there are two downstream links, one ordinary and one liquid-separation; the former has half the flow rate of the latter. Again, the node is driven to a vapour quality of 100% but this time the liquid-link carries vapour 25% of the time.

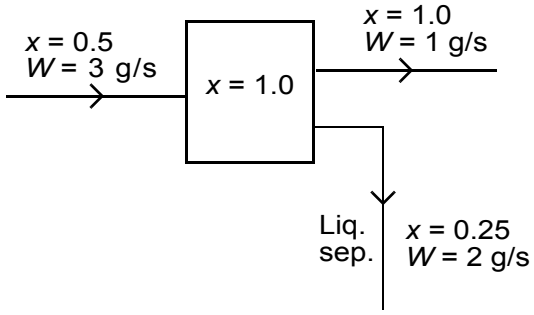


Figure 4-6. Phase separation: steady state example (3)

If, say, liquid separation is specified but only pure vapour is available then the link will revert to normal behaviour, allowing the vapour to pass.

Although in the above we have assumed *liquid* separation to illustrate the facility, the user may specify that only the vapour component is allowed to flow along the link, or, more generally, a particular mixture, i.e. $0 < x < 1$. This allows more realistic modelling of, say, a liquid separator where a small amount of vapour is drawn off as well.

Any mass flow link may be designated for phase separation, unless it is of evaporative type (Section 4.5). This is achieved by calling the library subroutines PHSDST and PHSXST (Section 6.4.35) from an operations block. The first defines the direction for phase separation to take effect, the second the required vapour quality; it is possible to have phase separation in one direction only, or liquid separation in one direction and vapour separation in the other.

Phase separation can, of course, only be used with a two-phase solver. Currently only FGENSS and FGENFI support this facility.

5. SYSTEM ELEMENTS

As mentioned in Section 3.4 a library of components is provided for inclusion into a model by using the $ELEMENT keyword. The elements are contained in the file ELEMSYS.DAT which is supplied with the ESATAN installation. Users should note that an element used in this way becomes a standard ESATAN/FHTS submodel, and will be a direct submodel of the model in which it is defined.

The components listed below belong either to ESATAN (Peltier and PID) or FHTS. When used in a fluid loop the FHTS components are connected hydraulically by defining mass flow links appropriately in the main model. Similarly, walls, casings, etc., may easily be coupled to the rest of the thermal network by defining conductors in the main model. Referencing of entities within the element submodel is possible in the conventional manner.

The following sections describe the definition of each element's characteristics in both engineering terms and ESATAN input terms. Lists of substitution data for each element are provided along with default values where applicable. Substitution data with no default value are shown in **bold**. For fluid-loop elements, unless otherwise stated, the fluid type for the submodel must be given on the $MODEL line.

Here is a list of ESATAN/FHTS elements:

Tube	Fluid-filled pipe, including walls
Pumps	Centrifugal pump
Valves	Two- or three-way valve
Evaporators	U-tube evaporator (one or multiple U)
Tees	Two- or four-node tee pieces
Transfer Function Heat Exchanger	Heat transfer by the method of Kays & London,
Annular Heat Exchanger	Parallel or counterflow concentric-tube heat exchanger
Cross-Flow Heat Exchanger	Tube banks within a containing shell
Condensing Heat Exchanger	Counterflow condensing heat exchanger for air/water-vapour loops
Heat Pipe-Bank Condensing Radiator	Tube connected to a heat pipe panel
Panel Condensing Radiator	2D panel with embedded fluid tubes
Ideal Accumulators	Constant pressure/temperature accumulators
Non-Ideal Accumulators	Active/passive accumulators
Capillary Evaporator	Two-phase device imposing massflow from heat input
Capillary Filter	Isolator; filters vapour from two-phase fluid
Two-phase Reservoir	Thermal accumulator
Flexible Hose	Flexible, corrugated hose
Peltier Element	Thermoelectric heat exchanger
PID Controller	Three-term PID controller

5.1 Tube

General Concepts:-

The tube element models a fluid-filled pipe, including the wall. There may be convective heat transfer between the wall and the fluid, as well as frictional pressure losses. By default the internal FHTS correlations are used, but the user may override these. As well as being used to model liquid/vapour transport lines, the element can be used to simulate a simple condenser or evaporator.

The wall is taken as radially isothermal, but conduction in the axial direction is allowed. Heat may be applied either directly to the wall or by coupling the tube element to the rest of the thermal network.

A constant circular cross section is assumed by default but the element allows the user to define a non-circular cross section.

ESATAN Input:-

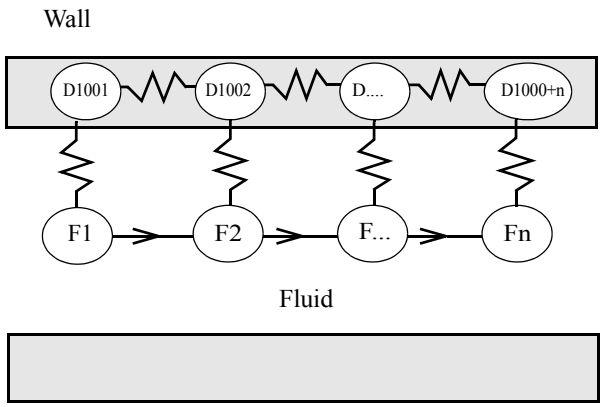


Figure 5-1. Tube element

$ELEMENT name:-

TUBE Tube

$SUBSTITUTIONS data:-

AX_COND	=	r-val;	(wall axial conductance)
C_WALL	=	r-val;	(wall capacitance)
DIAM	=	r-val;	(tube diameter)
ETA_TUBE	=	r-val;	(overall flow resistance coefficient)
FLA	=	r-val;	(flow area)

HTC_METH	=	z-val;	(heat transfer coefficient calculation method: '*' or 'USER')
HTC_USER	=	r-val;	(user heat transfer coefficient value)
LENGTH	=	r-val;	(tube length)
MFLOW	=	r-val;	(initial mass flow rate)
NMESH	=	i-val;	(meshing—number of nodes in the tube)
N_END	=	i-val;	(fluid outlet node number)
PRESS	=	r-val;	(initial fluid pressure)
Q_LOAD	=	r-val;	(heat load on the wall)
RUG	=	r-val;	(surface roughness of the tube)
TEMP	=	r-val;	(initial fluid temperature)
HT_AREA	=	r-val;	(total heat transfer area)
T_WALL	=	r-val;	(initial wall temperature)
X_END Y_END Z_END	=	r-val;	(x,y,z-coordinates at tube end)
X_START Y_START Z_START	=	r-val;	(x,y,z-coordinates at tube start)

Defaults provided for this element are:-

AX_COND	=	0.0
C_WALL	=	0.0
ETA_TUBE	=	0.0
FLA	=	$\pi \cdot DIAM^2 / 4$
HTC_METH	=	'USER'
HTC_USER	=	0.0
MFLOW	=	0.0
NMESH	=	3
N_END	=	NMESH
Q_LOAD	=	0.0
RUG	=	0.0
HT_AREA	=	$\pi \cdot DIAM \cdot LENGTH$
T_WALL	=	TEMP
X_END	=	LENGTH
Y_END	=	0.0
Z_END	=	0.0
X_START Y_START Z_START	=	0.0

Input user constants:-

Htc_meth	(heat transfer coefficient calculation method)
Htc_user	(user heat transfer coefficient value)
Q_load	(heat load into the wall)

Connection points:-

F1	-	Fluid inlet
F*N_END*	-	Fluid outlet
D1001	-	Wall inlet
D1002, D1003....	-	Wall body
D*(1000+N_END)*	-	Wall outlet

The tube element consists of a series of F-nodes connected by mass flow links, and a wall modelled by D-nodes connected by GL conductors.

The user can define the number of fluid nodes required in the tube using NMESH. The corresponding thermal nodes will be automatically generated. Note that NMESH must be at least 3. The node numbering starts from 1 for fluid, and 1001 for thermal nodes. NMESH includes the inlet and outlet nodes. By default, the fluid outlet will be numbered NMESH and the thermal outlet 1000 + NMESH. However, the user can easily define the outlet node number by setting N_END.

For instance, if the user defines NMESH = 5 and N_END takes the default value, the following nodes will be generated:

F1	and	D1001
F2	and	D1002
F3	and	D1003
F4	and	D1004
F5	and	D1005

On the other hand, if the user defines NMESH = 5 and N_END = 10, the following nodes will be generated:

F1	and	D1001
F2	and	D1002
F3	and	D1003
F4	and	D1004
F10	and	D1010

By defining N_END, the user is able to change the meshing within the tube without changing the numbering of the outlet. This will avoid having to change the connecting mass flow links in the main model.

A circular cross section is assumed by default. The user can easily define a non-circular cross section by setting DIAM with the appropriate hydraulic diameter and the actual flow area, FLA (see Appendix K for more information).

A heat load can be applied to the wall by setting the user constant Q_load as required. This heat load can be a constant or a variable value and will be shared equally between the thermal nodes.

The method for calculating heat transfer coefficients is specified by substitution data HTC_METH. The default is to use the user defined value (HTC_METH = 'USER'). Then the user has to define the value of the heat transfer coefficient by setting the substitution data HTC_USER (the default defines an adiabatic tube, no heat transfer between the fluid and the wall). The internal correlations provided by FHTS (see Appendix K) can be used by the user by setting HTC_METH = '*'. This can be changed during the run using the user constants Htc_meth and Htc_user.

An overall flow resistance for the tube can be defined by setting ETA_TUBE. The appropriate value will be assigned equitably onto each mass flow link by defining the corresponding GP conductance.

Care should be taken not to make the CSG value (the ratio of nodal capacitance to sum of conductances) too small, as this may cause instability in the solution. For thin-walled pipes, the capacitance C_WALL should be set to zero.

5.2 Pumps

General Concepts:-

A centrifugal pump may be used to provide the motive force in a fluid loop by performing work on the fluid in order to raise its pressure.

The pressure rise, Δp, across the pump is usually given as a function of volumetric flow rate, V, at specified conditions. This function is known as the pump *characteristic*, and includes any turbulent or frictional losses experienced by the fluid as it passes through the pump.

As the pump does work on the fluid it will put energy into it, resulting in a rise in temperature. In an ideal pump all of the power input will be converted into mechanical energy of the fluid. The *efficiency* of a pump, η, is defined as the ratio of the gain in mechanical energy to the total power input, Q, and is given by the expression

$$\eta = \frac{V\Delta p}{Q}.$$

Efficiency also is a function of volumetric flow rate.

The characteristic of a pump is normally given at one particular speed of rotation. From this, the characteristic at any other speed can be obtained using homologous theory[8]. This states that, for geometrically similar pumps, the characteristic data can be described using the dimensionless parameters

$$\Pi_1 = \eta ,$$

$$\Pi_2 = \frac{\frac{\Delta p}{\rho}}{N^2 \cdot D^2} ,$$

$$\Pi_3 = \frac{V}{N \cdot D^3} ,$$

where ρ is the fluid density, D is a dimension representative of the pump, and N is the rotational speed. D is only included if comparison between different-sized pumps of the same family is made: if homologous theory is applied to the same pump with only changes in pump speed, then D is constant and can be ignored.

Note that it is not Δp but the ratio $\Delta p / \rho$ in the dimensionless parameter Π_2. Thus, even for constant speed the characteristic should be scaled to account for changes in density from that at the reference conditions.

ESATAN Input:-

$ELEMENT name:-

PUMP_CF Centrifugal pump

$SUBSTITUTIONS data:-

CHAR_TYPE	=	z-val;	(characteristic type: tabular – 'TABLE'; polynomial – 'POLY')
DIAM	=	r-val;	(diameter)
DP_ARRAY	=	r-val, r-val, …, r-val;	(characteristic array: tabular – Δp values; polynomial – Δp coefficients)
EFF_ARRAY	=	r-val, r-val, …, r-val;	(characteristic array: tabular – efficiency values; polynomial – efficiency coefficients)
MFLOW	=	r-val;	(initial mass flow rate)
PRESS	=	r-val;	(initial pressure)
REL_HUM	=	r-val;	(initial relative humidity—AIRW only)
RHO_REF	=	r-val;	(reference density for characteristic)
SPEED	=	r-val;	(rotational speed)
TEMP	=	r-val;	(initial temperature)
VF_ARRAY	=	r-val, r-val, …, r-val;	(characteristic array: tabular only – volumetric flow rate values)
VOL	=	r-val;	(volume)
X_COORD Y_COORD Z_COORD	=	r-val;	(x,y,z-coordinates)

Defaults provided for this element are:-

CHAR_TYPE	=	'TABLE'
DP_ARRAY	=	0.0
EFF_ARRAY	=	1.0
REL_HUM	=	0.0
RHO_REF	=	RHO(F1)
SPEED	=	1000.0
VF_ARRAY	=	0.0
X_COORD Y_COORD	=	0.0

Z_COORD⌋

The pump element consists of two F-nodes connected by a mass flow link; the nodes are numbered 1 and 2 at the inlet and outlet, respectively. The calculated Δp is applied to the link via the library function SETDPV. An initial mass flow rate for the link must be specified.

The user must supply the inlet/outlet diameter—cylindrical geometry being assumed here—and the total internal volume. For modelling gravitation/acceleration effects the extent of the pump is assumed to be negligible; that is, both nodes are assigned the same coordinates. The casing of the pump is not modelled.

Two methods of representing the characteristics of a pump are available. The data may be supplied in tabular form (this is the default), with the user giving values of pressure rise and efficiency vs. volumetric flow rate:

$$V_1, V_2, \ldots, V_n$$

$$\Delta p_1, \Delta p_2, \ldots, \Delta p_n$$

$$\eta_1, \eta_2, \ldots, \eta_n \ .$$

Intermediate values are found by linear interpolation on *V;* extremal values of Δp and η are taken if the current flow rate is out of range. Note that the values for flow rate must be given in ascending order.

Alternatively, a polynomial in flow rate may be defined, with the user giving the polynomial coefficients:

$$\Delta p = a_0 + a_1 V + a_2 V^2 + \ldots + a_n V^n$$

$$\eta = b_0 + b_1 V + b_2 V^2 + \ldots + b_n V^n \ .$$

In both cases, the element will allow the characteristic to be defined for negative values of *V*, i.e. the pump will operate under conditions of reverse flow.

Initial values of pressure and temperature must be supplied. The reference density may also be supplied by the user. If it is not, then it is calculated by the element from the given conditions and stored in the user constant RHO_REF. Note that the element assumes any change in density across the pump is negligible.

The pump speed is controlled via a user constant named SPEED; this may be modified in an operations block and the element will scale the characteristic appropriately. The reference speed is stored in the user constant SPEED_REF.

Fluid type must be given on the $MODEL card.

$ELEMENT name:-

PUMPARRAY
PUMPPOLY

These elements have been superseded by PUMP_CF and should be considered obsolete.

5.3 Valves

General Concepts:-

A valve element is provided within FHTS. The method used for modelling valves involves including the pressure drop through the valve in the system flow balance. The following equation is used to characterise the pressure drop through each side of the valve:

$$\Delta p = \frac{EW^2}{\rho^2 X^2}$$

where E is the valve pressure drop factor (user-supplied constant), W is the mass flow rate and X is the valve opening fraction. The units for E are derived from the above equation and, in the case of SI units, are $Pa \cdot s^2/m^6$.

For a three-way valve the pressure drop across each side is defined similarly with X and $1-X$ as the opening fractions of each side. The user must specify whether a two-way or three-way valve is required using the substitution data VALTYP.

ESATAN Input:-

Once the pressure drop through the valve has been calculated it is converted to a GP conductor value and then incorporated into the system flow balance. The pressure drop is related to the GP value by:

$$GP = \frac{0.5\rho u^2}{\Delta p}$$

The size of valve pressure drop and hence GP conductor value are determined by the valve opening fraction.

The opening fraction can be set to unity or zero to simulate 'ON/OFF'states. This idea is used in the routine VLSTAT (see Section 6.4.27), which the user can call in a higher level model to turn the valve ON/OFF.

An alternative more versatile option is to use a modulating valve which can have any value of opening fraction between zero and one depending on the valve characteristics and flow conditions. The routine RLVALV (see Section 6.4.27), used in FHTS to calculate the opening fraction, uses a technique based on the Rate Limiting Valve method. It should be noted that as the Rate Limiting Valve method is based on the change of valve opening fraction with time, the routine RLVALV can only be used within transient analysis.

$ELEMENT name:-

VALVE Two/three-way valve

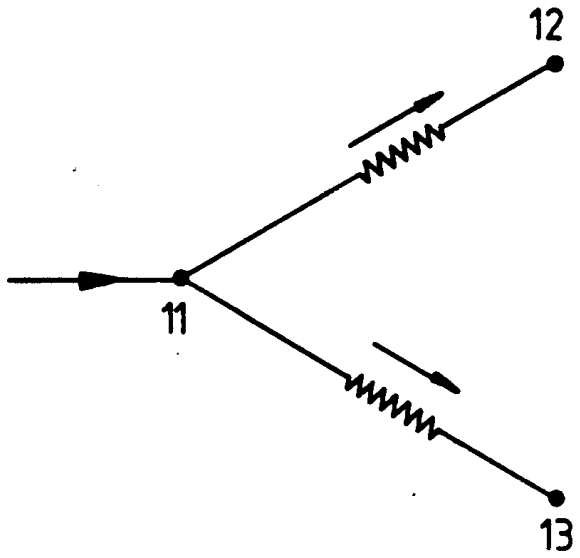


Figure 5-2. Three-way valve model

$SUBSTITUTIONS data:-

VA1, VA2, VA3	=	r-val;	(heat transfer area)
VFD1, VFD2, VFD3	=	r-val;	(hydraulic diameter)
VFLA1, VFLA2, VFLA3	=	r-val;	(flow area)
VFL1, VFL2, VFL3	=	r-val;	(length)
VFST	=	z-val;	(fluid state descriptor)
VP1, VP2, VP3	=	r-val;	(pressure)
VT1, VT2, VT3	=	r-val;	(temperature)
VFF1, VFF2, VFF3	=	r-val;	(roughness)
VFE1, VFE2, VFE3	=	r-val;	(enthalpy)
VVQ1, VVQ2, VVQ3	=	r-val;	(vapour quality)
VFX1, VFX2, VFX3	=	r-val;	(x-coordinate)
VFY1, VFY2, VFY2	=	r-val;	(y-coordinate)
VFZ1, VFZ2, VFZ3	=	r-val;	(z-coordinate)
MFLOW1	=	r-val;	(mass flow between nodes 1 & 2)
MFLOW2	=	r-val;	(mass flow between nodes 1 & 2)
EVALUE	=	r-val;	(valve pressure drop factor)
XOPEN	=	r-val;	(valve opening fraction)
VALTYP	=	z-val;	(valve type)

Note that in a three way valve the opening fraction XOPEN always refers to the main branch valve and 1 - XOPEN to the side valve. VALTYP can be set to either TWOWAY or THREEWAY depending upon the configuration required. In the two-way configuration the side branch is automatically set inactive, similarly if either branch is closed during solution the link is set inactive. The fluid state can be defined using pressure and enthalpy ('P&FE'), pressure and temperature ('P&T'), pressure and vapour quality ('P&VQ') and temperature

and vapour quality ('T&VQ') by setting the fluid state descriptor VFST to the appropriate string. No error will be generated if no state variables are defined, however the state must be defined prior to solution (i.e. within the $INITIAL block). The fluid properties (P, T, FE and VQ) of the side branch node (node 3) are set to be equal to the fluid properties of the junction node (node 1).

Default values provided for this element are:

VA1	=	$\pi \cdot VFD1 \cdot VFL1$
VA2	=	$\pi \cdot VFD2 \cdot VFL2$
VA3	=	$\pi \cdot VFD3 \cdot VFL3$
VFLA1	=	$\pi \cdot VFD1^2 / 4$
VFLA2	=	$\pi \cdot VFD2^2 / 4$
VFLA3	=	$\pi \cdot VFD3^2 / 4$
VFD3	=	VFD2
VFL3	=	VFL2
VFST	=	'P&T’
VP1, VP2, VP3	=	0.0
VFE1, VFE2, VFE3	=	0.0
VT1, VT2, VT3	=	0.0
VVQ1, VVQ2, VVQ3	=	0.0
VFF1, VFF2, VFF3	=	0.0
VFX1, VFX2, VFX3	=	0.0
VFY1, VFY2, VFY3	=	0.0
VFZ1, VFZ2, VFZ3	=	0.0
XOPEN	=	1.0
VALTYP	=	’TWOWAY’
MFLOW2	=	U

Both EVALUE and XOPEN are defined as user constants within the element and thus can be modified dynamically during solution from a parent model within user Mortran.

5.4 Evaporators

General Concepts:-

A coldplate evaporator element is provided as a simple element which the user may characterise. It is modelled as a two-dimensional plate containing a U-shaped fluid tube. Any number of such plates may be joined together in series, allowing the generation of a serpentine type of arrangement. The user may specify discrete heat sources corresponding to nodal positions within the plate.

The user provides the number of nodes in each direction on the plate, a single node being used to represent the plate thickness. Other inputs required from the user are:-

- Width
- Length
- Thickness
- Tube characteristics
- Plate material properties
- Number of passes

ESATAN Input:-

$ELEMENT name:-

COLDPLATE	U-tube evaporator.
COLDPLATE1	U-tube evaporator (one U).

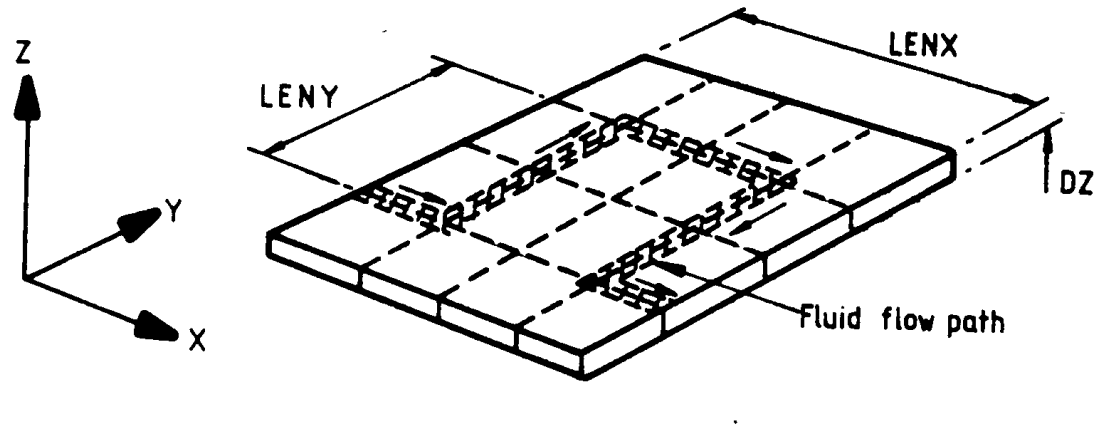


Figure 5-3. U-tube evaporator model

$SUBSTITUTIONS data:-

LENX	=	r-val;	(length of fluid tube in x direction)
LENY	=	r-val;	(length of fluid tube in y direction)
DZ	=	r-val;	(plate thickness)
NX	=	i-val;	(number of nodes in x direction)
NY	=	i-val;	(number of nodes in y direction)
ETP	=	r-val;	(plate temperature)
EK	=	r-val;	(plate conductivity)
ERHO	=	r-val;	(plate density)
ECP	=	r-val;	(plate specific heat)
EQ	=	r-val;	(total impressed heat source)
EFD	=	r-val;	(tube diameter)
EFST	=	z-val;	(fluid state descriptor)
EP	=	r-val;	(tube pressure)
EFE	=	r-val;	(tube enthalpy)
ET	=	r-val;	(tube temperature)
EVQ	=	r-val;	(vapour quality)
EFF	=	r-val;	(tube roughness)
KLOSS	=	r-val;	(total pressure loss coefficient)
EFX1, EFY1, EFZ1	=	r-val;	(coordinates of inlet bottom corner of coldplate structure)
EFX2, EFY2, EFZ3	=	r-val;	(coordinates of outlet bottom corner of coldplate structure)
EFX3, EFY3, EFZ3	=	r-val;	(coordinates of inlet top corner of coldplate structure)
NU	=	i-val;	(number of U-tubes in series)
MFLOW	=	r-val;	(mass flow rate)

Note that the above data represent the dimensions of one U-tube. The parameter NU is used to generate several identical U-tubes in series.

Fluid type must be specified on the $MODEL card. Note that all nodes are generated to have the same physical dimensions and properties.

Defaults provided for this element are:

EFST	=	’P&T’
EP	=	0.0
EFE	=	0.0
ET	=	0.0
EVQ	=	0.0
EQ	=	0.0
NU	=	2
NX, NY	=	4

ERHO = 0.0
ECP = 0.0
EFF = .0
KLOSS = 0.0
EFX1, EFY1, EFZ1 = 0.0
EFX2, EFY2, EFZ2 = 0.0
EFX3, EFY3, EFZ3 = 0.0

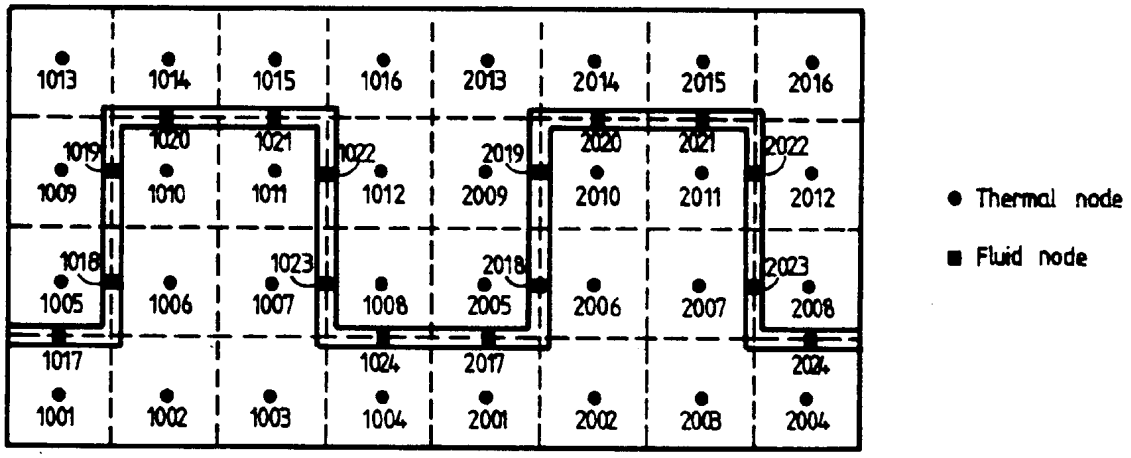


Figure 5-4. U-tube evaporator nodal discretisation

In general, each U-tube section is generated such that one row of thermal nodes is present in the x,y-plane between the edges of the coldplate and the fluid tube.

The fluid state can be defined using pressure and enthalpy ('P&FE'), pressure and temperature ('P&T'), pressure and vapour quality ('P&VQ') and temperature and vapour quality ('T&VQ') by setting the fluid state descriptor EFST to the appropriate string. No error will be generated if no state variables are defined, however the state must be defined prior to solution (i.e. within the $INITIAL block).

The total flow resistance of the coldplate can be defined via the substitution data KLOSS. From this, fitting loss conductors are generated with the appropriate conductance.

If the material properties ERHO and/or ECP are not defined, arithmetic solid nodes are assumed.

An internal heat source (QI), equally distributed over the surface of the coldplate, can be defined via the substitution data EQ. Alternatively, heat sources can be defined at specific nodes by including Mortran statements in a model containing the coldplate model, e.g.

```
$INITIAL
#
```

```
# COLDPLATE HEAT SOURCES
#
        QR:PLATE:1003 = 2.5
        QR:PLATE:2010 = 6.77
```

or, alternatively, nodes in the coldplate may be supernoded to other models or connected by intermodel links to other models.

1. NX must be a multiple of 4.
2. The node numbers are generated as shown.
3. $NY \geq NX/2 + 1$
4. $NX*NY + 2NY \leq 999$

   For connection to the adjacent fluid tubing,
   the input node number is:-

   $$1000 + NX * NY + 1$$

   and the output node is:-

   $$NU * 1000 + NX * NY + 2NY$$

5. If only one U-tube is required, then the element COLDPLATE1 must be used. COLDPLATE is for $NU \geq 2$.

   Heat transfer and pressure drop are calculated by the correlations given in Appendix K.

5.5 Tees

General Concepts:-

Elements are used to define tee-pieces. This is necessary so that the momentum flux components can be evaluated correctly by the solution routines. To do this, it is necessary that the 'side branch' is known distinctly from the main pipe.

There are two tee-piece elements; firstly, a two-node model, connected by a mass flow link. This link represents the side branch of the tee-piece. The user connects the main flow branch to the first node, and the side branch to the second. A fitting-loss conductance for the link to the side branch can be specified, if no value is defined a default value of 1.0D10 is used. This effectively represents no irreversible loss. (In that case, the default value of 1.0D10 can be relied upon). The user must specify:-

- Nodal entities defining both nodes
- Mass flow link initial value
- Fitting loss value

The user may incorporate fitting losses for the main flow branch, note that the model does not automatically incorporate this. For realistic predictions, it is recommended that the user includes some form of loss. Appendix L provides a recommended method of achieving this.

The second tee-piece model contains four nodes, three for the main branch and one representing the side branch. The centre node in the main branch is automatically generated, with the length defined as the intersection length of the side branch (user defined). This model utilises standard F-type nodes and the pressure loss coefficients are evaluated in $VARIABLES1. The pressure loss coefficients represent both the reversible loss (occurring due to the change in direction) and the irreversible losses. Due to the possible geometrical variety and the combinations of flow directions, several formulas based upon measurements and correlations are used to model the pressure losses. The angle between the side branch and the main branch of the tee piece may be between 0 degrees and 180 degrees. However, the user should take in mind that the used formulas are only valid if the flow between main branch and side branch deflects not more than 90 degrees. If this condition is violated then a warning is computed and the GP-value is set to 1.0E9.

The fluid state for both elements can be defined using pressure and enthalpy ('P&FE'), pressure and temperature ('P&T'), pressure and vapour quality ('P&VQ') and temperature and vapour quality ('T&VQ') by setting the fluid state descriptor TFST to the appropriate string. No error will be generated if no state variables are defined, however the state must be defined prior to solution (i.e. within the $INITIAL block).

ESATAN Input:-

$ELEMENT name:-

TEE 2-node tee piece
TEEFN 4-node tee piece

$SUBSTITUTIONS data:-

TEE

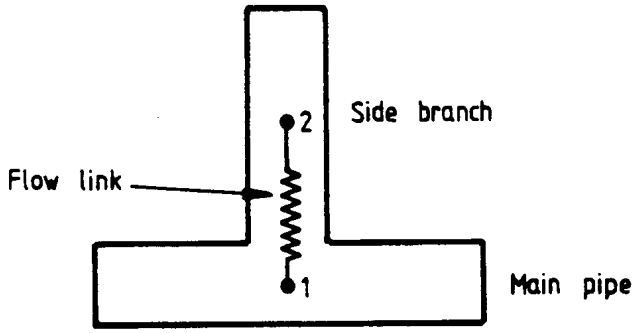


Figure 5-5. Two-node tee-piece model and nodal discretisations

TA1, TA2	=	r-val;	(heat transfer area)
TFD1, TFD2	=	r-val;	(hydraulic diameter)
TFLA1, TFLA2	=	r-val;	(flow area)
TFL1, TFL2	=	r-val;	(length)
TFST	=	z-val;	(fluid state descriptor)
TP1, TP2	=	r-val;	(pressure)
TFE1, TFE2	=	r-val;	(enthalpy)
TT1, TT2	=	r-val;	(temperature)
TVQ1, TVQ2	=	r-val;	(vapour quality)
TFF1, TFF2	=	r-val;	(roughness)
TFX1, TFY1, TFZ1	=	r-val;	(main branch coordinates)
TFX2, TFY2, TFZ2	=	r-val;	(side branch coordinates)
MFLOW	=	r-val;	(mass flow between nodes 1 and 2)
TGP	=	r-val;	(flow conductance between nodes 1 & 2)

Note that node 1 is the main pipe node, node 2 is the side branch. Fluid type must be specified on the $MODEL card.

Defaults provided for this element are:

TFX1, TFX2	=	0.0
TFY1, TFY2	=	0.0
TFZ1, TFZ2	=	0.0
TA1	=	$\pi \cdot$ TFD1 $\cdot$ TFL1
TA2	=	$\pi \cdot$ TFD2 $\cdot$ TFL2
TFLA1	=	$\pi \cdot$ (TFD1 ^{2} / 4)
TFLA2	=	$\pi \cdot$ (TFD2 ^{2} / 4)
TFST	=	'P&T'
TP1, TP2	=	0.0
TFE1, TFE2	=	0.0
TT1, TT2	=	0.0
TVQ1, TVQ2	=	0.0
TFF1, TFF2	=	0.0
TGP	=	1.0D10

TEEFN

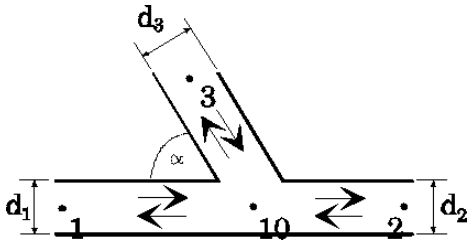


Figure 5-6. Four-node tee-piece model

TA1, TA2, TA3	=	r-val;	(heat transfer area)
TFD1, TFD2, TFD3	=	r-val;	(hydraulic diameter)
TFL1, TFL2, TFL3	=	r-val;	(length)
TP1, TP2, TP3	=	r-val;	(pressure)
TT1, TT2, TT3	=	r-val;	(temperature)
TFF1, TFF2, TFF3	=	r-val;	(surface roughness)
TFE1, TFE2, TFE3	=	r-val;	(enthalpy)
TVQ1, TVQ2, TVQ3	=	r-val;	vapour quality)
TFST	=	z-val;	(variables defined fluid state)
TFLA1 TFLA2 TFLA3	=	r-val;	(flow area)
TFX1 TFY1 TFZ1 TFX2	=	r-val;	(node 1 coordinates)

TFY2 TFZ2	=	r-val;	(node 2 coordinates)
TFX3 TFY3 TFZ3	=	r-val;	(node 3 coordinates)
TFLI	=	r-val;	(length of intersection of side branch with the main branch)
ANGLE	=	r-val;	(junction angle)

Defaults provided within this element are

TA1	=	$\pi \cdot TFD1 \cdot TFL1$
TA2	=	$\pi \cdot TFD2 \cdot TFL2$
TA3	=	$\pi \cdot TFD3 \cdot TFL3$
TFLA1	=	$\pi \cdot (TFD1^2 / 4)$
TFLA2	=	$\pi \cdot (TFD2^2 / 4)$
TFLA3	=	$\pi \cdot (TFD3^2 / 4)$
TFX1, TFX2, TFX3	=	0.0
TFY2, TFY2, TFY3	=	0.0
TFZ3, TFZ3, TFZ3	=	0.0
TFST	=	’P&T’
TFE1, TFE2, TFE	=	0.0
TT1, TT2, TT3	=	0.0
TP1, TP2, TP3	=	0.0
TVQ1, TVQ2, TVQ3	=	0.0
ANGLE	=	90°

5.6 Transfer Function Heat Exchanger

General Concepts:-

Heat exchanger performance may be evaluated using the approach developed by Kays and London[1]. This method provides a transfer function to calculate the heat flow between two fluid streams as a function of flow conditions and geometrical arrangement. The heat flow is then modelled by providing positive heat sources on the cold stream nodes, and similar negative heat sources on the hot stream nodes.

Various configurations of heat exchanger are provided, and the user must specify, for each fluid stream, nodal definitions to describe each passage of the heat exchanger.

The heat exchanger can model steady state or transient effects by including the thermal inertia of the heat exchanger. This is achieved by attaching thermal nodes to all four nodes of the heat exchanger.

ESATAN Input:-

$ELEMENT name:-

HXKL3 Transfer-function heat exchanger.

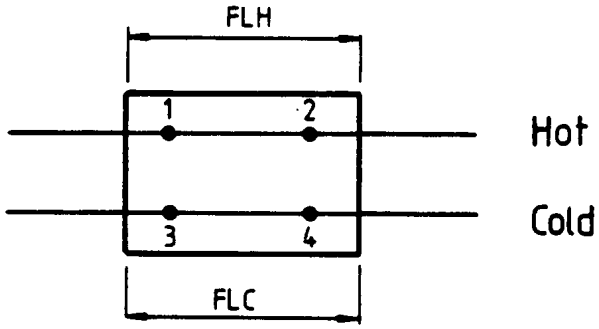


Figure 5-7. Transfer function heat exchanger model‘‘‘

$SUBSTITUTIONS data:-

For both sides:-

MCP	=	r-val;	(thermal capacity of each diffusion node)
EFF	=	r-val;	(heat exchanger effectiveness)
UAVAL	=	r-val;	(heat transfer coefficient)

For the "hot" side:-

FLH	=	r-val;	(total heat exchanger length)
ADH	=	r-val;	(hydraulic diameter)
FSTH	=	z-val;	(fluid state descriptor)
FEH	=	r-val;	(enthalpy)
PH	=	r-val;	(pressure)
TH	=	r-val;	(temperature)
VQH	=	r-val;	(vapour quality)
FFH	=	r-val;	(surface roughness)
FLAH	=	r-val;	(flow area)
FXIH, FYIH, FZIH	=	r-val;	(coordinates of inlet)
FXOH, FYOH, FZOH	=	r-val;	(coordinates of outlet)
TYPH	=	z-val;	(fluid type)
MFLOWH	=	r-val;	(mass flow rate)
ETAH	=	r-val;	(hot fin temperature effectiveness)
GPH	=	r-val;	(flow conductance)
DPHOT	=	r-val;	(pressure drop)

For the "cold" side:-

FLC	=	r-val;	(total heat exchanger length)
ADC	=	r-val;	(diameter)
FSTC	=	z-val;	(fluid state descriptor)
FEC	=	r-val;	(enthalpy)
PC	=	r-val;	(pressure)
TC	=	r-val;	(temperature)
VQC	=	r-val;	(vapour quality)
FFC	=	r-val;	(surface roughness)
FLAC	=	r-val;	(flow area)
FXIC, FYIC, FZIC	=	r-val;	(coordinates of inlet)
FXOC, FYOC, FZOC	=	r-val;	(coordinates of outlet)
TYPC	=	z-val;	(fluid type)
MFLOWC	=	r-val;	(mass flow)
ETAC	=	r-val;	(heat transfer efficiency)
GPC	=	r-val;	(flow conductance)
DPCOLD	=	r-val;	(pressure drop)

In addition,

MODE	=	1	(counter flow heat exchanger)
	=	2	(parallel flow heat exchanger)
	=	3	(cross flow heat exchanger)

There are also three methods of inputting the heat exchanger characteristics:

HXTYP	=	1	(user-defined effectiveness)
	=	2	(effectiveness calculated internally)
	=	3	(effectiveness obtained from array interpolation)

The following should be observed:-

Hot side input node = 1
Hot side exit node = 2
Cold side input node = 3
Cold side exit node = 4

Defaults provided within this heat exchanger are:

FLAC	=	$\pi \cdot ADC^2 / 4$
FLAH	=	$\pi \cdot ADH^2 / 4$
FSTC, FSTH	=	'P&T'
PC, PH	=	0.0
FEC, FEH	=	0.0
TC, TH	=	0.0
VQC, VQH	=	0.0
FXIH, FYIH, FZIH	=	0.0
FXOH, FYOH, FZOH	=	0.0
FXIC, FYIC, FZIC	=	0.0
FXOC, FYOC, FZOC	=	0.0
MODE	=	2
HXTYP	=	2
GPH, GPC	=	1.0D10
EFF	=	0.0
ETAH, ETAC	=	0.0
DPHOT, DPCOLD	=	0.0
MCP	=	0.0
UAVAL	=	0.0
FFH, FFC	=	0.0

The method used calculates the heat transferred from the hot stream to the cold stream and simulates the heat transfer via the addition of heat sources onto the outlet nodes (i.e. a negative FQ on the hot side outlet and a positive FQ on the cold side outlet).

The fluid state for both elements can be defined using pressure and enthalpy ('P&FE'), pressure and temperature ('P&T'), pressure and vapour quality ('P&VQ') and temperature and vapour quality ('T&VQ') by setting the fluid state descriptors FSTC and FSTH to the

appropriate string. No error will be generated if no state variables are defined, however the state must be defined prior to solution (i.e. within the $INITIAL block).

To model the thermal inertia of the heat exchanger within transient analysis GL conductors are generated between the fluid and solid nodes. The overall heat transfer coefficient for HXTYPE= 1 is calculated from the input heat exchanger efficiency. It must be noted that for both the cross flow and the parallel flow arrangement there is a range where the heat transfer coefficient is indeterminate. It is therefore necessary for the user to define the heat transfer coefficient within this region.

An outline of each of the different heat exchanger types is given below, for more information refer to chapter 6 of the ESATAN Engineering Manual.

HXTYP = 1

This option allows the user to input values of the heat exchanger effectiveness and the pressure drop over the heat exchanger. These values can be modified during the solution by user MORTRAN at a higher level model. The effectiveness is assigned to a user constant called EFF and the heat exchanger overall pressure drop to constants called DPHOT and DPCOLD for the hot and cold sides respectively. From the heat exchanger effectiveness the overall heat transfer between the hot and cold sides is evaluated. For transient analysis linear conductors are generated between the fluid and solid nodes to represent the inertia of the system. The overall heat transfer coefficient is calculated as given in the reference via the input effectiveness, see note above.

Redundant data with this heat exchanger type are

- Heat exchanger conductance values GPH and GPC.
- Heat exchanger surface roughness FFC and FFH.

HXTYP = 2

This option forces the model to evaluate the heat exchanger number of transfer units, effectiveness and hence the heat transferred between the hot and cold sides. The pressure drop over the heat exchanger is automatically calculated using the input surface roughness values FFC and FFH and also the conductance values GPH and GPC.

HXTYP = 3

This option calculates the heat exchanger effectiveness via an array of Re versus $St.Pr^{0.6}$ pairs and the pressure loss via an array of Re versus friction factor. The overall heat transfer coefficient between the fluid and solid nodes is automatically calculated from the number of transfer units.

The method used for HXTYP=1 is defined within reference.[1]

5.7 Annular Heat Exchanger

General Concepts:-

This heat exchanger is formed by two concentric tubes, with one fluid flowing in the circular gap and the other in the annular gap. Both parallel and counterflow conditions are modelled by this unit.

The user specifies nodal properties for shell and tube nodes, temperature and capacitance of the tube, and number of nodes in the axial direction. The minimum (default) value of the number of nodes in the axial direction is 2; defining NNODE = 1 shall generate an error.

ESATAN Input:-

$ELEMENT name:-

HXANNULAR Annular heat exchanger.

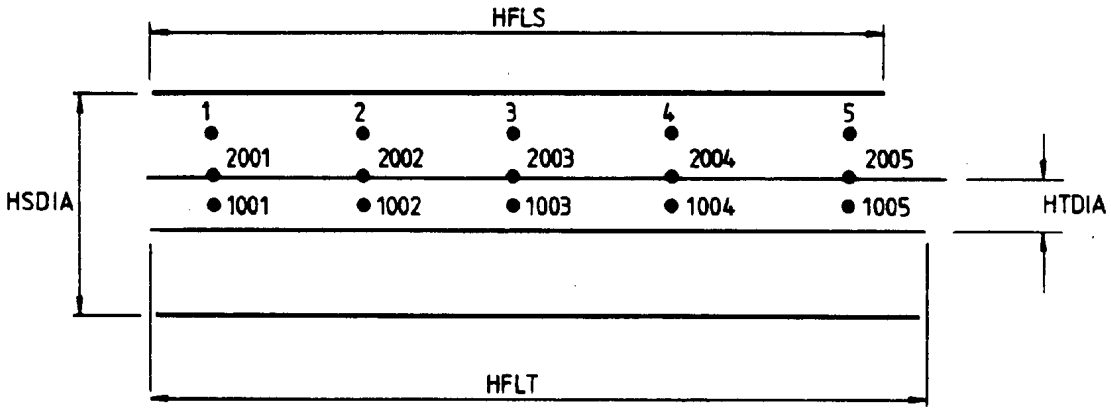


Figure 5-8. Annular heat exchanger model

$SUBSTITUTIONS data:-

For the tube-side fluid nodes:-

HAT	=	r-val;	(heat transfer area)
HTDIA	=	r-val;	(hydraulic diameter)
HFLT	=	r-val;	(length)
HFSTT	=	z-val;	(fluid state descriptor)
HPT	=	r-val;	(pressure)
HFET	=	r-val;	(enthalpy)
HFTT	=	r-val;	(temperature)
HVQT	=	r-val;	(vapour quality)

HFFT	=	r-val;	(roughness)
HFXI, HFYI, HFZI	=	r-val;	(coordinates of the inlet node)
HFXO, HFYO, HFZO	=	r-val;	(coordinates of the outlet node)
HTYPT	=	z-val;	(fluid type)
TKLOSS	=	r-val;	(total pressure loss coefficient)

For the shell-side fluid nodes:-

HAS	=	r-val;	(heat transfer area)
HSDIA	=	r-val;	(diameter)
HFLS	=	r-val;	(length)
HFSTS	=	z-val;	(fluid state descriptor)
HPS	=	r-val;	(pressure)
HFES	=	r-val;	(enthalpy)
HFTS	=	r-val;	(temperature)
HVQS	=	r-val;	(vapour quality)
HFFS	=	r-val;	(roughness)
HTYPS	=	z-val;	(fluid type)
SKLOSS	=	r-val;	(total pressure loss coefficient)

Tube-wall nodes:-

TT	=	r-val;	(temperature)
TC	=	r-val;	(capacitance)

General features:-

NNODES	=	i-val;	(number of nodes in the axial direction; ≥ 2)
MFLOWT	=	r-val;	(mass flow rate on tube side)
MFLOWS	=	r-val;	(mass flow rate on shell side)

The following should be noted:-

1. Tube side input node = 1001
2. Shell side input node = 1
3. Tube side exit node = 1000 + NNODES
4. Shell side exit node = NNODES

Defaults provided within this element are;

NNODES	=	2
HAT	=	$\pi \cdot$ HTDIA $\cdot$ HFLT / NNODES
HAS	=	$\pi \cdot$ HSDIA $\cdot$ HFLS / NNODES
HFSTT	=	’P&T’
HFSTS	=	'P&T’
HPT, HPS	=	0.0
HFET, HFES	=	0.0
HTT, HTS	=	0.0
HVQT, HVQS	=	0.0
HFFT, HFFS	=	0.0
TKLOSS	=	0.0
SKLOSS	=	0.0
TC	=	0.0
HFXI, HFYI, HFZI	=	0.0
HFXO, HFYO, HFZO	=	0.0

This element uses the correlations given in Appendix K for the heat transfer and pressure drop on the tube side. The total flow resistance for both the tube and shell side flows can be defined via the substitution data TKLOSS and SKLOSS respectively. From these, fitting loss conductors are generated with the appropriate conductance.

The fluid state can be defined using pressure and enthalpy ('P&FE'), pressure and temperature ('P&T'), pressure and vapour quality ('P&VQ') and temperature and vapour quality ('T&VQ') by setting the fluid state descriptors HFSTT or HFSTS to the appropriate string. No error will be generated if no state variables are defined, however the state must be defined prior to solution (i.e. within the $INITIAL block).

The shell side uses the Dittus-Boelter correlation[2] based upon the annulus hydraulic diameter.

5.8 Cross-Flow Heat Exchanger

General Concepts:-

These are modelled as tube banks within a containing shell. The tube bundle configuration may be staggered or in-line. The user specifies:-

- Nodal information for tube and shell
- Number of tubes.

ESATAN Input:-

$ELEMENT name:-

HXXFLOW Cross-flow heat exchanger.

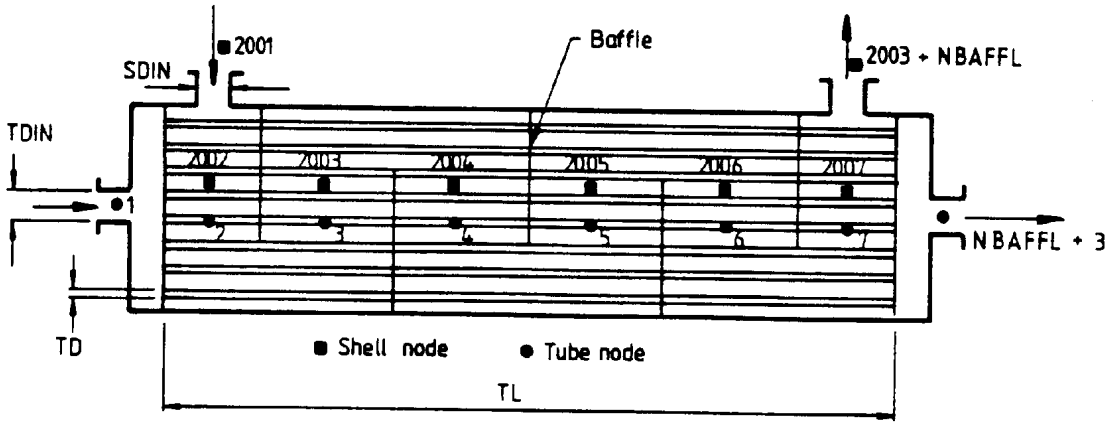


Figure 5-9. Cross-flow heat exchanger model

The flow in the heat exchanger is modelled by one node per baffle width on the shell side, and one corresponding node on the tube side.

$SUBSTITUTIONS data:-

The following data are given representing one tube:-

TD	=	r-val;	(diameter)
TL	=	r-val;	(length)
TA	=	r-val;	(heat transfer area)
TFST	=	z-val;	(fluid state descriptor)
TP	=	r-val;	(pressure)
TE	=	r-val;	(enthalpy)
TT	=	r-val;	(temperature)

TVQ	=	r-val;	(vapour quality)
TFF	=	r-val;	(roughness)
TTYP	=	z-val;	(fluid type)

For the shell-side fluid nodes:-

SA	=	r-val;	(heat transfer area)
SFST	=	z-val;	(fluid state descriptor)
SP	=	r-val;	(pressure)
SE	=	r-val;	(enthalpy)
ST	=	r-val;	(temperature)
SVQ	=	r-val;	(vapour quality)
STYP	=	z-val;	(fluid type)

General features:-

SDIN	=	r-val;	(diameter of shellside inlet/outlet nodes)
SFXI, SFYI, SFZI	=	r-val;	(coordinates of shellside inlet node)
SFXO, SFYO, SFZO	=	r-val;	(coordinates of shellside outlet node)
TDIN	=	r-val;	(diameter of tubeside inlet/outlet nodes)
TFXI, TFYI, TFZI	=	r-val;	(coordinates of tubeside inlet node)
TFXO, TFYO, TFZO	=	r-val;	(coordinates of tubeside outlet node)
DSHELL	=	r-val;	(shell diameter)
NBAFFL	=	i-val;	(number of tube plate baffles)
MFLOWT	=	r-val;	(tube side flow rate)
MFLOWS	=	r-val;	(shell side flow rate)
NTUBES	=	i-val;	(number of tubes)
LAYOUT	=	i-val;	(tube layout)
	=	90°	(in-line square)
	=	45°	(rotated square)
	=	30°	(in-line triangular)
	=	60°	(rotated triangular))
TW	=	r-val;	(tube wall temperature)
CW	=	r-val;	(tube wall node capacitance)

Defaults provided within this element are:-

TA, SA	=	$\pi \cdot TD \cdot TL \cdot NTUBES$
TFST, SFST	=	'P&T'
TP, SP	=	0.0

TFE, SFE = 0.0
TT, ST = 0.0
TVQ, SVQ = 0.0
NBAFFL = 1
LAYOUT = 90°
CW = 0.0 (arithmetic nodes)
TFXI, TFYI, TFZI = 0.0
TFXO, TFYO, TFZO = 0.0
SFXI, SFYI, SFZI = 0.0
SFXO, SFYO, SFZO = 0.0

The following should be noted:-

1. Tube side input node = 1
2. Tube side exit node = NBAFFL + 3
3. Shell side inlet node = 2001
4. Shell side exit node = 2003 + NBAFFL

The fluid state can be defined using pressure and enthalpy ('P&FE'), pressure and temperature ('P&T'), pressure and vapour quality ('P&VQ') and temperature and vapour quality ('T&VQ') by setting the fluid state descriptors TFST and SFST to the appropriate string. No error will be generated if no state variables are defined, however the state must be defined prior to solution (i.e. within the $INITIAL block). Heat transfer and pressure drop on the tube side are calculated by the method given in Appendix K.

See references[3][4] for tube layout. Shell-side heat transfer and pressure drop are calculated by the methods described in references[5][6].

5.9 Condensing Heat Exchanger

General Concepts:-

A simple counterflow condensing heat exchanger element is provided for use within air/water-vapour loops, i.e. with fluid type AIRW. Water vapour is condensed out by cooling the working fluid to its dew-point (if possible) and removing the resulting water droplets; the air/vapour mixture emerges from the outlet at 100% humidity.

Steady-state conditions are assumed, heat storage and transient effects not being accounted for.

ESATAN Input:-

$ELEMENT name:-

CHX	Condensing heat exchanger.

$SUBSTITUTIONS data:-

For both sides:-

DAMPQ	=	r-val;	(heat input damping factor)
DAMPW	=	r-val;	(condensation damping factor)
UA	=	r-val;	(heat transfer coefficient)

For the hot side:-

XTH	=	r-val;	(temperature)
XPH	=	r-val;	(pressure)
XFLH	=	r-val;	(total heat exchanger length)
XFDH	=	r-val;	(diameter)
XFLAH	=	r-val;	(flow area)
XFFH	=	r-val;	(surface roughness)
XFXIH, XFYIH, XFZIH	=	r-val;	(coordinates of inlet node)
XFXOH, XFYOH, XFZOH	=	r-val;	(coordinates of outlet node)
MFLOWH	=	r-val;	(flow rate)
XGPH	=	r-val;	(fitting loss conductance)

For the cold side:-

XFSTC	=	z-val;	(fluid state descriptor)
XTC	=	r-val;	(temperature)
XPC	=	r-val;	(pressure)
XFEC	=	r-val;	(enthalpy)
XVQC	=	r-val;	(vapour quality)

XFLC	=	r-val;	(total heat exchanger length)
XFDC	=	r-val;	(diameter)
XFLAC	=	r-val;	(flow area)
XFFC	=	r-val;	(surface roughness)
XFXIC, XFYIC, XFZIC	=	r-val;	(coordinates of inlet node)
XFXOC, XFYOC, XFZOC	=	r-val;	(coordinates of outlet node)
XTYPC	=	z-val;	(coolant fluid type)
MFLOWC	=	r-val;	(mass flow)
XGPC	=	r-val;	(fitting loss conductance)

Defaults provided within this element are:-

XFLAC	=	$\pi \cdot XFDC^2 / 4$
XFLAH	=	$\pi \cdot XFDH^2 / 4$
XFSTC	=	'P&T’
XPC	=	0.0
XTC	=	0.0
XFEC	=	0.0
XVQC	=	0.0
XFFC, XFFH	=	0.0
DAMPQ	=	1.0
DAMPW	=	1.0
XGPC, XGPH	=	1.0D10
XFXIH, XFYIH, XFZIH	=	0.0
XFXOH, XFYOH, XFZOH	=	0.0
XFXIC, XFYIC, XFZIC	=	0.0
XFXOC, XFYOC, XFZOC	=	0.0

The following should be observed:-

Hot side input node	= 1
Hot side exit node	= 2
Cold side input node	= 3
Cold side exit node	= 4

The fluid state can be defined using pressure and enthalpy ('P&FE'), pressure and temperature ('P&T'), pressure and vapour quality ('P&VQ') and temperature and vapour quality ('T&VQ') by setting the fluid state descriptor XFSTC to the appropriate string. No error will be generated if no state variables are defined, however the state must be defined prior to solution (i.e. within the $INITIAL block).

An iterative method is used to calculate the heat transferred from the hot stream to the cold stream and the rate of condensation of water vapour. The hot-side fluid is assumed to be of type AIRW; the coolant type must be specified by the user. The rate of condensation is stored in the user constant WCOND, and the condensate is assumed to be at the same temperature as the main outlet. Frictional pressure drop is calculated in this model by the method of Appendix K; other irreversible losses may be accounted for by setting the fitting-loss conductances provided on both sides of the heat exchanger.

Two damping factors, user constants DAMPQ and DAMPW, are provided to aid in attaining convergence between the CHX and the main solution. The control constants NLOOPH and NLOOPT may also be useful for this, by limiting the number of hydraulic and thermal iterations, respectively, executed by the main solution before the CHX outputs are recalculated.

5.10 Heat Pipe-Bank Condensing Radiator

General Concepts:-

This consists of a tube connected to a heat pipe panel. The heat pipes are not part of the fluid loop as such; they must be modelled by conventional submodels or by VCHP submodels. The heat pipe-bank element gives the user a method of describing several fluid flow paths, to which heat pipes may be attached. The user describes data for one such path, and the number of nodes along each such path. The element generated contains a pair of nodes, one fluid and one thermal, at each point along the path. Any external heat pipes must be connected, by inter-model thermal conductances, to the thermal nodes generated. The user specifies:-

- Tube characteristics
- Number of parallel tubes

ESATAN Input:-

$ELEMENT name:-

HPBANK Heat pipe bank

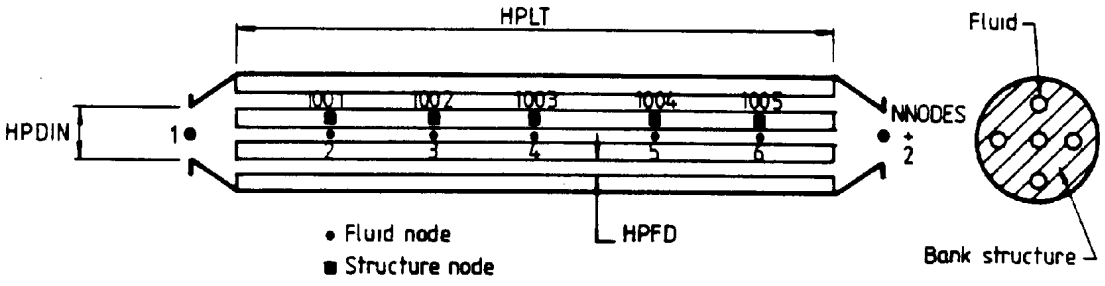


Figure 5-10. Heat pipe bank model

$SUBSTITUTION data:-

For each node in one passage:-

HPA	=	r-val;	(heat transfer area)
HPFD	=	r-val;	(diameter)
HPLT	=	r-val;	(length)
HPFST	=	z-val;	(fluid state descriptor)
HPP	=	r-val;	(pressure)
HPE	=	r-val;	(enthalpy)
HPT	=	r-val;	(temperature)
HPVQ	=	r-val;	(vapour quality)

HPFF	=	r-val;	(roughness)
KLOSS	=	r-val;	(total flow resistance)

General features:-

HPDIN	=	r-val;	(diameter of inlet/outlet nodes)
HPFXI, HPFYI, HPFZI	=	r-val	(coordinates of inlet node)
HPFXO, HPFYO, HPFZO	=	r-val	(coordinates of outlet node)
TT	=	r-val;	(temperature of bank structure thermal
TC	=	r-val;	(capacitance of bank structure thermal nodes)
MFLOWT	=	r-val;	(mass flow rate)
NNODES	=	i-val;	(number of node in the axial direction)
NTUBES	=	i-val;	(number of parallel tubes)

Defaults provided for this element are:-

HPA	=	NTUBES · π · HPFD · HPLT
HPFST	=	'P&T’
HPP	=	0.0
HPFE	=	0.0
HPT	=	0.0
HPVQ	=	0.0
NNODES	=	1
TC	=	0.0 (arithmetic nodes)
HPFF	=	0.0
KLOSS	=	0.0
HPFXI, HPFYI, HPFZI	=	0.0
HPFXO, HPFYO, HPFZO	=	0.0

The fluid state can be defined using pressure and enthalpy ('P&FE'), pressure and temperature ('P&T'), pressure and vapour quality ('P&VQ') and temperature and vapour quality ('T&VQ') by setting the fluid state descriptor HPFST to the appropriate string. No error will be generated if no state variables are defined, however the state must be defined prior to solution (i.e. within the $INITIAL block).

The total flow resistance of the coldplate can be defined via the substitution data KLOSS. From this, fitting loss conductors are generated with the appropriate conductance.

The following should be noted:-

1. Inlet and exit fluid nodes are 1 and NNODES + 2 respectively.

2. One thermal node exists at each axial tube location, numbered 1001 to (1001 + NNODES). Connections to the 'outside' should be made by conductances to these nodes.

Heat transfer and pressure drop are calculated by the method given in Appendix K.

5.11 Panel Condensing Radiator

General Concepts:-

This is a two-dimensional panel, with several passes of fluid tubing embedded. Radiation conductances are generated from the two panel surfaces. It is assumed that these conductances connect to a boundary node on each side. The user specifies:-

1. Panel dimensions and thermal properties.
2. Fluid tube characteristics.
3. External emissivities and view factors from each side of the panel.

ESATAN Input:-

$ELEMENT names:-

PANEL	Panel radiator (multiple sections)
PANEL1	Panel radiator (single section)

The panel radiator is almost identical to the coldplate element described earlier. In order to build up a labyrinth of fluid tubing in a sheet of solid material, input identical to that for the coldplate is used. In addition, radiative heat transfer conductances are generated from both sides of the panel, connecting each side of the panel to a boundary node.

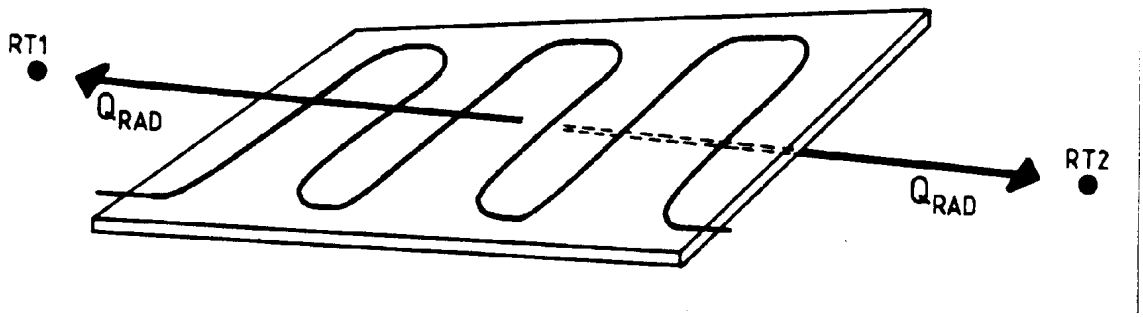


Figure 5-11. Panel radiator model

$SUBSTITUTIONS data:-

LENX	=	r-val;	(length of fluid tube in x - direction)
LENY	=	r-val;	(length of fluid tube in y - direction)
DZ	=	r-val;	(plate thickness)
NX	=	i-val;	(number of nodes in x direction)
NY	=	i-val;	(number of nodes in y direction)

RTP	=	r-val;	(panel temperature)
RK	=	r-val;	(panel conductivity)
RRHO	=	r-val;	(panel density)
RCP	=	r-val;	(panel specific heat)
RFD	=	r-val;	(tube diameter)
RFST	=	z-val;	(fluid state descriptor)
RP	=	r-val;	(tube pressure)
RFE	=	r-val;	(tube enthalpy)
RT	=	r-val;	(tube temperature)
RVQ	=	r-val;	(vapour quality)
RFF	=	r-val;	(tube roughness)
KLOSS	=	r-val;	(total pressure loss coefficient)
RFX1, RFY1, RFZ1	=	r-val;	(coordinate of inlet bottom corner of coldplate structure)
RFX2, RFY2, RFZ2	=	r-val;	(coordinate of outlet bottom corner of coldplate structure)
RFX3, RFY3, RFZ3	=	r-val;	(coordinate of inlet top corner of coldplate structure)
NU	=	i-val;	(number of U-tubes in series)
MFLOW	=	r-val;	(mass flow rate)
REMM1, REMM2	=	r-val;	(emissivities of the two panel sides)
RVFAC1, RVFAC2	=	r-val;	(view factors of the two panel sides)
RT1, RT2	=	r-val;	(boundary temperatures)

Defaults provided for this element are:-

RFST	=	’P&T’
RP	=	0.0
RFE	=	0.0
RT	=	0.0
RVQ	=	0.0
RFF	=	0.0
NX, NY	=	4
KLOSS	=	0.0
RRHO	=	0.0
RCP	=	0.0
REMM1	=	1.0
REMM2	=	1.0
RVFAC1	=	1.0
RVFAC2	=	1.0
RFX1, RFY1, RFZ1	=	0.0
RFX2, RFY2, RFZ2	=	0.0
RFX3, RFY3, RFZ3	=	0.0

The following should be noted:-

1. NX must be a multiple of 4.
2. The node numbers are generated as shown.
3. $NY \geq NX/2 + 1$.
4. $NX*NY + 2NY \leq 999$.

   For connection to the adjacent fluid tubing,
   input node number is:-

   $1000 + NX * NY + 1$

   output node is:-

   $NU * 1000 + NX * NY + 2NY$
5. $NU \geq 2$

If only one U-section is required, element PANEL1 should be used. This does not require a variable NU.

Fluid type must be specified on the $MODEL card. The fluid state can be defined using pressure and enthalpy ('P&FE'), pressure and temperature ('P&T'), pressure and vapour quality ('P&VQ') and temperature and vapour quality ('T&VQ') by setting the fluid state descriptor RFST to the appropriate string. No error will be generated if no state variables are defined, however the state must be defined prior to solution (i.e. within the $INITIAL block).

The total flow resistance of the coldplate can be defined via the substitution data KLOSS. From this, fitting loss conductors are generated with the appropriate conductance. Heat transfer and pressure drop are calculated by the method given in Appendix K.

Note that if the user wishes to specify a more complex set of radiative links, it may be easier to use the COLDPLATE element to give the physical structure of the panel radiator, and then explicitly define external nodes and radiative links to the thermal nodes making up the radiator body.

5.12 Ideal Accumulators

General Concepts:-

Ideal accumulators are modelled by boundary nodes, and as such may be defined as either prescribed pressure only, or prescribed pressure and temperature, essentially assuming an infinite volume. The user therefore has the choice of modelling an ideal accumulator explicitly as a boundary node, or using the element feature available. In the latter case, the accumulator becomes a named submodel.

Through-flow accumulators are modelled by creating the accumulator model directly in a flow line, i.e. the accumulator will have two connections, to represent inlet and outlet flow. A 'dead-end' accumulator is modelled by connecting the accumulator model to the side branch of a tee-piece, and in this case will have only one connection. Control of ideal accumulators is achieved by modifying the nodal parameters through operations-block statements.

ESATAN Input:-

$ELEMENT names:-

ACCP	Constant-pressure accumulator
ACCPT	Constant-pressure and -temperature accumulator

$SUBSTITUTIONS data:-

AA	=	r-val;	(heat transfer area)
AFD	=	r-val;	(diameter)
AFLA	=	r-val;	(flow area)
AVOL	=	r-val;	(volume)
AFL	=	r-val;	(length)
AFST	=	z-val;	(fluid state descriptor)
AP	=	r-val;	(pressure)
AT	=	r-val;	(temperature)
AFE	=	r-val;	(enthalpy)
AVQ	=	r-val;	(vapour quality)
AFF	=	r-val;	(roughness)
AFX, AFY, AFZ	=	r-val;	(coordinates)

Defaults provided for the accumulator elements are:-

AA	=	$\pi \cdot AFD \cdot AFL$
AFLA	=	$\pi \cdot AFD^2 / 4$
AFL	=	AVOL / AFLA
AVOL	=	0.0 ($\Rightarrow$ AFL must be defined)
AFST	=	'P&T'
AP	=	0.0

AFE	=	0.0
AT	=	0.0
AVQ	=	0.0
AFX	=	0.0
AFY	=	0.0
AFZ	=	0.0

Fluid type must be specified on the $MODEL card.

The geometry of the accumulators is defined using the flow area (AFLA) and either the length (AFL) or the volume (AVOL). If the volume is defined the nodal length is derived automatically. If the flow area is not defined it is assumed to be of circular cross section, the diameter taken equal to the hydraulic diameter (AFD).

The fluid state can be defined using pressure and enthalpy ('P&FE'), pressure and temperature ('P&T'), pressure and vapour quality ('P&VQ') and temperature and vapour quality ('T&VQ') by setting the fluid state descriptors HFSTT or HFSTS to the appropriate string. No error will be generated if no state variables are defined, however the state must be defined prior to solution (i.e. within the $INITIAL block).

5.13 Non-Ideal Accumulators

General Concepts:-

Active and passive accumulators consist of a finite but variable volume, a cylinder/piston arrangement being assumed. In the active type the volume is under direct user control via the speed of the piston, whilst in a passive accumulator the volume varies according to a compliance provided by a quantity of perfect gas behind the piston. If the gas and the working fluid are assumed to be in thermal equilibrium then the compliance is independent of the species of gas; however, for adiabatic processes, such as in fast transients, the compliance depends on the ratio of specific heats for the gas, and is somewhat reduced.

ESATAN Input:-

$ELEMENT names:-

ACTA	active accumulator
PASSA	passive accumulator

$SUBSTITUTIONS data:-

ACTA

AA	=	r-val;	(heat transfer area)
AFD	=	r-val;	(diameter)
AFLA	=	r-val;	(flow area)
AFST	=	z-val;	(fluid state descriptor)
AP	=	r-val;	(pressure)
AT	=	r-val;	(temperature)
AFE	=	r-val;	(enthalpy)
AVQ	=	r-val;	(vapour quality)
AFX, AFY, AFZ	=	r-val;	(coordinates)
PPOS	=	r-val;	(piston position)
SPEED	=	r-val;	(piston speed)
VOLACC	=	r-val;	(total accumulator volume)

PASSA

AA	=	r-val;	(heat transfer area)
AFD	=	r-val;	(diameter)
AFLA	=	r-val;	(flow area)
AFST	=	z-val;	(fluid state descriptor)
AP	=	r-val;	(pressure)
AT	=	r-val;	(temperature)
AFE	=	r-val;	(enthalpy)
AFX, AFY, AFZ	=	r-val;	(coordinates)
PPOS	=	r-val;	(piston position)
GAMMA	=	r-val;	(ratio of specific heats of gas)
PATYPE	=	z-val;	(accumulator type: 'THEQM' or 'ADIAB')
VOLACC	=	r-val;	(total accumulator volume)
AVQ	=	r-val;	(vapour quality)

Defaults provided for both elements are:-

AA	=	$4 \cdot PPOS \cdot VOLACC / AFD$
AFLA	=	$\pi \cdot AFD^2 / 4$
AFST	=	'P&T'
AP	=	0.0
AT	=	0.0
AFE	=	0.0
AVQ	=	0.0
AFX	=	0.0
AFY	=	0.0
AFZ	=	0.0
PPOS	=	0.5
SPEED	=	0.0 (ACTA only)

Piston position PPOS varies from 0.0, with the accumulator emptied of working fluid, to 1.0 with fluid occupying the whole volume; its value is available to the user via a user constant of the same name. SPEED, the piston speed, is positive for increasing working-fluid volume, negative for decreasing; it may be updated in operations blocks via a user constant also called SPEED. PATYPE denotes whether thermal equilibrium ('THEQM') or adiabatic ('ADIAB') conditions are to be assumed in the passive accumulator. In the former case the value of GAMMA is irrelevant. GAMMA may be updated as a user constant in operations blocks. VOLACC is the total volume of the accumulator, not just that occupied by the working fluid.

Fluid type must be specified on the $MODEL card. The fluid state for both elements can be defined using pressure and enthalpy ('P&FE'), pressure and temperature ('P&T'), pressure and vapour quality ('P&VQ') and temperature and vapour quality ('T&VQ') by setting the fluid state descriptor AFST to the appropriate string. No error will be generated if no state

variables are defined, however the state must be defined prior to solution (i.e. within the $INITIAL block).

5.14 Capillary Evaporator

General Concepts:-

A capillary evaporator provides mass flow in a two-phase loop, driven by heat input. The outer casing contains a wick made of porous material such as polyethylene or sintered nickel. Working fluid enters the inner core as liquid and soaks through the wick to its outer surface, where it is evaporated due to the conduction of heat from the outside of the casing. When evaporation starts the evaporator is primed and starts pumping. A meniscus forms at the liquid/vapour interface in each pore of the wick, capable of supporting the pressure difference between the liquid and the vapour, the latter emerging at a higher pressure.

When the evaporator is primed, a mass flow rate is induced according to the heat flux,

$$W = \frac{Q}{h_v - h_{in}}$$

and the fluid leaving the wick is saturated vapour. The heat flux here is calculated from the casing temperature, the fluid-saturation temperature, and a user-defined conductance value, thus:

$$Q = G_{primed}(T_{casing} - T_s)$$

The wick imposes a pressure drop given by Darcy's law for flow through a porous medium.

In practice a minimum heat flux, Q_{start}, is needed before the evaporator will prime; similarly, if the heat flux falls below a certain value, Q_{stop}, the evaporator will deprime (generally, $Q_{start} > Q_{stop}$). A minimum subcooling, ΔT_{sub}, of the liquid at the inlet is usually required to ensure that no vapour reaches the inner surface of the wick, as this would cause depriming. On the other hand, the evaporator may be tolerant of a small amount of vapour entering. These parameters are all empirical and will vary from one design of evaporator to another.

The maximum pressure rise that can be supported by the meniscus is known as the capillary limit, and is dependent on the fluid properties, the wick material, and the radius of the pores in the wick. If this is exceeded vapour will blow back through the wick: the wick will dry out and the evaporator will deprime. This must be avoided or the wick will risk degradation.

When the evaporator is deprimed, the pumping effect stops and the evaporator acts merely as a resistance to flow; it behaves actually as a capillary filter, blocking the flow of vapour.

Typically a capillary evaporator will be of cylindrical or flat plate geometry.

ESATAN Input:-

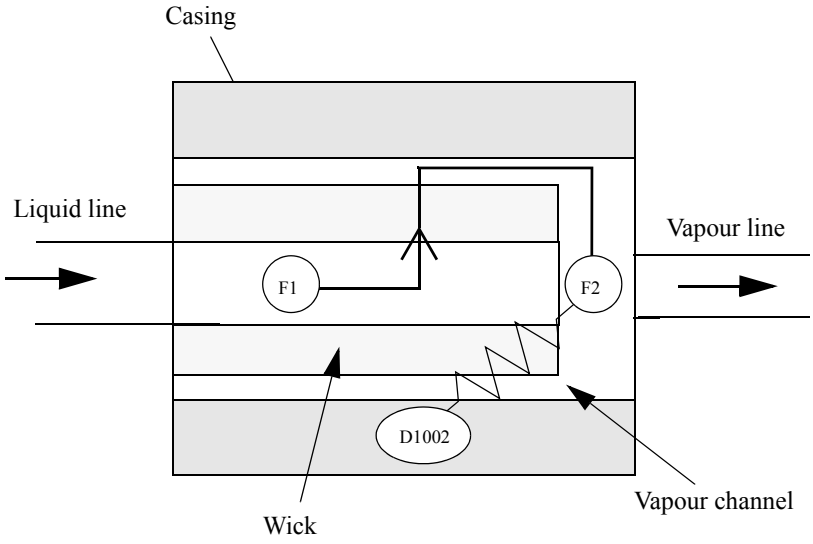


Figure 5-12. Capillary evaporator element

$ELEMENT name:-

EVAPORATOR_CAP Capillary evaporator

$SUBSTITUTIONS data:-

C_WALL	=	r-val;	(wall capacitance)
DELAY_CAPLIM	=	r-val;	(time delay before re-priming)
DIAMI_WICK	=	r-val;	(wick inlet diameter)
DIAMO_WICK	=	r-val;	(wick outlet diameter)
DT_SUB	=	r-val;	(required subcooling)
FLA_VAP	=	r-val;	(flow area of each vapour channel)
G_DEPRIMED	=	r-val;	(wall-fluid conductance: deprimed mode)
G_PRIMED	=	r-val;	(wall-fluid conductance: primed mode)
LENGTH	=	r-val;	(evaporator length)
MFLOW	=	r-val;	(initial mass flow rate)
MODE	=	z-val;	(initial operating mode: 'PRIMED' or 'DEPRIMED')
NVAPC	=	i-val;	(number of vapour channels)
PERM_WICK	=	r-val;	(wick permeability)
PRESS	=	r-val;	(initial pressure of the fluid)
Q_LOAD	=	r-val;	(heat load on the casing)
Q_START	=	r-val;	(heat flux to prime the evaporator)
Q_STOP	=	r-val;	(heat flux to deprime the evaporator)

RAD_PORE	=	r-val;	(effective pore radius of the wick)
RUG_LIQ	=	r-val;	(surface roughness of liquid core)
RUG_VAP	=	r-val;	(surface roughness of vapour channels)
TEMP	=	r-val;	(initial temperature of the fluid)
THETA_WET	=	r-val;	(wetting angle of the fluid in the wick)
T_WALL	=	r-val;	(initial wall temperature)
VQLIMH	=	r-val;	(vapour quality high limit allowed into the wick before depriming)
VQLIML	=	r-val;	(vapour quality low limit allowed into the wick before repriming)
WIDTH_VAP	=	r-val;	(width of each vapour channel)
X_COORD Y_COORD Z_COORD	=	r-val;	(x,y,z-coordinates)

Defaults provided for this element are:-

C_WALL	=	0.0
DELAY_CAPLIM	=	0.0
DT_SUB	=	0.0
FLA_VAP	=	$WIDTH\_VAP^2$
G_DEPRIMED	=	G_PRIMED
MFLOW	=	0.0
MODE	=	'DEPRIMED'
PERM_WICK	=	1.0E10
Q_LOAD	=	0.0
Q_START	=	0.0
Q_STOP	=	0.0
RAD_PORE	=	0.0
RUG_LIQ	=	0.0
RUG_VAP	=	0.0
THETA_WET	=	0.0
T_WALL	=	TEMP
VQLIMH	=	0.0
VQLIML	=	0.0
X_COORD Y_COORD Z_COORD	=	0.0

Input user constants:-

G_primed	(wall-fluid conductance: primed mode)
G_deprimed	(wall-fluid conductance: deprimed mode)
Q_load	(heat load on the wall)

Q_start	(heat flux to prime the evaporator)
Q_stop	(heat flux to deprime the evaporator)
Dt_sub	(subcooling difference temperature)

Input/output user constants:-

Mode	(operating mode; 'PRIMED' or 'DEPRIMED')

Output user constants:-

Dp_caplim	(capillary limit)
Dp_meniscus	(pressure rise over liquid/vapour interface)
Dp_wick	(Darcy pressure loss in wick)
Dryflg	(dry out flag: 1 if a dry-out has occurred, 0 if not)
Q_primed	(heat flux transferred to the fluid when primed)
Timdry	(time at which dry-out occurred)
Ts	(saturation temperature for the fluid at outlet)

Connection points:-

F1	-	Inlet (liquid core)
F2	-	Outlet (vapour channels)
D1002	-	Casing

The capillary evaporator element consists of two F-nodes connected by a mass flow link (5-12). The inlet node represents the liquid core and the outlet node the vapour channels. The outer casing is modelled by a D-node which is connected to the fluid outlet using a GL conductor. (There is also a dummy boundary node, not shown in the diagram, to ensure correct thermal balance of the casing when the evaporator is primed.) The geometry is cylindrical.

Pressure losses in the liquid core and in the vapour channels are computed by intrinsic friction correlations. The vapour channels are assumed to be of square section by default.

The user must define a conductance between the casing and the fluid by setting the substitution data G_PRIMED and G_DEPRIMED. The conductance when primed is in general substantially greater than when deprimed; however, the element defaults the value for G_DEPRIMED to the same as G_PRIMED.

The default values for the wick are such that the wick is 'ideal', i.e. the Darcy pressure loss is zero and the capillary limit is infinite.

Certain parameters can be varied during solution by addressing the appropriate user constant. In particular, the heat load applied to the casing can be varied during solution by setting Q_load from the main model as required, for instance by interpolation on time-dependent values.

Mode transitions (PRIMED -> DEPRIMED, DEPRIMED -> PRIMED) are carried out within the $VARIABLES2 block. This means that for a steady-state simulation (solver

FGENSS), the mode will stay fixed during the solution. At the end, the conditions will be tested and if the evaporator cannot sustain the specified mode, a warning message will be produced. The default mode being 'DEPRIMED', the user should set the mode to 'PRIMED' before calling the steady-state solver.

In a transient simulation (solver FGENFI), the element will determine the mode at the end of each time step according to the current conditions. The user can check if the evaporator is primed or not by examining the user constant Mode.

If the evaporator deprimes because the capillary limit is exceeded, it is prevented from re-priming for a length of time given by the substitution data DELAY_CAPLIM. After this delay, if the heat flux is not reduced the capillary limit will again be reached, the evaporator will again deprime, and so on. The user should build logic into their main model to handle this situation, which can be detected via the user constants Dp_meniscus, Dp_caplim and Dryflg. The time step at which it has occurred is recorded by Timdry.

By default, if there is some vapour at the inlet, the evaporator will deprime. However, the user is able to allow an amount of a vapour at the inlet without depriming, by setting the substitution data VQLIMH as required. Up to this value, the evaporator will remain primed (assuming DT_SUB = 0.0). Conversely, the evaporator is unable to reprime until the inlet vapour quality has fallen to VQLIML.

$ELEMENT name:-

EVAPUMP

This element has been superseded by EVAPORATOR_CAP and should be considered obsolete.

5.15 Capillary Filter

General Concepts:-

A capillary filter, or isolator, is used to block small amounts of vapour in the fluid. Subcooled liquid is allowed to pass through the filter, but if the fluid is two-phase then vapour bubbles are trapped and only the liquid component passes. The filter contains a capillary wick which results in a pressure loss according to Darcy's law for flow through a porous medium. The more vapour there is at the wick, the smaller the available flow area and hence the greater the resistance.

There is a maximum pressure difference that can be supported by the wick, or rather by the menisci which form in the pores. Known as the capillary limit, if this is exceeded the filter deprimes and will no longer block the passage of vapour until the wick is rewetted.

There are two common geometries: circular, where the fluid flows axially through the wick, and annular, where the flow is radial.

The filter element will block vapour only for forward flow; if backflow occurs in the filter and vapour is present, a warning message is given.

ESATAN Input:-

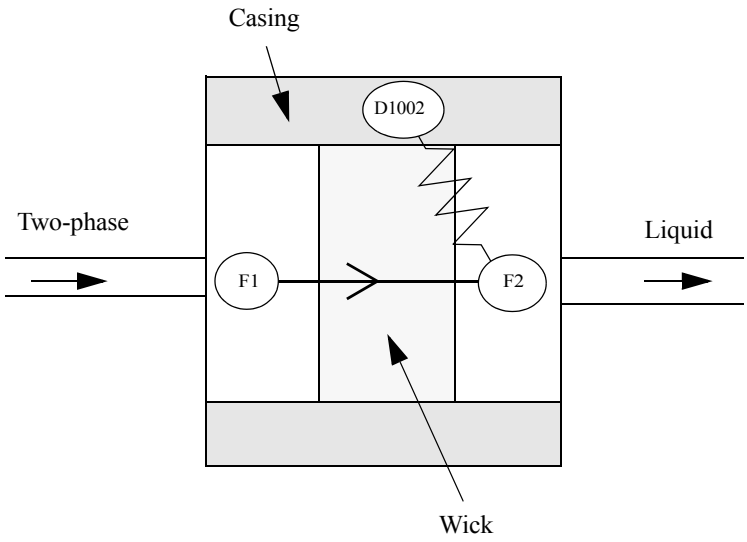


Figure 5-13. Capillary filter (circular design)

$ELEMENT name:-

FILTER_CAP Capillary filter

$SUBSTITUTIONS data:-

C_WALL	=	r-val;	(wall capacitance)
DIAM	=	r-val;	(circular design: flow diameter; annular design: outer flow diameter)
DIAMI_WICK	=	r-val;	(wick inner diameter: annular design)
DIAMO_WICK	=	r-val;	(wick outler diameter: annular design)
G_WF	=	r-val;	(wall-fluid conductance)
LENGTH	=	r-val;	(filter length)
LENGTH_WICK	=	r-val;	(wick length)
MFLOW	=	r-val;	(initial mass flow rate)
PERM_WICK	=	r-val;	(wick permeability)
PRESS	=	r-val;	(initial pressure of the fluid)
RAD_PORE	=	r-val;	(effective pore radius of the wick)
RUG	=	r-val;	(surface roughness)
TEMP	=	r-val;	(initial fluid temperature)
THETA_WET	=	r-val;	(wetting angle in the wick)
TYPE_FILT	=	z-val;	(filter type: 'CIRC' or 'ANN')
T_WALL	=	r-val;	(initial wall temperature)
VQ_WET	=	r-val;	(maximal vapour quality to re-prime)
X_COORD Y_COORD Z_COORD	=	r-val;	(x,y,z-coordinates)

Defaults provided for this element are:-

C_WALL	=	0.0
DIAMI_WICK	=	0.0
DIAMO_WICK	=	0.0
G_WF	=	0.0
LENGTH_WICK	=	0.0
MFLOW	=	0.0
PERM_WICK	=	1.0E10
RAD_PORE	=	0.0
RUG	=	0.0
THETA_WET	=	0.0
TYPE_FILT	=	'CIRC'
T_WALL	=	TEMP
VQ_WET	=	1.0
X_COORD Y_COORD Z_COORD	=	0.0

Input user constants:-

Mode	(operating mode: ’PRIMED’ or ’DEPRIMED’)

Output user constants:-

Dp_caplim	(capillary limit)
Dp_wick	(Darcy pressure loss)
Mode	(operating mode: ’PRIMED’ or ’DEPRIMED’)

Connection points:-

F1	-	Inlet (two-phase fluid)
F2	-	Outlet (pure liquid)
D1002	-	Casing

The capillary filter element consists of two F-nodes connected by a mass flow link. The outer casing is modelled by a D-node which is connected to the outlet node by a GL conductor.

The element can model either a circular type or an annular one (5-14 and 5-15), specified by setting TYPE_FILT as either ’CIRC’ or ’ANN’, respectively.

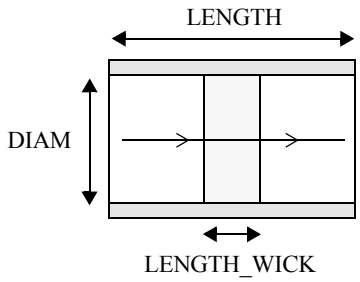


Figure 5-14. Circular filter

The user has to define the diameter (DIAM) and the length (LENGTH) of the filter, and, for the annular design, the wick inner diameter (DIAMI_WICK). If the wick length (LENGTH_WICK) for the circular type, or the wick outer diameter (DIAMO_WICK) for the annular, is not set, the filter will be ideal, i.e. no Darcy pressure loss will be computed but phase separation will still occur. Furthermore, the default pore radius is such that the capillary limit is effectively infinite.

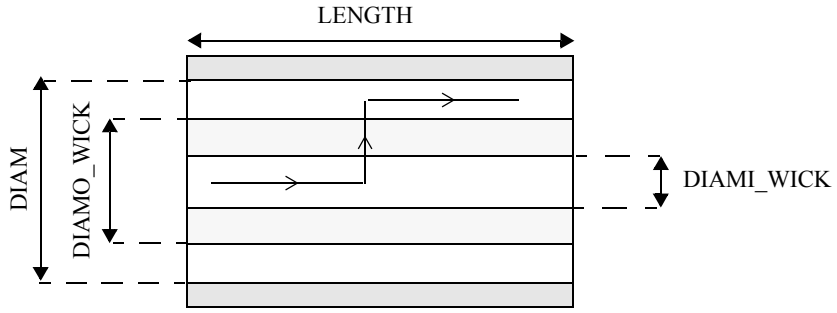


Figure 5-15. Annular filter

The user can define a thermal conductance between the casing and the fluid by setting the substitution data G_WF.

Mode transitions (PRIMED -> DEPRIMED and DEPRIMED -> PRIMED) are carried out within the $VARIABLES2 block. This means that for a steady-state simulation (solver FGENSS), the mode will stay fixed during the solution. At the end, the conditions will be tested and if the filter cannot sustain the specified mode, a warning message will be produced.

In a transient simulation (solver FGENFI), the element will determine the mode at the end of each time step according to the current conditions. The user can check if the filter is primed or not by examining the user constant Mode. Further information is provided by the user constants Dp_meniscus and Dp_caplim.

Once the filter has deprimed, then if the vapour quality falls strictly lower than 1.0 at the inlet, the filter is able to re-prime. The user can reduce this limit by setting the substitution data VQ_WET as required.

$ELEMENT name:-

CAPFILTER

This element has been superseded by FILTER_CAP and should be considered obsolete.

5.16 Two-phase Reservoir

General Concepts:-

The two-phase reservoir is basically a tank of two-phase fluid with a single inlet/outlet; fluid of any phase may enter, but only liquid may exit because a capillary wick is present at the inlet/outlet.

The pressure inside the reservoir is varied by means of heating- and cooling-elements. As heat is applied, the pressure rises and liquid is forced out into the main loop.

The volume of two-phase fluid is assumed to be isothermal, homogeneous and in quasi-equilibrium at all times. Any fluid entering the reservoir undergoes instantaneous mixing. The pressure-head due to gravity can be assumed negligible, so the reservoir element can be used for modelling a fluid loop either in-flight (0-g) or ground-based (1-g).

ESATAN Input:-

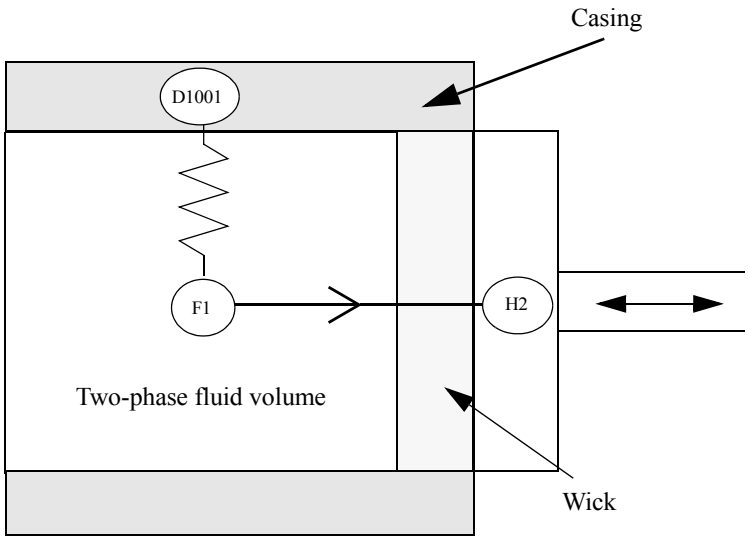


Figure 5-16. Two-phase reservoir element

$ELEMENT name:-

RESERVOIR_2PH Two-phase reservoir

$SUBSTITUTIONS data:-

C_WALL	=	r-val;	(wall capacitance)
DIAM	=	r-val;	(diameter)
G_WF	=	r-val;	(wall-fluid conductance)
LENGTH	=	r-val;	(reservoir length)
MFLOW	=	r-val;	(initial mass flow rate)
PRESS	=	r-val;	(initial pressure of the fluid)
Q_EXT	=	r-val;	(external heat flux into the casing)
Q_INT	=	r-val;	(internal heat flux into the fluid)
RUG	=	r-val;	(surface roughness of the reservoir)
VOID_FRAC	=	r-val;	(fluid void fraction)
X_COORD, Y_COORD, Z_COORD	=	r-val;	(x,y,z-coordinates)

Defaults provided for this element are:-

C_WALL	=	0.0
G_WF	=	0.0
MFLOW	=	0.0
Q_EXT	=	0.0
Q_INT	=	0.0
RUG	=	0.0
VOID_FRAC	=	0.0
X_COORD, Y_COORD, Z_COORD	=	0.0

Input user constants:-

Q_ext	(external heat flux into the casing)
Q_int	(internal heat flux into the fluid)

Output user constants:-

Void_frac	(fluid void fraction)

Connection points:-

H2	-	Inlet/Outlet
D1001	-	Casing

The reservoir element consists of one F-node modelling the working fluid volume and a zero-length H-node for the inlet/outlet. These nodes are connected by a mass flow link. The casing is modelled by a D-node.

The reservoir casing is considered isothermal, with a user-defined capacitance (C_WALL). The geometry is taken as cylindrical; the user has to define the diameter (DIAM) and the length of the reservoir (LENGTH). A thermal conductance between the casing and the fluid may be specified by the user (G_WF).

The user can apply a heat flux either directly into the fluid or into the casing. The initial values are given by the substitution data Q_INT and Q_EXT, respectively, and they can be varied during solution via the user constants Q_int and Q_ext.

The user can track the volume of liquid inside the reservoir via the user constant Void_frac. If the reservoir is full of liquid, Void_frac = 0.0; if it is full of vapour, Void_frac = 1.0. The amount of liquid initially present is set by the substitution data VOID_FRAC.

The reservoir is not applicable for direct steady state computation, i.e. solver FGENSS. In this case, the loop pressure should be controlled by a fixed pressure boundary node rather than a reservoir.

This reservoir element can be easily adapted to a flow-through reservoir by connecting the working fluid node F1 to the fluid network using a mass flow link, hence making it an inlet. The H-node will now be the outlet of the reservoir. Phase separation will be active from the inlet to the outlet.

5.17 Flexible Hose

General Concepts:-

This element models a flexible, corrugated hose. The flexible hose is represented by fluid nodes, connected by mass flow links. The overall hydraulic resistance for the flexible hose is calculated using the formulae described below. The formula was given in [17].

Hydraulic resistance for a flexible hose without bending is represented by:

$$k \equiv \Delta P / \tfrac{1}{2}\rho u^2 = N\left(1-\left[\frac{D_I}{D_I+\alpha S}\right]^2\right)^2$$

Where D_I is the inner diameter, S the pitch. N, the number of corrugations, is calculated from S, the hose length L (i. e., $N = L/S$) and α takes the value 0.44 in the above reference.

Where the hose bends, there is an additional resistance

$$k_B = f(r_B/D_I)(\theta/90^0)^b$$

Where r_B is the bend radius, θ the bend angle in degrees, and f and b are given below:

r_B/D_I	f
1.0	0.37
1.5	0.27
2.0	0.21
2.5	0.16
3.0	0.12
3.5	0.10
4.0	0.08

Table 5-1. Variation of *f* as a function of r_B/D_I

$$b = \begin{cases} 0.5 & (0^0 < \theta < 90^0) \\ 1.0 & (90^0 \le \theta) \end{cases}$$

ESATAN Input:-

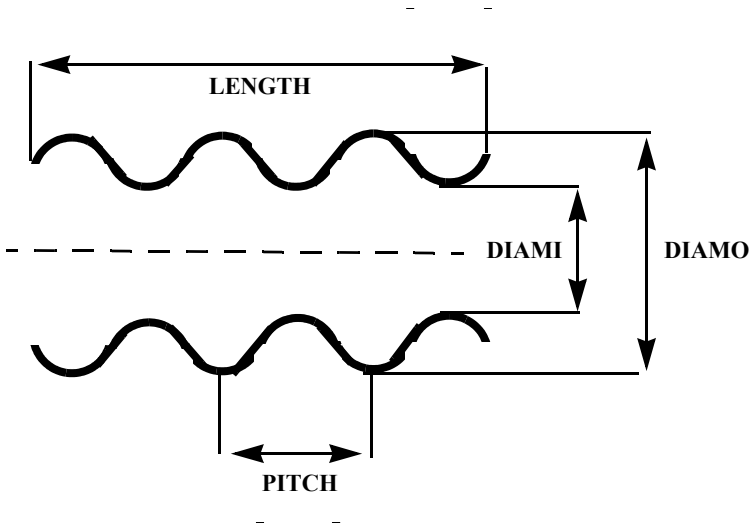


Figure 5-17. Flex hose section

$ELEMENT name:-

FLEX_HOSE Flexible, corrugated hose

$SUBSTITUTIONS data:-

BEND_ANGLE	=	r-val;	(bend angle of the hose in degree)
DIAM	=	r-val;	(average diameter of the hose)
ETA_METH	=	z-val;	(overall hydraulic resistance calculation method: '*' or 'USER')
ETA_USER	=	r-val;	(user-provided overall hydraulic resistance)
FLA	=	r-val;	(flow area)
DIAMI	=	r-val;	(inner diameter of the hose)
LENGTH	=	r-val;	(length of the hose)
MFLOW	=	r-val;	(initial mass flow rate)
NMESH	=	i-val;	(meshing - number of nodes in the hose)
N_END	=	i-val;	(fluid outlet node number)
PRESS	=	r-val;	(initial fluid pressure)
DIAMO	=	r-val;	(outer diameter of the hose)

PITCH	=	r-val;	(pitch of corrugation in the hose)
PITCH_COEF	=	r-val;	(pitch coefficient α)
RUG	=	r-val;	(surface roughness)
TEMP	=	r-val;	(initial fluid temperature)

Defaults provided for this element are:-

BEND_ANGLE	=	0.0
DIAM	=	(DIAMI + DIAMO) /2
ETA_METH	=	'*'
ETA_USER	=	0.0
FLA	=	$\pi \cdot DIAM^2 / 4$
MFLOW	=	0.0
NMESH	=	3
N_END	=	NMESH
PITCH_COEF	=	0.44
RUG	=	0.0

Input user constants:-

Bend_angle	=	r-val;	(bend angle of the flexible hose)
Eta_user	=	r-val;	(user provided ETA value)
Eta_meth	=	z-val;	(calculation method for ETA)

Output user constants:-

Bend_radius	=	r-val;	(bend radius of the flexible hose)

Connection points:-

F1	-	Fluid inlet
F*N_END*	-	Fluid outlet

The user can define the number of fluid nodes required in the hose using NMESH. Note that NMESH must be at least 3. The node numbering starts from 1. NMESH includes the inlet and outlet nodes. By default, the fluid outlet will be numbered NMESH. However, the user can easily define the outlet node number by setting N_END.

For instance, if the user defines NMESH = 5 and N_END takes the default value, the following nodes will be generated:

F1
F2
F3
F4
F5

On the other hand, if the user defines NMESH = 5 and N_END = 10, the following nodes will be generated:

F1
F2
F3
F4
F10

By defining N_END, the user is able to change the meshing within the hose without changing the numbering of the outlet. This will avoid having to change the connecting mass flow links in the main model.

The method for calculating overall hydraulic resistance is specified by substitution data ETA_METH. The default method (ETA_METH = '*') is to calculate the hydraulic resistance values using the formulae described above. If ETA_METH has been set to 'USER', the values of the overall hydraulic resistance have to be defined by setting the substitution data ETA_USER.

Bend angle and radius of the flexible hose, and calculation method and user provided values for ETA, can be changed during the run using the user constants Bend_angle, Bend_radius, Eta_meth and Eta_user, respectively.

5.18 Peltier Element

General Concepts:-

The Peltier element, also known as a thermoelectric heater or cooler, is a semiconductor-based electronic component that functions as a heat pump. It is composed of pairs of thermoelectric bars placed between two ceramic plates. By applying a low voltage DC power source to the element, heat will be moved from one side (cold side) to the other (hot side). A Peltier element may be used for either heating or cooling by reversing the direction of the applied current.

Peltier-effect devices are therefore highly suitable for precise temperature-control applications as well as where space limitations and reliability are important considerations.

ESATAN Input:-

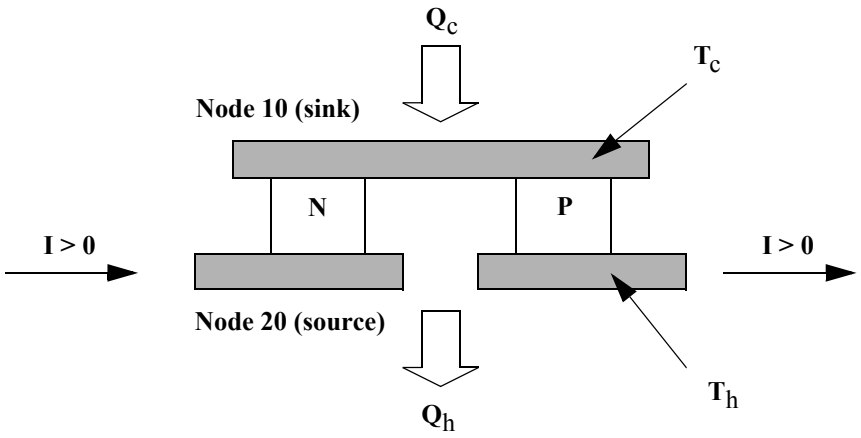


Figure 5-18. Peltier section

$ELEMENT name:-

PELTIER	Peltier Element

$SUBSTITUTIONS data:-

C_SOURCE	=	r-val;	(thermal capacity for cold-side node)
C_SINK	=	r-val;	(thermal capacity for hot-side node)
DATA_TEMP	=	r-val;	(nominal hot-side temperature)
GAS_GL	=	r-val;	(effective conductance of gas filling)
GEOM_FACTOR	=	r-val;	(geometric factor of bars: cross-section area / height (cm))

MATERIAL_TYPE	=	i-val;	(Set of thermoelectric material properties 1 - $Z=2.67x10^{-3}$ 2 - $Z=2.46x10^{-3}$)
MAX_DELTAT	=	r-val;	(max temperature difference between hot and cold sides (no heat load))
MAX_INTENSITY	=	r-val;	(current corresponding to MAX_DELTAT)
MAX_POWER	=	r-val;	(max heat load supported by the element (no temperature difference))
MAX_VOLTAGE	=	r-val;	(voltage corresponding to MAX_DELTAT)
METH	=	z-val;	(requested method to compute the geometric factor and the number of couples. Can be ’POWER’ or ’VOLTAGE’ or ’ ’)
N_COUPLES	=	i-val;	(number of pairs of bars)
TEMP	=	r-val;	(initial temperature of the nodes)

Defaults provided for this element are:-

C_SOURCE	=	0.0
C_SINK	=	0.0
DATA_TEMP	=	0.0
GAS_GL	=	0.0
GEOM_FACTOR	=	0.0
MATERIAL_TYPE	=	2
MAX_DELTAT	=	0.0
MAX_INTENSITY	=	0.0
MAX_POWER	=	0.0
MAX_VOLTAGE	=	0.0
METH	=	’ ’
N_COUPLES	=	0

Input user constants:-

Ipelt	=	r-val;	(current going through the element)
Thermal_source	=	r-val;	(thermal load on the source face)
Thermal_sink	=	r-val;	(thermal load on the sink face)

Output user constants:-

Cop	=	r-val;	(coefficient of performance of the element)
Geom_factor	=	r-val;	(geometric factor)
N_couples	=	r-val;	(number of pair of bars)
Ptot	=	r-val;	(total power)
Q_source	=	r-val;	(total Q on the source face)

Q_sink	=	r-val;	(total Q on the sink face)
Vtot	=	r-val;	(total tension or voltage)

Connection points:-

D10	-	Sink face, usually the hot face (if I>0)
D20	-	Source face, usually the cold face (if I>0)

The thermo-electric bars are always configured in pairs. The substitution data N_COUPLES represents the number of pairs of bars. The bars are characterised by their Seebeck coefficient, their thermal conductivity, their electrical resistivity, and the ratio of their cross-sectional area to their height. This last is the geometric factor the PELTIER element needs as input (GEOM_FACTOR).

In practice the parameters N_COUPLES and GEOM_FACTOR are not usually provided in the manufacturer's data for an off-the-shelf Peltier device, and are difficult for the thermal engineer to ascertain. Hence, the PELTIER element may be supplied with an alternative set of parameters by which its performance is determined. There are two options for this, selected by setting the substitution data METH to 'POWER' [recommended] or 'VOLTAGE', in which case the number of pairs of bars and the geometric factor are deduced from the other parameters and made available as user constants. The default value (METH = ' ') signals that N_COUPLES and GEOM_FACTOR are expected as direct user inputs. Both methods should in theory give the same results, but in practice one may give a performance closer to the manufacturer's data. Users should choose the appropriate method according to the set of data at their disposal, as shown in Table 5-2.

Substitution data	METH = ' ' (default)	METH = 'VOLTAGE'	METH = 'POWER'
N_COUPLES	◀		
GEOM_FACTOR	◀		
DATA_TEMP		◀	◀
MAX_DELTAT		◀	◀
MAX_INTENSITY		◀	◀
MAX_VOLTAGE		◀	
MAX_POWER			◀

Table 5-2 Required substitution data according to METH

The following equations are used to determine equivalent values for GEOM_FACTOR and N_COUPLES according to the value of METH. (Note that all temperatures in the following are assumed to be absolute.)

$$GEOM\_FACTOR = \frac{MAX\_INTENSITY}{T_{COLD}} \times \frac{Ro}{S}$$

$$N\_COUPLES = \begin{cases} \text{int}\left(\dfrac{\frac{1}{2} \times MAX\_VOLTAGE}{S \times T_{HOT}} + 0.5\right) & (METH = \text{'VOLTAGE'}) \\ \text{int}\left(\dfrac{\frac{1}{2} \times MAX\_POWER/MAX\_INTENSITY}{S \times \left(T_{HOT} - \frac{1}{2} \times T_{COLD}\right)} + 0.5\right) & (METH = \text{'POWER'}) \end{cases}$$

where

$$T_{HOT} = DATA\_TEMP$$

$$T_{COLD} = T_{HOT} - MAX\_DELTAT$$

$$T_{MEAN} = \frac{1}{2}(T_{HOT} + T_{COLD})$$

$$Ro = Ro_0 + Ro_1 \times T_{MEAN} + Ro_2 \times T_{MEAN}^2$$

$$S = S_0 + S_1 \times T_{MEAN} + S_2 \times T_{MEAN}^2$$

Two material sets are provided within the PELTIER element and reflect the material properties of PELTIER elements provided by two of the main manufacturers, Marlow and Melcor. (Typically Marlow is Type 1 and Melcor Type 2). This is controlled by the substitution data MATERIAL_TYPE. A corresponding set of constants Ka_n, Ro_n and S_n will be automatically set for computing the total thermal conductivity, *Ka*, electrical resistivity, *Resi*, and Seebeck coefficient, *Seebeck*, of the material according to the following equations:

$$T_{AV} = \frac{1}{2}(T_{HOT} + T_{COLD})$$

$$Resi = 2 \times N\_COUPLES \times \frac{Ro_0 + Ro_1 T_{AV} + Ro_2 T_{AV}^2}{GEOM\_FACTOR}$$

$$Seebeck = 2 \times N\_COUPLES \times (S_0 + S_1 T_{AV} + S_2 T_{AV}^2)$$

$$\text{with units} \begin{cases} Ka = \dfrac{W}{cmK} \\ Ro = \Omega cm \\ S = \dfrac{V}{K} \end{cases}$$

For MATERIAL_TYPE = 1 the default values for electrical resistivity and Seebeck coefficient are:

$$\textit{with by default} \begin{cases} S_0 = 2.1449\times10^{-5} \\ S_1 = 8.9817\times10^{-7} \\ S_2 = -9.5593\times10^{-10} \end{cases} \quad \textit{and} \begin{cases} Ro_0 = 4.845\times10^{-5} \\ Ro_1 = 1.5489\times10^{-6} \\ Ro_2 = 5.9514\times10^{-9} \end{cases}$$

For MATERIAL_TYPE = 2 the default values for electrical resistivity and Seebeck coefficient are:

$$\textit{with by default} \begin{cases} S_0 = 2.2224\times10^{-5} \\ S_1 = 9.306\times10^{-7} \\ S_2 = -9.905\times10^{-10} \end{cases} \quad \textit{and} \begin{cases} Ro_0 = 5.112\times10^{-5} \\ Ro_1 = 1.634\times10^{-6} \\ Ro_2 = 6.279\times10^{-9} \end{cases}$$

Inside the Peltier element itself, a gas could exist, although such an element is usually working in a vacuum environment. The substitution data GAS_GL is used to define the conductance between the plates of the Peltier element due to this gas. Therefore the conductance between the two faces (D10 and D20) can be represented as:

$$GL(10,20) = GL\_GAS + 2 \times N\_COUPLES \times GEOM\_FACTOR \times Ka$$

$$Ka = (Ka_0 + Ka_1 T\_avr + Ka_2 T\_avr^2)$$

$$\textit{with by default} \begin{cases} Ka_0 = 6.2605\times10^{-2} \\ Ka_1 = -2.777\times10^{-4} \\ Ka_2 = 4.131\times10^{-7} \end{cases} \quad \textit{and} \quad T\_avr = \frac{T10 + T20}{2}$$

The default values of thermal conductivity for MATERIAL_TYPE 1 and 2 are as follows:

$$MATERIAL\_TYPE = 1 \begin{cases} Ka_0 = 5.8077\times10^{-2} \\ Ka_1 = -2.5921\times10^{-4} \\ Ka_2 = 3.8553\times10^{-7} \end{cases} \quad MATERIAL\_TYPE = 2 \begin{cases} Ka_0 = 6.2605\times10^{-2} \\ Ka_1 = -2.777\times10^{-4} \\ Ka_2 = 4.131\times10^{-7} \end{cases}$$

The following equations are used to compute the fluxes associated with the sink face (node 10) and the source face (node 20):

$$QI10 = Resi \times Ipelt^2 + Thermal\_sink$$

$$QR10 = Seebeck \times T10 \times Ipelt$$

$$QI20 = \frac{1}{2} Resi \times Ipelt^2 + Thermal\_source$$

$$QR20 = -Seebeck \times T20 \times Ipelt$$

The Peltier element can be characterised by its Coefficient of Performance, defined with the following equation:

$$Cop = \frac{Q\_source}{Ptot}$$

The equations used to evaluate Q_source and Ptot are detailed below:

$$Q\_source = GL(10,20) \times (T20 - T10) - QI20 - QR20 + Thermal\_source$$

$$Ptot = Resi \times Ipelt^2 + Seebeck \times (T10 - T20) \times Ipelt$$

5.19 PID Controller

General Concepts:-

This element simulates a three-term controller with a Proportional, Integral and Derivative control. The PID element can work as a P, PI, or PID to regulate the system.

The element does not have any network representations. It purely contains the PID control logic and constants to maintain its state.

By default, only a positive response is provided.

If the PID element is used in steady-state, then iteration count is used as 'pseudo-time'.

ESATAN Input:-

$ELEMENT name:-

PID	PID Controller

$SUBSTITUTIONS data:-

D_GAIN	=	r-val;	(derivative gain)
I_GAIN	=	r-val;	(integral gain)
P_GAIN	=	r-val;	(proportional gain)

Defaults provided for this element are:-

D_GAIN	=	0.0
I_GAIN	=	0.0
P_GAIN	=	0.0

Input user constants:-

Ctrl_max	=	r-val;	(upper controller limit)
Ctrl_min	=	r-val;	(lower controller limit)
Ctrl_neg	=	z-val;	('OFF' = positive response only 'ON' = negative response allowed)
Delta_min	=	r-val;	(minimum value for the PID to operate)
Measure	=	r-val;	(measure of the control variable)
Onoff	=	z-val;	(set the controller state to On or OFF)
Setpoint	=	r-val;	(set point to reach)

Default for Input user constants:-

Ctrl_max	=	1.0D10;
Ctrl_min	=	0.0;

Ctrl_neg	=	'OFF';
Delta_min	=	0.0;
Measure	=	0.0;
Onoff	=	'ON';
Setpoint	=	0.0;

Output user constants:-

Ctrl_avr	=	r-val;	(average PID control or action)
Ctrld	=	r-val;	(derivative control or action)
Ctrli	=	r-val;	(integral control or action)
Ctrlp	=	r-val;	(proportional control or action)
Ctrl_tot	=	r-val;	(total PID control or action)
Ctrl_inc	=	r-val;	(incremental PID control or action)
Ctrl_state	=	z-val;	(controller state: ACTIVE, OFF, OVERLOAD, NEUTRAL, I_OVERLOAD, MINIMUM)

Note:

The positive responses are limited by Ctrl_min [threshold] and Crtl_max [saturation].

The negative responses are limited by -Ctrl_min [threshold] and -Crtl_max [saturation].

The user can keep track of the controller's state with Ctrl_state as follow:

ACTIVE	if Onoff is ON (default)
I_OVERLOAD	if ABS(Ctrli) > Ctrl_max
MINIMUM	if ABS(Ctrl_tot) < Ctrl_min
NEUTRAL	if Abs(Measure - Setpoint) < ABS(Delta_min)
OFF	if Onoff is OFF
OVERLOAD	if ABS(Ctrl_tot) > Ctrl_max

A use-case example for using a PID could be the control of a middle node temperature of a bar element. A sinusoidal heat load is input at one end of the bar (node 100) and the aim is to keep the middle node temperature (node 16) at a constant temperature of -7°C.

The PID element will then be used to input some heat load on node 16 to maintain it at a constant -7°C. The element is physically limited to a maximum deliverable heat load of 60W and can not work as a heat extractor (positive response only).

The element will be used as follows in ESATAN:

```
Element Definition:

 $MODEL PID_ELEM
   $ELEMENT PID
   $SUBSTITUTIONS
      D_GAIN = 0.1;
      I_GAIN = 3.5;
      P_GAIN = 6.0;
 $ENDMODEL PID_ELEM

Initialise data within the $INITIAL block,

#
$INITIAL
#
# Initialise PID Controller Constants
#
       PID_ELEM:Ctrl_max = 60.0
       PID_ELEM:Ctrl_min = 0.0
       PID_ELEM:Measure = T16
       PID_ELEM:Setpoint = -7.0

Heat load input into the system

#
$VARIABLES1
#
       DOUBLE PRECISION PI
       PARAMETER (PI = 3.14159265D0)
#
# Set input heat source
#
       QR100 = 45.0D0 + (25.0D0 * SIN(2.0D0 * PI * TIMEN/20.0D0))
#
# Update PID set point and assign heat load on control point
#
       PID_ELEM:Measure = T16
       QR16 = PID_ELEM:Ctrl_tot
```

The following picture shows the heat load input into node 100 and the measured temperature at node 16.

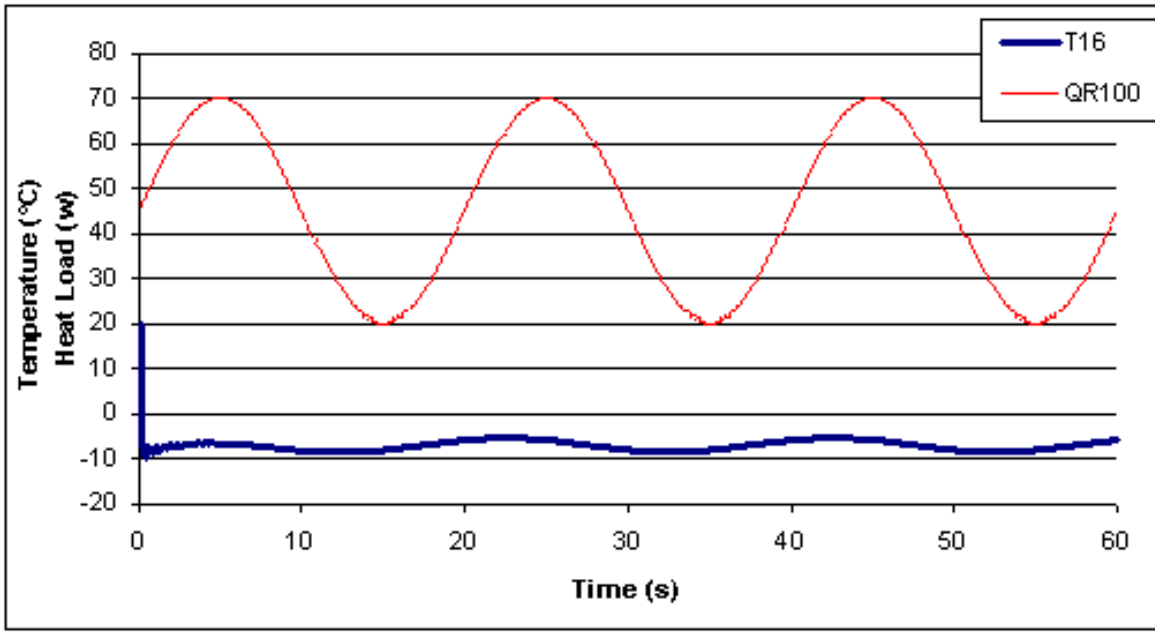


Figure 5-19. PID Example Results

6. LIBRARY

6.1 Introduction

Chapter 6 deals with the ESATAN library routines, categorised into functional areas from Section 6.2 onwards. The current section deals with some important background of the ESATAN package in order to aid understanding of the use of the library.

6.1.1 Execution Program Organisation

The ESATAN library routines may be used from any of the operations blocks, and library functions may also be used in the data blocks wherever Mortran statements are allowed, i.e. in node or conductor definitions.

The ESAFOR program generates a FORTRAN program from the Model Database (MDB). Each operations block within the model is translated into a corresponding FORTRAN subroutine. For example there will be one subroutine generated for each $INITIAL block within the model (Including all submodels). Similarly for $VARIABLES1 and $VARIABLES2 blocks. As only one $EXECUTION block and one $OUTPUTS block are allowed within a complete model only one subroutine for each will be generated. If these blocks are present within any submodel then they will be ignored.

The generated FORTRAN program is organised as follows. At the start of a solution the numeric data stored in the model data base is read. After loading the common blocks, a call to the subroutine generated from the $INITIAL block of the level-zero model (Main model) is made. This subroutine immediately calls in turn the lower level subroutines, generated from the $INITIAL blocks of the contained submodels, before executing any statements it may contain. Following this, the subroutine generated from the $EXECUTION block is called. The appropriate solution routines should be called from this block and no other.

The solution routines themselves call the subroutines corresponding to the main model's $VARIABLES1 and $VARIABLES2 blocks. The former is executed at the start of each steady-state solution iteration, or, for a transient solution, before the first timestep (for initialisation purposes) and then during each timestep calculation. The latter is executed at the end of a steady state solution or at the end of each timestep in a transient solution.

At the start of the main model's $VARIABLES1 subroutine, dynamic data is read from the ACD file (3.10) and the states of any thermostatic heaters defined in the model (6.4.48) are updated; this applies to the whole model, including submodels. Then, in turn, the lower-level subroutines generated from the same block in any submodels are called. The contents of each $VARIABLES1 block are executed *after* calling the subordinate routines.

Similarly, for $VARIABLES2, the states of all thermostatic heaters are updated in the main-model routine before subordinate (submodel) routines are called, and the contents of each $VARIABLES2 block are executed after those of any child submodels.

The subroutine generated from the $OUTPUTS block is also called by the solution routine at the following times: at the end of a steady state solution; at the start, at every output interval and at the end of a transient solution.

6.1.2 Control Constants

ESATAN provides control constants principally to control solution and output. A list of control constants is given in Appendix E, together with type, default values and purpose. Many of these may be set by the user in the $CONTROL block of the $CONSTANTS block, others may not be set by the user but are available for reference.

A description of the usage of control constants is given elsewhere in this chapter at the appropriate points.

6.1.3 Library Arguments

This section describes the usage of certain arguments which are common to a number of library routines.

ZENTS:-

A combination of either a) A, ALP, C, CMP, EPS, FD, FF FL, FLA, FH, FM, FQ, FR, FRG, FST, FW, FX, FY, FZ, L, P, PHI, QA, QAI, QE, QEI, QI, QR, QS, QSI, T, VDT, VOL, VQ and any user nodal entities that have been defined; or b) GF, GL, GP, GR, GV, M and G (this last denoting conductors of all types).

It is passed as a character literal or variable, with names separated by commas, in order to request output of node properties or conductances. The presence of **-INC** or **-DEC** behind an entity determines the sequence in which the print-out should be given but does not request that the particular entity should appear. A sequence instruction in a ZENTS of node properties is used for all of the entities present. In a ZENTS of conductors, however, a sequence instruction is only valid for the entity to which it refers. The lack of a sequence instruction causes printing in ascending order to node reference number. Only one sequence instruction is allowed in any ZENTS. If there are two or more then the final valid one is used.

Examples:-

ZENTS = 'T-INC,C'
C printed in order of increasing temperature;

ZENTS = 'T-INC,T,C'
T and C printed, both in order of increasing temperatures;

ZENTS = 'G,GR-DEC'
All conductors printed; GR in decreasing order, GL, GF GV, M and GP all in increasing order of node reference number.

Note: -INC or -DEC should not be used with a character entity.

ZENT:-

This is similar to ZENTS but consists of only a single entity name. Unless otherwise stated, the -INC and -DEC modifiers cannot be used.

ZLABEL:-

A character literal or variable, this is an argument of many output routines. It is used to specify a group of nodes (said to be "of ZLABEL") for which calculation or a data print-out is required. There are two ways to do this:-

1. The nodes requested are those which, as well as being contained in the sub-model given by CNAME (see below), have ZLABEL as a substring of their node label.

2. The nodes are specified by a list of node numbers, starting with a '#' character. The general format for this is as follows:
'#[sub:]nodelist[;[sub:]nodelist][;[sub:]nodelist]...'
where sub is a submodel concatenation and nodelist is a list of nodes. Individual node numbers are separated by a comma and ranges of nodes indicated by a hyphen. If the submodel concatenation is omitted then the associated nodelist is assumed to refer to the current submodel. All submodel concatenations in ZLABEL are assumed to be relative to CNAME. A node alias can be used instead of a node number in a node list, and the same submodel concatenation can be used more than once.

If ZLABEL is a single blank then all nodes in CNAME are considered to be required.

Example:-

If in the main model
L10 = 'TOPSIDE'
L15 = 'LOWERLEFT'

and in submodel sub1
L20 = 'SIDEWAYS'
L25 = 'LOWERRIGHT'
L30 = 'TOPHOLE'

then

with CNAME = CURRENT

ZLABEL = 'TOP'	refers to nodes 10 and 30
ZLABEL = 'SIDE'	refers to nodes 10 and 20
ZLABEL = '#25'	does not match a node
ZLABEL = '#10, 20-30'	refers to node 10
ZLABEL = '#10; sub1:20-30'	refers to node 10 in the main model and

	nodes 20, 25 and 30 in sub1
ZLABEL = ' '	refers to all five nodes

with CNAME = sub1

ZLABEL = 'TOP'	refers to node 30
ZLABEL = 'SIDE'	refers to node 20
ZLABEL = '#25'	refers to node 25
ZLABEL = '#10, 20-30'	refers to nodes 20, 25 and 30
ZLABEL = '#10; sub1:20-30'	does not match a node (there is no submodel sub1 which is a submodel of sub1)
ZLABEL = ' '	refers to all three nodes in sub1 (20, 25 and 30)

CNAME:-

An input in almost all of the output routines, this is the concatenated model name of the sub-model for which information is required. The rules for specification of CNAME are described in Section 3.1.1; the user should note specifically that the concatenation need contain only sufficient names to ensure an unambiguous reference. Specification of a full concatenation, however, is not treated as an error. It is understood to include all of the included sub-models of the sub-model referenced. The user has the option, however, of referencing a sub-model without its included sub-models by adding the word "ONLY" to the concatenation.

The current sub-model may be identified by setting CNAME to **CURRENT**. The words **CURRENT** and **ONLY** are reserved names and, therefore, not available as model names.

CNAME is not a character variable, and thus is not treated as such an input, i.e. the concatenated submodel name should not be given as a literal character variable. The following examples should make this clear.

Examples:-

CNAME = ENGINE:EXHAUST:ONLY
refers to sub-model ENGINE:EXHAUST, but not its included sub-models;

CNAME = CURRENT
refers to the current sub-model and its included sub-models;

CNAME = CURRENT:ONLY
refers to the current sub-model, but not its included sub-models.

Note: CURRENT is not a model name and therefore must not be used in concatenation with other submodel names. For example, the specification CURRENT:ENGINE:EXHAUST:ONLY is not permitted.

6.1.4 Progress Monitoring

During a solution run a file *model*.MON is created. This file is periodically updated with details of the current state of the analysis. The data output are as described for subroutine PROGRS (Section 6.8) and the frequency of output is determined by the control constant PRGFBK. By default PRGFBK is set to 10, implying that a line of data is written every ten iterations of a steady state analysis and every ten time steps of a transient. In addition to controlling the frequency of output using PRGFBK, the user can also call PROGRS with UNIT = 0 to generate an output to the *model*.MON file.

6.1.5 Output Streams

ESATAN uses a number of output streams during solution, and it is possible that in the future more will be required. It is therefore recommended that users defining their own input / output files should not use stream numbers 1-20: These are considered reserved for use by ESATAN.

The following stream numbers are currently used:

2	*model*.COM
3	*model*.PAR
4	*model*.DMP
6	*model*.OUT
8	*model*.UNF and PRNCSV output files
9	*model*.MON
10	*model*.MDB
11	*model*_PAR*nnnn_PARNAM*.csv
13	*model*.TRA
14	*model*_PAR*nnnn_PARNAM*.GFF*n*

6.2 Data Dumping

DMPGFS	Deprecated. Replaced by DMPGFF
DMPGFF	Dump data in ESATAN neutral file format
DMPTMD	Dump data to a binary file in HDF format
LOADT	Initialise temperatures from an existing TMD file
FETCHT	Retrieve temperatures from file
SAVET	Save temperatures in file
DMPTHM	Dump basic thermal model data to .CSV files
DMPFR	Dump frequency response to file

6.2.1 GFF Neutral File Data Dump

Name:

SUBROUTINE **DMPGFS** Deprecated. Replaced by DMPGFF

SUBROUTINE **DMPGFF** (ZLABEL, ENTS, CNAME, FNAME)

ZLABEL	-	CHARACTER:Node group specification(see Section 6.1.3)
ENTS	-	CHARACTER:Entities to be output
CNAME	-	Submodel reference (see Section 6.1.3)
FNAME	-	CHARACTER:Basename of output file

Purpose:

Produces a dump of entity data to a file in the GFF neutral format*. This data can then be loaded into a graphical post-processing program such as the ESATAN Thermal Suite's ThermNV. Note that a replacement binary format is now available for storing and loading data into ThermNV. See Section 6.2.2. The binary format is far more efficient, producing smaller dump files and allowing larger networks to be processed by ThermNV.

The currently supported entities that can be written to a GFF file are: nodal attributes, conductances, and user constants.

Node and conductor data is output for the group of nodes specified by ZLABEL, relative to the submodel given by CNAME. Conductors having only one node in the group are ignored. The user constants are those defined in submodel CNAME or its included models; ZLABEL is irrelevant for these.

The entities to be output are specified in ENTS using the data-block names NODES, CONDUCTORS and CONSTANTS as keywords, separated by commas and enclosed in quotes. The first two may be qualified by listing the required entities inside brackets. Any block name may be omitted if no entities of that type are required.

If FNAME is specified then the file generated is named FNAME.GFF; if FNAME is a blank string, then the name of the current submodel is used.

The format for writing real values to the GFF file can be specified via the control constant FORMAT using Fortran 77-like edit descriptors, e.g. 'F15.8'. If FORMAT = '*' then a default format of 'G12.6' is used. (In the special case of an entity value being zero, it is written as '0.0' in the file in order to save space.)

The routine would normally be called from the $OUTPUTS block or from $VARIABLES2, although it may be called from other operations blocks.

* An ASCII data format developed by ESA. The origin of the initials 'GFF' is somewhat unclear; possibly 'GRABE File Format'.

Normally, all the results from different timesteps of a transient solution will be recorded in a single GFF file. On the other hand, if consecutive solutions are performed, either transient or steady state, the results for each one will usually go to separate GFF files distinguished by an integer appended sequentially to the 'GFF' extension, starting at 2. This behaviour is determined by whether or not the current solution time (control constant TIMEN) has increased since the previous call to DMPGFF with the given basename. Thus, in order to store results from several steady-state solutions in a single GFF file, the user can increase TIMEN between the solutions.

Examples:

Maximal set of node and conductor data written to file every output interval:

```
$OUTPUTS
#
# GFF file for whole model:
#   - All nodes
#   - All nodal entities
#   - All conductors
#   - All user constants.
#   - File name: <model>.GFF.
#
      CALL DMPGFF(' ', 'NODES, CONDUCTORS, CONSTANTS',
     &            CURRENT, ' ')
```

Subset of data written to file every timestep:

```
$VARIABLES2
#
# GFF file for submodel GIZMO:
#   - Nodes 1-100
#   - Temperature, internal heat source
#   - Linear & radiative conductances
#   - No user constants
#   - File name: Friday.GFF
#
      CALL DMPGFF(
     &   '#1-100',
     &   'NODES(T, QI), '      //
     &   'CONDUCTORS(GL, GR)',
     &   GIZMO,
     &   'Friday')
```

Restrictions:

The nodal entity list must be the same for all calls to the routine which output to the same GFF file. This is because the nodal record mapping is written once only, in the file header.

The node-selection string ZLABEL is not applied to view factor conductors (GVs).

The nodal entity qualifier in ENTS is not applied to user-defined nodal entities.

The conductor-type qualifier in ENTS is not applied to view factors (GVs).

A maximum of 5 different file basenames can be used. This routine makes use of I/O streams 15-19 (see Section 6.1.5).

Note:
For backward-compatibility reasons, a blank string may be used for ENTS to mean all entities; i.e. ' ' is equivalent to 'NODES, CONDUCTORS, CONSTANTS'. This behaviour may be changed in a future release of ESATAN.

6.2.2 Thermal Model Data Binary File Dump

Name:

SUBROUTINE DMPTMD (ZLABEL, ENTS, CNAME, FNAME)

ZLABEL	-	CHARACTER:Node group specification(see Section 6.1.3)
ENTS	-	CHARACTER:Entities to be output
CNAME	-	Submodel reference (see Section 6.1.3)
FNAME	-	CHARACTER:Basename of output file

Purpose:

Produces a dump of entity data to a binary TMD (Thermal Model Data) file in the HDF (Hierarchical Data Format) file format. This data can then be loaded into a graphical post-processing program such as the ESATAN Thermal Suite's ThermNV version 4 or above.

The currently supported entities that can be written to a TMD file are: nodal attributes and conductances.

Node and conductor data is output for the group of nodes specified by ZLABEL, relative to the submodel given by CNAME. Conductors having only one node in the group are ignored.

The entities to be output are specified in ENTS using the data-block names NODES and CONDUCTORS as keywords, separated by commas and enclosed in quotes. The first two may be qualified by listing the required entities inside brackets. Any block name may be omitted if no entities of that type are required. If FNAME is specified then the file generated is named FNAME.TMD; if FNAME is a blank string, then the name of the current submodel is used.

The routine would normally be called from the $OUTPUTS block or from$VARIABLES2, although it may be called from other operations blocks.

Normally, all the results from different timesteps of a transient solution will be recorded in a single TMD file. On the other hand, if consecutive solutions are performed, either transient or steady state, the results for each one will usually go to separate TMD files distinguished by an integer appended sequentially to the 'TMD' extension, starting at 2. This behaviour is determined by whether or not the current solution time (control constant TIMEN) has increased since the previous call to DMPTMD with the given basename. Thus, in order to store results from several steady-state solutions in a single TMD file, the user can increase TIMEN between the solutions.

Examples:

Maximal set of node and conductor data written to file every output interval:

```
$OUTPUTS
#
# TMD file for whole model:
#   - All nodes
#   - All nodal entities
#   - All conductors
#   - File name: <model>.TMD.
#
      CALL DMPTMD(' ', 'NODES, CONDUCTORS', CURRENT, ' ')
```

Subset of data written to file every timestep:

```
$VARIABLES2
#
# TMD file for submodel GIZMO:
#   - Nodes 1-100
#   - Temperature, internal heat source
#   - Linear & radiative conductances
#   - File name: Friday.TMD
#
      CALL DMPTMD(
     &   '#1-100',
     &   'NODES(T, QI), '      //
     &   'CONDUCTORS(GL, GR)',
     &   GIZMO,
     &   'Friday')
```

Restrictions:

The nodal entity list must be the same for all calls to the routine which output to the same TMD file. This is because the file structure is created when the file is initially produced.

The node-selection string ZLABEL is not applied to view factor conductors (GVs).

The conductor-type qualifier in ENTS is not applied to view factors (GVs).

A maximum of 99 different file basenames can be used.

Currently the dumping of user or control-constant data is not supported. If a routine is called with CONSTANTS defined as one of the entities to output, this entity shall be ignored without warning.

Note:
For backward-compatibility reasons, a blank string may be used for ENTS to mean all entities; i.e. ' ' is equivalent to 'NODES, CONDUCTORS'. This behaviour may be changed in a future release of ESATAN.

6.2.3 Load Temperatures from Thermal Model Data TMD file

Name:

SUBROUTINE **LOADT** (FILE, TIME, CNAME)

FILE - Character: TMD file name
TIME - Double Precision Time in TMD file at which to read temperatures
CNAME - Character Submodel name

Purpose:

This routine is used for the initialisation of temperatures from an existing Thermal Model Data file (TMD) for a given time step.

For every matching node between the TMD file and the submodel or submodels defined by CNAME, load the temperatures from those stored in the TMD file for the requested time. If specific nodes exist in the model but not in the TMD, then their temperature is left unchanged. If specific nodes exist in the TMD file but not in the model, then they are ignored. Note that no warning is given in both of these cases.

If the TMD file does not contain the time requested, then linear interpolation is used between the two spanning time steps in the TMD file. If the time requested is before the first time step in the TMD file then the temperatures at the first time step are used. If the time requested is after the last time step in the TMD file then the temperatures at the last time step are used.

The file can be specified with either a full or relative path, and must include the file extension.

Examples:

```
CALL LOADT('MyModel.TMD',1000.D0, CURRENT)
CALL LOADT('MyOtherModel.TMD',750.D0, SUBMOD:ONLY)
```

Restrictions:

An error is given if the specified TMD file does not exist.

6.2.4 Save/Fetch Temperatures

Name:

SUBROUTINE **SAVET** (CNAME)
SUBROUTINE **FETCHT** (CNAME)

CNAME - Character submodel name

Purpose:

Save and fetch nodal temperatures for a submodel or submodels. **SAVET** saves the temperature values writing them to an unformatted file. **FETCHT** retrieves them from an unformatted file overwriting existing temperature values for the submodel/s specified. The name of the file is given by the model name plus the file extension '.UNF'.

Examples:

```
CALL SAVET(CURRENT)

CALL FETCHT(SUBMOD:ONLY)
```

Restrictions:

The file must exist before **FETCHT** is called and the temperatures must correspond to the submodel/s referenced. Only temperatures of thermal nodes are output.

6.2.5 Thermal Model Data CSV File Dump

Name:

SUBROUTINE **DMPTHM** (FILE)

FILE - CHARACTER: Basename for output files

Purpose:

Dump to file basic thermal model data in generic (CSV) format. (Thermal nodes only.)

DMPTHM produces a set of CSV files containing nodal data in vector form (temperature, capacitance and total heat load, one row per node) and conductance matrices (separately linear, radiative and fluidic). The latter are complete NxN matrices (N being the number of nodes in the model), with a zero entry wherever there is no conductor defined between the row and column nodes. This format is designed to be amenable to loading into numeric computation tools such as MATLAB®. There is also a list of nodes in the same row/column order, giving node type, number, label and model name.

FILE is used as the first part of each file name, unless it is blank in which case the model name is used. The files produced are as follows:

FILE.nl.csv -	Node list: number, type, label, model
FILE.nd.csv -	Node data: temperature, capacitance, total heat load (vectors)
FILE.gl.csv -	Linear conductances (matrix)
FILE.gr.csv -	Radiative conductances (matrix)
FILE.gf.csv -	Fluidic conductances (matrix)

Examples:

This code segment,

```
$MODEL DEMO
...
$EXECUTION
        CALL SOLVFM
        CALL DMPTHM(' ')
```

will produce files that look like the following.

DEMO.nl.csv:

```
# Basic Thermal Model Data - Node List
# DEMO                     11 OCTOBER 2007   09:51:30
# ESATAN 10.2.0
#
```

```
# Type,Number,Label,Model
D,11101,"Shear panel 1",DEMO
D,11201,"Shear panel 2",DEMO
D,11301,"Shear panel 3",DEMO
D,11401,"Shear panel 4",DEMO
...
```

DEMO.nd.csv:

```
# Basic Thermal Model Data - Nodes
# DEMO                    11 OCTOBER 2007   09:51:30
# ESATAN 10.2.0
#
# T,C,Q
17.1194,844.795,0.0
21.1469,1027.44,100.0
19.0061,1027.44,0.0
18.8696,844.795,0.0
...
```

DEMO.gl.csv:

```
# Basic Thermal Model Data - Linear Conductances
# DEMO                    11 OCTOBER 2007   09:51:30
# ESATAN 10.2.0
#
# GL
0.0,0.174240E-01,0.871200E-02,0.0,...
0.174240E-01,0.0,0.0,0.123456E-01,...
0.871200E-02,0.0,0.0,0.0,...
0.0,0.123456E-01,0.0,0.0,...
...
```

Restrictions:

The output is for thermal nodes only: fluid nodes and conductors are ignored.

6.2.6 Dump Frequency Response

Name:

SUBROUTINE DMPFR(ZLABEL1, CNAME1, ZLABEL2, CNAME2,
FSTART, FEND, NUMF, FILE)

ZLABEL1	-	CHARACTER: Node group specification for inputs
CNAME1	-	Submodel reference for inputs
ZLABEL2	-	CHARACTER: Node group specification for outputs
CNAME2	-	Submodel reference for outputs
FSTART	-	DOUBLE PRECISION: Start of frequency range (Hz)
FEND	-	DOUBLE PRECISION: End of frequency range (Hz)
NUMF	-	INTEGER: Number of frequency values
FILE	-	CHARACTER: Basename for output files

Purpose:

Dump to file the frequency response of specified nodal temperatures to variations in specified boundary conditions. (Thermal nodes only.)

The frequency response transfer function computed by SLFRTF enables the prediction of how temperature will behave as a result of variations in boundary conditions. A variation could be applied to either a heat load or a boundary temperature (input). The transfer function relates this to the resulting fluctuation in temperature (output) at any diffusion node, assuming the input to be a sinusoidal oscillation at a given frequency. An arbitrary input fluctuation can be decomposed into sinusoidal components of different frequencies, according to Fourier analysis.

If a diffusion node is specified as input, it is assumed to imply a heat load variation of some sort, i.e. a QA, QE, QI, QR or QS. A boundary node as input implies a temperature variation. For output, which is always a temperature variation, it only makes sense to specify diffusion nodes. Any inactive nodes, supernode constituents and fluid nodes included in the input and output specifications are silently ignored, as are boundary nodes in the latter.

The transfer function is usually given as the gain and phase of a particular output over a particular input. DMPFR writes the gains and phases of all possible input/output pairs over the specified frequency range to two files, FILE.fr_gain.csv and FILE.fr_phase.csv, respectively. If FILE is blank then the model name is used as the file basename.

The values of frequency are logarithmically evenly spaced, e.g. 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2} Hz. Gain is in either K/W for a heat load input or dimensionless for a temperature input; phase is in degrees.

The data is written to file as comma-separated values (CSV), in a floating-point format governed by the control constant FORMAT. By default the format is G12.6.

Examples:

```
$MODEL DEMO
   $MODEL SUB1
   $NODES
   D5, T = 0.0, C = 0.456, QR = 25.0;
   D15, T = 0.0, C = 0.0;
   D25, T = 0.0, C = 2.8;
   ...
   $ALIAS
   Top_left = D5;
   Bottom_right = D15;
   Middle = D25;
   ...
   $ENDMODEL SUB1
$NODES
B1, T = 0.0;
B2, T = 0.0;
...
D300, T = 0.0, C = 1.23;
...
$EXECUTION
      CALL SOLVFM
      CALL SLFRTF
      CALL DMPFR('#1, 2', CURRENT,
     &           '#300', CURRENT,
     &           1.0D-7, 1.0D-1, 100,
     &           ' ')
      CALL DMPFR('#Top_left', SUB1,
     &           '#Bottom_right, Middle', SUB1,
     &           1.0D-5, 1.0D-3, 20,
     &           'heater')
$ENDMODEL DEMO
```

This will compute a) the response of temperature at node 300 resulting from temperature fluctuations at nodes 1 and 2, writing the data to DEMO.fr_gain.csv and DEMO.fr_phase.csv; and b) the response of temperature at nodes Bottom_right and Middle resulting from a variation in heat load at Top_left, all within submodel SUB1, writing the data to heater.fr_gain.csv and heater.fr_phase.csv.

Restrictions:

DMPFR should be called only from the $EXECUTION block.

The frequency response is calculated for thermal nodes only, and fluidic conductors (GFs) are ignored.

6.3 Error Reporting

SETERX Report error

6.3.1 Introduction

A new method for the generation and formatting of error messages in the library routines has been introduced in Esatan v6.3. This has involved the removal of the stack which previously stored up to 100 entries. Error messages are now written to either standard output or to an error trace file. The code numbering of messages has been removed, but the classification system remains, i.e. the following categories of errors are output:-

- DIAGNOSTIC
- WARNING
- FAILURE
- FATAL ERROR

6.3.2 Message Format

The general format for all messages is as follows:-

```
#@
#@------------------------------------------------- | MESSAGE    1 |-
#@
#@@@@  WARNING
#@
#@     Reference..........: (SNAME, ERRNUM)
#@
#@     Problem............: This field summarizes the problem
#@     Current action.....: This field describes what ESATAN was doing
#@     Comment............: This field provides extra comments or
#@                          suggested solutions
#@     Context............: This field gives some clues about the error
#@                          context
#@-----------------------------------------------------------------------
#@
```

where:-

SNAME	Routine name
ERRNAM	Error message identifier in the calling routine

The reference information is to help the software developers in case of problems.

6.3.3 Control Constants

To aid the user in selecting the output he would like, an additional three control constants have been set up. These enable him to:-

- select summary or full output messages
- direct messages to an error trace file if desired
- set the code severity level at which messages are printed out

The control constants and their allowed values are as follows:-

MSHORT	(integer)	0 for full message (including summary) 1 for summary message
TRACE	(integer)	0 for all messages to be printed to standard output. 1 for all messages to be printed on an error trace file (extension .TRA). Failure and fatal errors shall continue to be printed to standard output.
MLEVEL	(integer)	0 for all messages to be printed 1 for diagnostic messages to be suppressed 2 for warning messages to be suppressed 3 for failure messages to be suppressed

The default for these control constants is 0, with the exception of MLEVEL, where the default is 2. This is to avoid unnecessary generation of numerous warning messages when running under normal execution conditions. The new error message facility can be utilised by users via the library routine SETERX, see Section 6.3.5.

6.3.4 Obsolete Routines

The removal of the stack from error message generation means that the following Error Reporting routines are now obsolete:-

- PRTCRM
- PRTCR
- PRTC
- WORST

6.3.5 SETERX

Name:

SUBROUTINE **SETERX** (MESAGE , TYPE , ERRNUM , NAME)

MESAGE	-	Character Error message
TYPE	-	Integer Error type
ERRNUM	-	Integer Error number
NAME	-	Character Identifier

Purpose:

This routine allows error reporting to be carried out using the same procedure as within ESATAN library routines. As described within the previous section, the error message string is split into several parts. To distinguish between the different parts of the error message the '@' character is utilised. SETERX scans the argument called MESAGE for '@'s and successively populates the different fields :

- Problem (before first '@')
- Current action (between first and second '@')
- Comment (between second and third '@')
- Context (after fourth '@')

Each field of the message is split automatically, with a 44 characters per line and is designed to split upon words. The character sequence '<NL>' can be used at any place to force a carriage return inside a field.

The class of error depends upon the value defined using TYPE and can be;

1	Diagnostic
2	Warning
3	Failure
4	Fatal

If TYPE is set to 4 the solution will halt and the error message printed. All other error message levels allow the solution to continue.

A unique error number can be assigned to the message via the argument ERRNUM, and is automatically displayed. This allows error messages to be distinguished in the output.

The final argument NAME allows a name to be given to the error message and is useful when SETERX is called from within user defined subroutines or functions.

As described with the previous section, the control constants MSHORT, MLEVEL and TRACE can be used in conjunction with SETERX.

Examples:

Input:

```
            SNAME = 'LSLWAE'
            MESAGE =
                  &  'This is<NL>How to@' //
                  &  'Combine<NL>the@' //
                  &  'Different<NL>control@' //
                  &  'Signs<NL>at your<NL>disposal.'
            TYPE = 1
            ERRNUM = 2
      C
            CALL SETERX(MESAGE , TYPE , ERRNUM , SNAME)
```

Output:

```
#@
#@-------------------------------------------------- | MESSAGE    1 |-
#@
#@@@@  WARNING
#@
#@     Reference..........: (LSLWAE ,    2)
#@
#@     Problem............: This is
#@                         How to
#@     Current action.....: Combine
#@                         the
#@     Comment............: Different
#@                         control
#@     Context............: Signs
#@                         at your
#@                         disposal.
#@-------------------------------------------------------------------------
#@
```

6.4 General Functions and Subroutines

ADIM	Return number of dimensions of array
ADIMVL	Return the *n*th dimension of array
AFTER	True if current time is after the event
ASIZE	Return size of an array
AT	True if current time is at the event
AUNDF	Return undefined position in array
AVG	Return average of two values
BEFORE	True if current time is before the event
CNDFNC	Conductance functions
EVALFR	Evaluate thermal frequency response between specified nodes
GRPAVE	Average value of an entity across all nodes in ZLABEL
GRPLST	Return list of all nodes in a group
GRPMAX	Maximum value of an entity across all nodes in ZLABEL
GRPMIN	Minimum value of an entity across all nodes in ZLABEL
GRPSUM	Sum of an entity across all nodes in ZLABEL
HEATER-DEFINE	Define a thermostatically controlled heater
HEATER-RESET	Zero the 'accumulated' output parameters of a heater
HEATER-STATUS	Return performance parameters for a heater
INTNOD	Return internal node number from user node number
MDLOFF	Turn model off
MDLON	Turn model on
NDMFL	Nodal flow rate function
NODFNC	Nodal functions
NODNUM	Return user node number from internal number
SETNDI	Set integer node entity values
SETNDR	Set real node entity values (e.g. temperature, capacitances, fluxes...)
SETNDZ	Set character node entity values (e.g. fluid type)
STATRP	Return entity status
STATST	Set entity status
STORMM	Store nodal entity min/max values
STRLNA	Active string length
SUBMDN	Return submodel name for a node
SUBMOD	Return submodel name of current model
TSINK	Sink temperature calculation
ACLOSS	Return contraction/expansion flow loss
COND	Return thermal conductivity
CP	Return specific heat capacity at constant pressure
ENTH	Return specific enthalpy
EVLQST	Set heat flux on evaporative link
FINITS	Fluid state initialisation
MTYPST	Set mass flow link type
PHSDST	Set direction of phase separation
PHSXST	Set vapour quality of phase separation

PLEBPI	Return pipe bend flow loss
PLNOZZ	Return nozzle flow loss
PLDIFF	Return diffusor flow loss
PLORIF	Return orifice flow loss
PLBUVA	Return butterfly valve flow loss
PLSLVA	Return slide valve flow loss
PROPS1	Fluid property data base (1 independent variable)
PROPS2	Fluid property data base (2 independent variables)
PROPS3	Fluid property data base (3 independent variables)
RHO	Return density
RLVALV	Set valve opening fraction
SETDP	Apply pressure source
SETDPV	Apply pressure source with linearisation
SHUM	Return specific humidity
SIG	Return surface tension
VLSTAT	Set valve ON/OFF
VISC	Return dynamic viscosity
VOLST	Set volume-change status
WVSINK	Set water vapour sink-rate

FTYPEC Convert fluid name to number
FTYPEI Convert fluid number to name

GETA Get nodal heat-transfer area (if fluid node) of area (if thermal node)
GETALP Get nodal solar absorptivity (thermal node only)
GETC Get nodal capacitance (both thermal and fluid)
GETCMP Get nodal compliance (fluid node only)
GETEPS Get nodal infra-red emissivity (thermal node only)
GETFD Get nodal hydraulic diameter (fluid node only)
GETFE Get nodal specific enthalpy (fluid node only)
GETFF Get nodal wall surface roughness (fluid node only)
GETFH Get nodal specific enthalpy of mass source (fluid node only)
GETFL Get nodal length (fluid node only)
GETFLA Get nodal flow area (fluid node only)
GETFM Get nodal mass source rate (fluid node only)
GETFQ Get nodal internal heat source (fluid node only)
GETFR Get nodal temperature of mass source (fluid node only)
GETFRG Get nodal predicted flow regime (fluid node only)
GETFST Get nodal fluid state descriptor (fluid node only)
GETFT Get nodal fluid type (fluid node only)
GETFW Get nodal vapour quality of mass source (fluid node only)
GETFX Get nodal cartesian coordinate X (both thermal and fluid nodes)
GETFY Get nodal cartesian coordinate Y (both thermal and fluid nodes)
GETFZ Get nodal cartesian coordinate Z (both thermal and fluid nodes)
GETL Get nodal label (both thermal and fluid)
GETP Get nodal static pressure (fluid node only)
GETPHI Get nodal relative humidity (fluid node only)

GETQA Get nodal albedo heat source (thermal node only)
GETQAI Get nodal incident albedo heat source (thermal node only)
GETQE Get nodal earth heat source (thermal node only)
GETQEI Get nodal incident earth heat source (thermal node only)
GETQI Get nodal internal heat source (thermal node only)
GETQR Get nodal other heat source (thermal node only)
GETQS Get nodal solar heat source (thermal node only)
GETQSI Get nodal incident solar heat source (thermal node only)
GETT Get nodal temperature (both thermal and fluid nodes)
GETVDT Get nodal rate of change of volume with respect to time (fluid node only)
GETVOL Get nodal volume (fluid node only)
GETVQ Get nodal vapour quality (fluid node only)

GETGF Get fluidic conductor value between two nodes
GETGF2 Get parallel fluidic conductor value between two nodes
GETGL Get linear conductor value between two nodes
GETGL2 Get parallel linear conductor value between two nodes
GETGP Get fluid conductor value between two nodes
GETGR Get radiative conductor value between two nodes
GETGR2 Get parallel radiative conductor value between two nodes
GETGV Get view factor value between two nodes
GETGV2 Get parallel view factor value between two nodes
GETM Get mass flow value between two nodes

SETA Set nodal heat-transfer area (if fluid node) of area (if thermal node)
SETALP Set nodal solar absorptivity (thermal node only)
SETC Set nodal capacitance (both thermal and fluid)
SETCMP Set nodal compliance (fluid node only)
SETEPS Set nodal infra-red emissivity (thermal node only)
SETFD Set nodal hydraulic diameter (fluid node only)
SETFE Set nodal specific enthalpy (fluid node only)
SETFF Set nodal wall surface roughness (fluid node only)
SETFH Set nodal specific enthalpy of mass source (fluid node only)
SETFL Set nodal length (fluid node only)
SETFLA Set nodal flow area (fluid node only)
SETFM Set nodal mass source rate (fluid node only)
SETFQ Set nodal internal heat source (fluid node only)
SETFR Set nodal temperature of mass source (fluid node only)
SETFRG Set nodal predicted flow regime (fluid node only)
SETFST Set nodal fluid state descriptor (fluid node only)
SETFT Set nodal fluid type (fluid node only)
SETFW Set nodal vapour quality of mass source (fluid node only)
SETFX Set nodal cartesian coordinate X (both thermal and fluid nodes)
SETFY Set nodal cartesian coordinate Y (both thermal and fluid nodes)
SETFZ Set nodal cartesian coordinate Z (both thermal and fluid nodes)
SETL Set nodal label (both thermal and fluid)

SETP Set nodal static pressure (fluid node only)
SETPHI Set nodal relative humidity (fluid node only)
SETQA Set nodal albedo heat source (thermal node only)
SETQAI Set nodal incident albedo heat source (thermal node only)
SETQE Set nodal earth heat source (thermal node only)
SETQEI Set nodal incident earth heat source (thermal node only)
SETQI Set nodal internal heat source (thermal node only)
SETQR Set nodal other heat source (thermal node only)
SETQS Set nodal solar heat source (thermal node only)
SETQSI Set nodal incident solar heat source (thermal node only)
SETT Set nodal temperature (both thermal and fluid nodes)
SETVDT Set nodal rate of change of volume with respect to time (fluid node only)
SETVOL Set nodal volume (fluid node only)
SETVQ Set nodal vapour quality (fluid node only)

SETGF Set fluidic conductor value between two nodes
SETGF2 Set parallel fluidic conductor value between two nodes
SETGL Set linear conductor value between two nodes
SETGL2 Set parallel linear conductor value between two nodes
SETGP Set fluid conductor value between two nodes
SETGR Set radiative conductor value between two nodes
SETGR2 Set parallel radiative conductor value between two nodes
SETGV Set view factor value between two nodes
SETGV2 Set parallel view factor value between two nodes
SETM Set mass flow value between two nodes

NDMFL Calculate nodal flow rate
XCOND Evaluate fluid thermal conductivity (single- or two-phase)
XCONDS ” (saturation)
XCP Evaluate fluid isobaric specific heat (single- or two-phase)
XCPS ” (saturation)
XENTH Evaluate fluid specific enthalpy (single- or two-phase)
XENTHS ” (saturation)
XJT Evaluate fluid Joule-Thompson coefficient (single- or two-phase)
XJTS ” (saturation)
XKT Evaluate fluid isothermal compressibility (single- or two-phase)
XKTS ” (saturation)
XPRES Evaluate fluid pressure (single- or two-phase)
XPRESS ” (saturation)
XRHO Evaluate fluid density (single- or two-phase)
XRHOS ” (saturation)
XSIG Evaluate fluid surface tension (single- or two-phase)
XSIGS ” (saturation)
XTEMP Evaluate fluid temperature (single- or two-phase)
XTEMPS ” (saturation)
XVISC Evaluate fluid dynamic viscosity (single- or two-phase)
XVISCS ” (saturation)

6.4.1 Contraction Flow Loss

Name:

FUNCTION **ACLOSS**(F[:CNAME:]NODE, F[:CNAME:]NODE)

F[:CNAME:]NODE - fluid node

Type:

Double precision.

Purpose:

Returns a fluid conductance value between the two nodes given. ACLOSS takes note of the current flow direction, and the relative diameters of the two nodes, in order to calculate either an expansion or contraction loss.

Example:

```
#
$CONDUCTORS
#
GP(1 , 2) = 1.2 ;
GP(2 , PUMP:1) = ACLOSS(F2 , F:PUMP:1) # Pump entry loss
#
```

Numerical Background:

Expansion:

The irreversible loss is calculated as:-

$$\Delta p = \frac{1}{2}\rho u_1^2\left(1-\frac{A_1}{A_2}\right)^2$$

where

A_1 = upstream node area

A_2 = downstream node area

u_1 = upstream velocity

ρ = fluid density

Thus the GP value is:-

$$GP = \frac{1}{\left(1-\frac{A_1}{A_2}\right)^2}$$

Contraction:

The irreversible loss is:-

$$\Delta p = \frac{1}{2} k_1 \rho u_1^2$$

hence:-

$$GP = \frac{1}{k_1}$$

where $k_1 = k\left(\frac{A_2}{A_1}\right)^2$ and k is dependent on the downstream/upstream diameter ratio such that:-

d_2/d_1	0.0	0.2	0.4	0.6	0.8	1.0
k	0.5	0.45	0.38	0.28	0.14	0.0

6.4.2 Array Dimensions

Name:

FUNCTION **ADIM** (ANAME)

ANAME - ESATAN array name

Type:

Integer.

Purpose:

To return the number of dimensions of an ESATAN array. For table arrays the number of dimensions refers to the number of independent variables.

Example:

Consider the two-dimensional array ARR(3,5),

```
NUMDIM = ADIM(ARR)
```

returns NUMDIM as 2.

6.4.3 Array Dimension Size

Name:

FUNCTION **ADIMVL** (ANAME , N)

ANAME	-	ESATAN array name
N	-	Integer Dimension of array

Type:

Integer.

Purpose:

To return the size of the Nth dimension of an array.

Example:

Consider the two-dimensional array ARR(3,5),

```
DIMSIZ = ADIMVL(ARR, 2)
```

returns DIMSIZ as 5.

Restrictions:

The value of N must be less than or equal to the number of dimensions declared for the array.

6.4.4 Total Size of an Array

Name:

FUNCTION **ASIZE** (ANAME)

ANAME - ESATAN array name

Type:

Integer.

Purpose:

To return the total size of an array in terms of the number of elements (Including undefined elements).

Example:

Consider the two-dimensional array ARR(3,5),

```
ARRSIZ = ASIZE(ARR)
```

returns ARRSIZ as 15.

Restrictions:

Should not be used with table arrays.

6.4.5 Undefined Array Value

Name:

FUNCTION **AUNDF** (ANAME)

ANAME - ESATAN array name

Type:

Integer.

Purpose:

This function returns the position in the array of the first element which has the undefined value. The position returned follows the Fortran convention in that the first dimension is incremented first. If no element has the undefined value, then zero is returned.

Example:

```
PUNDF = AUNDF(ARR)
```

Restrictions:

Should be used for integer and real arrays only.

6.4.6 Average of Two Values

Name:

FUNCTION **AVG** (VAL1 , VAL2)

VAL1,VAL2 - Double Precision values

Type:

Double Precision.

Purpose:

To return the average of the two values given as arguments.

Examples:

```
AVGNUM = AVG(1.5D0, 6.4D0)

TAVG = AVG(T1, T25) # average of node 1 and node 25 temp.
```

6.4.7 Events

Name:

FUNCTION **BEFORE** (Event , OCCUR)
FUNCTION **AT** (Event , OCCUR)
FUNCTION **AFTER** (Event , OCCUR)
FUNCTION **BTWEEN** (Event1, Event2, OCCUR)

Event	-	Event
Event1	-	Event
Event2	-	Event
OCCUR	-	Integer occurrence number

Type:

Logical.

Purpose:

To return true or false dependent on the relationship of the current analysis time to the specified event.

The function BEFORE returns true if the current analysis time is earlier than the event time. For a periodic event the occurrence number indicates which occurrence of the event is to be used for comparison. For simple, non-periodic events, OCCUR is ignored.

The function AT returns true if the current analysis time is the event time. For a periodic event the occurrence number indicates which occurrence of the event is to be used for comparison, with the special case 0 meaning that the function returns true if the current time matches any occurrence of the event. For simple, non-periodic events, OCCUR is ignored.

The function AFTER returns true if the current analysis time is later than the event time. For a periodic event the occurrence number indicates which occurrence of the event is to be used for comparison. For simple, non-periodic events, OCCUR is ignored.

The function BTWEEN returns true if the current analysis time is between the two event times (this is a closed interval - the function returns true if the current analysis time coincides with either event time). For a periodic event the occurrence number indicates which occurrence of the events is to be used for comparison, with the special case 0 meaning that the function returns true if the current time matches any occurrence of the events. For simple, non-periodic events, OCCUR is ignored.

An event is at the current analysis time from the start of execution of $VARIABLES2 at the end of the time step which finishes at the event time, until the end of execution of the $VARIABLES1 call at the start of the next time step (which starts at the event time).

Example:

```
$EVENTS
#
$TIMESTEP
My_event = 100.0;           # simple event at 100 seconds
StartEvent = 50.0,100.0;    # periodic event at 50, 150, 250, ...
                            # seconds
EndEvent = 65.0,100.0;
#
$OUTPUT
OutputTrigger = 125.0;      # event at 125 seconds causing $OUTPUTS
                            # to be run
...
$VARIABLES1
      IF(AT(My_event,0))QI10 = 50.0
...
      IF(BTWEEN(StartEvent,EndEvent,2))THEN
#        between 150 and 165 seconds
...
```

6.4.8 Conductance Functions

Name:

CNDFNC (TYPE, ANAME|USRFNC[, N])

TYPE	-	Integer Required function type, 1 – 5
ANAME	-	ESATAN array (TYPE = 1, 3, 4)
USRFNC	-	Double precision User function (TYPE = 2, 5)
N	-	Integer Order of interpolation (TYPE = 1)

Purpose:

CNDFNC provides a shorthand way of defining linear conductances, GL($n1$, $n2$). Such conductances usually depend on the temperatures of the two nodes which are linked. This function removes the need for the user to supply the nodal temperature references explicitly. The value returned is normally an average conductivity.

CNDFNC is not actually a function in its own right, but a reference to it in the $CONDUCTORS block is translated by the preprocessor into a call to an appropriate underlying function.

The available function types, specified by TYPE, are:-

1 Lagrangian interpolation of order N on data points in the array ANAME at the average nodal temperature, (T$n1$ + T$n2$) / 2.

2 Evaluation of user function USRFNC at the average nodal temperature, (T$n1$ + T$n2$) / 2.

3 Trapezoidal integration of data points in the array ANAME between nodal temperatures T$n1$ and T$n2$, then division by T$n2$ –T$n1$.

4 Analytical integration of a polynomial between nodal temperatures T$n1$ and T$n2$, then division by T$n2$ – T$n1$. The polynomial coefficients are held in the array ANAME, in increasing order of degree (i.e. constant term first).

5 Integration by Simpson's rule of user function USRFNC between nodal temperatures T$n1$ and T$n2$, then division by T$n2$ –T$n1$.

For TYPE = 1 or 3, ANAME may be either a 2×n real array, or a table array with one independent variable, containing (temperature, conductivity) pairs. For TYPE = 4, ANAME must be a 1-dimensional real array.

For TYPE = 2 or 5, the user function USRFNC may be defined in the $SUBROUTINES block or may be contained in an external library. In either case, the function must take one double-precision argument (temperature), and the user must declare the function as EXTERNAL at the start of the $VARIABLES1 block in each submodel that calls it.

All temperatures are in units consistent with the control constant TABS, i.e. degrees Celsius if TABS=273.15 or kelvins if TABS=0.0.

The underlying double-precision functions and their arguments are, in the obvious order, as follows:-

CNDFN1(T*n*1, T*n*2, ANAME, N)
CNDFN2(T*n*1, T*n*2, USRFNC)
CNDFN3(T*n*1, T*n*2, ANAME)
CNDFN4(T*n*1, T*n*2, ANAME)
CNDFN5(T*n*1, T*n*2, USRFNC)

Examples:

Type = 1:
A conductor definition

```
GL(3,4)=INTRP1((T3+T4)/2.0,CONDX,1)*CSA/DX;
```

where `CONDX` is an array of conductivity vs. temperature, could be replaced by:-

```
GL(3,4)=CNDFNC(1,CONDX,1)*CSA/DX;
```

Type = 2:
A conductor definition

```
GL(3,4)=MYFUNC((T3+T4)/2.0)*CSA/DX;
```

could be replaced by

```
GL(3,4)=CNDFNC(2,MYFUNC)*CSA/DX;
```

Type = 3:
A conductor definition

```
GL(3,4)=(INTGL1(T3,T4,MYARR)/(T4-T3))*CSA/DX;
```

could be replaced by

```
GL(3,4)=CNDFNC(3,MYARR)*CSA/DX;
```

Type = 4:
Define a polynomial $f(x) = a_0 + a_1x + \ldots + a_nx^n$ where the coefficients $a_0, \ldots, a_n$ are stored in MYARR.

The conductor definition

```
GL(3,4)=CNDFNC(4,MYARR)*CSA/DX;
```

integrates $f(x)$ over the range defined by T3 and T4 and then divides by the temperature interval to give a conductivity which can be multiplied by the geometric factor `CSA/DX`.

Type = 5:
A conductor definition

```
GL(3,4)=CNDFNC(5,MYFUNC)*CSA/DX;
```

performs integration by Simpson's rule on MYFUNC prior to division by the temperature interval to give a conductivity which can be multiplied by the geometric factor `CSA/DX`. The number of intervals used for Simpson is progressively increased until convergence of the calculated conductivity value is achieved.

Restrictions:

CNDFNC may only be used within GL definitions in the $CONDUCTORS block. If it is required to set conductances in this way elsewhere the underlying functions listed above must be used.

The user function USRFNC must be declared EXTERNAL in $VARIABLES1.

6.4.9 Nodal Functions

Name:

NODFNC(TYPE, ANAME|USRFNC[, N])

TYPE	-	Integer Required function type
ANAME	-	ESATAN array (TYPE = 1)
USRFNC	-	Double precision User function (TYPE = 2)
N	-	Integer Order of interpolation (TYPE = 1)

Purpose:

NODFNC provides a shorthand way of defining nodal entities which depend on temperature, such as capacitance or emissivity. This function removes the need for the user to supply the nodal temperature reference explicitly.

NODFNC is not actually a function in its own right, but a reference to it in the $NODES block is translated by the preprocessor into a call to an appropriate underlying function.

The available function types, specified by TYPE, are:-

1 Lagrangian interpolation of order N on data points in the array ANAME at the nodal temperature.

2 Evaluation of user function USRFNC at the nodal temperature.

For TYPE = 1, ANAME may be either a $2\times n$ real array, or a table array with one independent variable, containing (temperature, property) pairs.

For TYPE = 2, the user function USRFNC may be defined in the $SUBROUTINES block or may be contained in an external library. In either case, the function must take one double-precision argument (temperature), and the user must declare the function as EXTERNAL at the start of the $VARIABLES1 block in each submodel that calls it.

All temperatures are in units consistent with the control constant TABS, i.e. degrees Celsius if TABS=273.15 or kelvins if TABS=0.0.

The underlying double-precision functions and their arguments are, in the obvious order, as follows:-

NODFN1(T*n*, ANAME, N)
NODFN2(T*n*, USRFNC)

where T*n* is the nodal temperature.

Examples:

Type = 1:
A node definition

```
D23, T=26.67, C=RHOX*VOLNOD*INTRP1(T23, SPECHT, 1);
```

where `SPECHT` is an array of specific heat vs. temperature, could be replaced by:-

```
D23, T=26.67, C=RHOX*VOLNOD*NODFNC(1, SPECHT, 1);
```

Type = 2:
A node definition

```
D23, T=26.67, C=RHOX*VOLNOD*MYFUNC(T23);
```

could be replaced by:-

```
D23, T=26.67, C=RHOX*VOLNOD*NODFNC(2, MYFUNC);
```

Restrictions:

NODFNC may only be called in a node definition in the $NODES block. If it is required to set nodal entities in this way in an operations block, the underlying functions listed above must be used.

The user function USRFNC must be declared EXTERNAL in $VARIABLES1.

6.4.10 Node Group Functions

Name:

FUNCTION **GRPMIN**(ZLABEL, ZENT, CNAME)
FUNCTION **GRPMAX**(ZLABEL, ZENT, CNAME)
FUNCTION **GRPSUM**(ZLABEL, ZENT, CNAME)
FUNCTION **GRPAVE**(ZLABEL, ZENT, CNAME)

ZLABEL	-	Character group of nodes (see Section 6.1.3)
ZENT	-	Character node entity
CNAME	-	Concatenated submodel name (see Section 6.1.3)

Type:

Double precision.

Purpose:

These routines return the minimum, maximum, sum and average respectively of the entity ZENT for all the nodes in the node group given by ZLABEL.

Example:

```
$VARIABLES2
      DOUBLE PRECISION QSUM, TAVE
      TAVE = GRPAVE('Instrument', 'T', sub1:ONLY)
      QSUM = GRPSUM('#1-5;sub1:1-10', 'QI', CURRENT)
```

Restrictions:

None.

6.4.11 List of Nodes in a Group

Name:

SUBROUTINE **GRPLST**(ZLABEL, CNAME, NODLST, N)

ZLABEL	-	Character group of nodes (see Section 6.1.3)
CNAME	-	Concatenated model name (see Section 6.1.3)
NODLST	-	Integer ESATAN array
N	-	Integer

Purpose:

GRPLST stores in the one-dimensional ESATAN array NODLST the internal node numbers of the nodes referenced by ZLABEL. The number of nodes identified is returned as N.

Example:

```
$CONSTANTS
$CHARACTER
#
# Group of fluid nodes
#
GROUP1 = '#1,3;SUB1:4-6;SUB2:3,4';
$ARRAYS
$INTEGER
GP1LST(15) = U;
...
$INITIAL
      INTEGER I, NUM
C
      CALL GRPLST(GROUP1, CURRENT, GP1LST, NUM)
C
C Print out density of all nodes in GROUP1 to standard output
C
      DO 100, I = 1, NUM
         WRITE(*,*) 'Density of Node ', NODNUM(GP1LST(I),
     &            ' in submodel ', SUBMDN(GP1LST(I), 'ALL'),
                  ' = ', RHO(GP1LST(I))
  100 CONTINUE
```

Restrictions:

NODLST must have been declared of sufficient length to contain the node list. If NODLST is inadequate - being either too short or of the wrong type, then an error message is produced and N is returned as 0.

6.4.12 Fluid Properties

Name:

FUNCTION **COND**(F[:CNAME:]NODE)
FUNCTION **CP**(F[:CNAME:]NODE)
FUNCTION **ENTH**(F[:CNAME:]NODE)
FUNCTION **RHO**(F[:CNAME:]NODE)
FUNCTION **SHUM**(F[:CNAME:]NODE)
FUNCTION **SIG**(F[:CNAME:]NODE)
FUNCTION **VISC**(F[:CNAME:]NODE)

F[:CNAME:]NODE - Fluid node reference

Type:

Double precision.

Purpose:

Fluid property values may be referenced through the above library functions and may be called from any where in the operations blocks and also anywhere in the data blocks where Mortran expressions are allowed. If user properties are defined for the fluid then these functions will return these values.

COND	Returns thermal conductivity
CP	Returns specific heat capacity at constant pressure
ENTH	Returns specific enthalpy
RHO	Returns density
SHUM	Returns specific humidity
SIG	Returns surface tension
VISC	Returns viscosity

It should be noted that the node reference should be in the form 'F*n*', irrespective of the fluid node status.

Example:

Calculation of the heat input along a pipe which is divided into two fluid nodes, numbers 1 and 2.

```
$VARIABLES2
      DOUBLE PRECISION CPAVE
# Note HT12 defined as a user constant
# Average specific heat capacity - CPAVE
      CPAVE = (CP(F1) + CP(F2)) / 2.0D0
      HT12 = CPAVE * M(1 , 2) * (T2 - T1)


      DENSTY = RHO(F:PUMP1:1)
```

Restrictions:

These functions can only be used with fluid nodes. SHUM is applicable to air/water-vapour (fluid type AIRW) only.

6.4.13 Fluid State Initialisation

Name:

SUBROUTINE **FINITS**(ZLABEL, CNAME)

ZLABEL - Character Node specifier (Section 6.1.3)
CNAME - Submodel name (Section 6.1.3)

Purpose:

Initialisation of state variables at fluid nodes of ZLABEL in submodel CNAME.

FINITS ensures consistency of the nodal entities P, T, FE and VQ—pressure, temperature, enthalpy and vapour quality. Depending on the fluid state descriptor at each node, FST, two variables will be taken as given and the other two calculated accordingly.

Flow area (if undefined) and volume are also computed, using the nodal length and hydraulic diameter.

Note that the preprocessor automatically inserts a call to FINITS for the current submodel only in each $INITIAL block.

Example:

```
#
$EXECUTION
      INTEGER I
#
# Read starting conditions for GIZMO from external file
#
      DO 100, I = 1, 10
          READ(21, *) T:GIZMO:1(I-1), VQ:GIZMO:1(I-1)
 100  CONTINUE
#
# Force consistent initialisation
#
      CALL FINITS(' ', GIZMO:ONLY)
#
```

6.4.14 Sub-Model Status

Name:

SUBROUTINE **MDLOFF** (CNAME)
SUBROUTINE **MDLON** (CNAME)

CNAME - Sub-model name (see Section 6.1.3)

Purpose:

MDLOFF defines sub-model CNAME as outside the solution domain by turning off the nodes in that sub-model. MDLON defines sub-model CNAME as within the solution domain. Previously inactive nodes and conductors remain as such.

Example:

```
$EXECUTION
      CALL SOLVIT
C
      CALL MDLON(SUBM2)
      CALL MDLOFF(SUBM1)
C
      CALL SLFWBK
```

Restrictions:

The user should note that a call to subroutine MDLOFF will not turn off a conductor defined within CNAME if both of its attached nodes are contained in sub-models outside CNAME. MDLON and MDLOFF cannot be used for submodels containing fluid nodes.

6.4.15 Internal Node Number from User Node Number

Name:

FUNCTION **INTNOD**(CNAME, NODE)

CNAME	-	Concatenated submodel name
NODE	-	Integer User node number

Type:

Integer.

Purpose:

INTNOD returns the internal node number, given the user defined submodel name and node number NODE as input. The returned value will lie between 1 and NNT + NNF, where the thermal nodes are numbered sequentially from 1 to NNT, and any fluid nodes are numbered sequentially from NNT + 1 to NNT + NNF.

Example:

```
INODE = INTNOD(CURRENT, 100)
```

returns the internal node number of node 100 in the current submodel.

```
INODE = INTNOD(SUB1:SUB2, 1)
```

returns the internal node number of node 1 (the first node) in submodel SUB2 of submodel SUB1.

Restrictions:

If the value of CNAME or NODE is invalid, an error message is output. It is therefore recommended that INTNOD should not be directly included in a WRITE statement, since on some platforms attempting to write out an error message when the function is itself part of a WRITE statement causes the program to crash.

6.4.16 User Node Number from Internal Node Number

Name:

FUNCTION **NODNUM**(NODE)

NODE - Integer Internal node number

Type:

Integer.

Purpose:

NODNUM returns the user supplied node number, given the internal node number NODE as input. NODE must lie between 1 and NNT + NNF, where the thermal nodes are internally numbered sequentially from 1 to NNT, and any fluid nodes are internally numbered sequentially from NNT + 1 to NNT + NNF.

NODNUM can be used in conjunction with SUBMDN to provide the submodel and user node reference for an internal node number.

Example:

```
UNODE = NODNUM(NRLXCC)
```

returns the user node number of the node at which the maximum relaxation change is found.

Restrictions:

NODE must lie between 1 and NNT + NNF.

If the value of NODE is invalid, an error message is output. It is therefore recommended that NODNUM should not be directly included in a WRITE statement, since on some platforms attempting to write out an error message when the function is itself part of a WRITE statement causes the program to crash.

6.4.17 Submodel Name from Internal Node Number

Name:

FUNCTION **SUBMDN**(NODE, ATTRIB)

NODE	-	Integer Internal node number
ATTRIB	-	Character flag for presenting the results

Type:

Character.

Purpose:

SUBMDN returns the submodel name, given the internal node number NODE as input. NODE must lie between 1 and NNT + NNF, where the thermal nodes are internally numbered sequentially from 1 to NNT, and any fluid nodes are internally numbered sequentially from NNT + 1 to NNT + NNF.

ATTRIB can take one of the following values: 'ALL', 'ROOT' or 'SUBMODEL'. 'ALL' implies that the full concatenated submodel name will be output. 'ROOT' returns the concatenated submodel name of the parent of the given submodel, and 'SUBMODEL' returns just the submodel name. Setting ATTRIB to the empty string defaults to the 'ALL' option.

SUBMDN can be used in conjunction with NODNUM to provide the submodel and user node reference for an internal node number.

Examples:

Given the submodel hierarchy MAIN:SUB1:SUB2, and assuming that CSG is minimal for a node in SUB2

```
#
$VARIABLES2
#
        CHARACTER *20 SUBM
#
# to return 'MAIN:SUB1:SUB2'
#
        SUBM = SUBMDN(NCSGMN, 'ALL')
#
# to return 'MAIN:SUB1'
#
        SUBM = SUBMDN(NCSGMN, 'ROOT')
#
# to return 'SUB2'
#
        SUBM = SUBMDN(NCSGMN, 'SUBMODEL')
```

Restrictions:

NODE must lie between 1 and NNT + NNF. This function will return a character string up to 256 characters.

If the value of NODE or ATTRIB is invalid, then the string returned by SUBMDN is an error message.

6.4.18 Submodel Name of Current Model

Name:

FUNCTION **SUBMOD**(ATTRIB)

ATTRIB - Character flag for presenting the results

Type:

Character.

Purpose:

SUBMOD returns the submodel name of the current (sub)model.

ATTRIB can take one of the following values: 'ALL', 'ROOT' or 'SUBMODEL'. 'ALL' implies that the full concatenated submodel name will be output. 'ROOT' returns the concatenated submodel name of the parent of the given submodel, and 'SUBMODEL' returns just the submodel name. Setting ATTRIB to the empty string defaults to the 'ALL' option.

Examples:

Given the submodel hierarchy MAIN:SUB1:SUB2, then

```
#
$INITIAL
#
        CHARACTER * 20 SUBM
#
# to return 'MAIN:SUB1:SUB2'
#
        SUBM = SUBMOD('ALL')
#
# to return 'MAIN:SUB1'
#
        SUBM = SUBMOD('ROOT')
#
# to return 'SUB2'
#
        SUBM = SUBMOD('SUBMODEL')
#
```

Restrictions:

This function will return a character string up to 256 characters.

If the value of ATTRIB is invalid, then the string returned by SUBMOD is an error message.

6.4.19 Active String Length

Name:

FUNCTION **STRLNA**(STRING)

STRING - Character string

Type:

Integer.

Purpose:

STRLNA returns the active length of a string - i.e., the position of the last non-blank character in the string.

Example:

Given SUBM being 'MAIN:SUB1:SUB2', then

```
STRLEN = STRLNA(SUBM)
```

returns 14.

6.4.20 Pipe Bend Flow Loss

Name:

FUNCTION **PLBEPI**(SR , DIA , RAD , ANGLE , F[:CNAME:]NODE)

SR	-	Double precision Roughness of pipewall
DIA	-	Double precision Pipe diameter
RAD	-	Double precision Radius of bend
ANGLE	-	Double precision Angle of bend in degrees
F[:CNAME:]NODE	-	Fluid node (upstream)

Type:

Double precision.

Purpose:

The function PLBEPI computes the pressure loss coefficient for a circular pipe bend.

Example:

```
#
$CONDUCTORS
#
GP(10,11) = PLBEPI(FF10, FD10, 0.002D0, 60.0D0, F10)
#
```

Numerical Background:

The range of validity of the function is for an angle from 11.25 ° to 90 ° and also for bend/radius ratio between 1 and 10. The computation procedure is based upon determination of friction coefficients for straight pipes and two-dimensional table interpolation. The node number is used internally to obtain the average flow rate and viscosity of the node in order to evaluate the Reynolds number; the upstream node is appropriate for this.

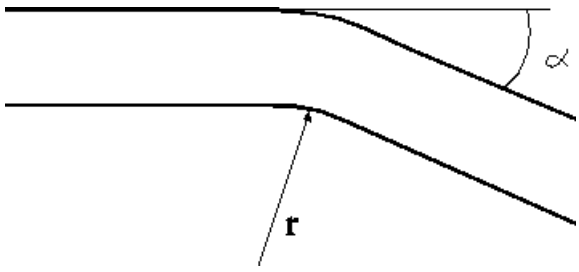


Figure 6-1. Pipe bend geometry

$$\xi = \lambda \cdot \frac{l}{d} + \xi_0$$

where

λ = friction factor

$$\xi_0 = f\left(\alpha, \frac{r}{d}\right)$$

$$\xi = \frac{1}{GP}$$

d = pipe diameter

r = bend radius

l = length of bead = $\frac{\pi r \alpha}{180}$

α	11.25°	22.50°	30.0°	45.0°	60.0°	90.0°
	ξ_0					
r/d 1.0	0.030	0.045	0.050	0.140	0.190	0.210
2.0	0.030	0.045	0.050	0.090	0.120	0.140
4.0	0.030	0.045	0.050	0.080	0.100	0.110
6.0	0.030	0.045	0.045	0.075	0.090	0.090
10.0	0.030	0.045	0.045	0.070	0.070	0.011

6.4.21 Nozzle Flow Loss

Name:

FUNCTION **PLNOZZ**(SR , DMAX , DMIN , LEN , F[:CNAME:]NODE)

SR	-	Double precision Roughness of pipe wall
DMAX	-	Double precision Pipe maximum diameter
DMIN	-	Double precision Pipe minimum diameter
LEN	-	Double precision Length of nozzle
F[:CNAME:]NODE	-	Fluid node (upstream)

Type:

Double precision.

Purpose:

The function PLNOZZ computes the pressure loss coefficient for a circular nozzle with converging angles.

Example:

```
#
$CONDUCTORS
#
GP(10,11) = PLNOZZ(FF10, FD10, FD11, 0.01D0, F10);
#
```

Numerical Background:

PLNOZZ computes the pressure loss coefficient for a circular nozzle with converging angles from 0 ° to 180 ° (see diagram for definition of the angle). The computation procedure is based upon determination of friction coefficients for straight pipes and formula evaluation to incorporate the influence of the nozzle[7]. The node number is used internally to obtain the average flow rate and viscosity of the node in order to evaluate the Reynolds number; the upstream node is appropriate for this.

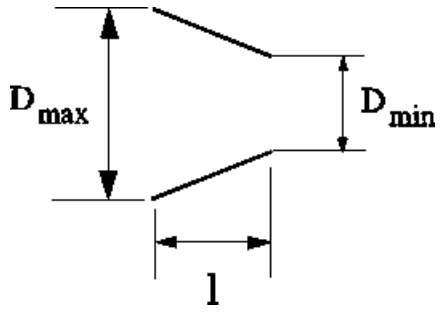


Figure 6-2. Nozzle geometry

$$\xi = (-0.0125n^4 + 0.0224n^3 - 0.00723n^2 + 0.0044n - 0.00745)(\alpha_r^3 - 2\pi\alpha_r^2 - 10\alpha_r) + \lambda\frac{l}{D_{\max}}$$

where

$$n = \left(\frac{D_{\min}}{D_{\max}}\right)^2$$

$$\alpha_r = 2\,\mathrm{atan}\left(\frac{D_{\max} - D_{\min}}{2l}\right)$$

λ = friction factor

$$\xi = \frac{1}{GP}$$

6.4.22 Diffusor Flow Loss

Name:

FUNCTION **PLDIFF**(SR , DMAX , DMIN , LEN , F[:CNAME:]NODE)

SR	-	Double precision Roughness of pipe wall
DMAX	-	Double precision Pipe maximum diameter
DMIN	-	Double precision Pipe minimum diameter
LEN	-	Double precision Length of diffusor cone
F[:CNAME:]NODE	-	Fluid node (upstream)

Type:

Double precision.

Purpose:

The function PLDIFF computes the pressure loss coefficient for a circular diffusor.

Example:

```
#
$CONDUCTORS
#
GP(10,11) = PLDIFF(FF10, FD11, FD10, 0.01D0, F10);
#
```

Numerical Background:

PLDIFF computes the pressure loss coefficient for a circular diffusor with an open angle from 3° to 180° and the square of the ratio of minimum pipe diameter to maximum pipe diameter from 0.0 to 0.6. The computation procedure is based upon determination of friction coefficients for straight pipes, formulae evaluation and two dimensional table interpolation. The node number is used internally to obtain the average flow rate and viscosity of the node in order to evaluate the Reynolds number; the upstream node is appropriate for this.

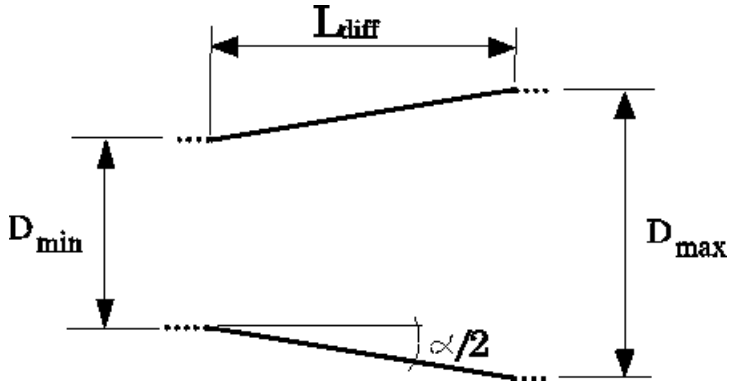


Figure 6-3. Diffusor geometry

$$\xi = \lambda \cdot \left(\frac{1 - \frac{F_{\min}}{F_{\max}}}{8 \sin \frac{\alpha}{2}} \right) + \xi_0$$

where

λ = friction factor

$$F = \frac{\pi D^2}{4}$$

$$\xi_0 = f\left(\alpha, \frac{F_{\min}}{F_{\max}}\right)$$

$$\xi = \frac{1}{GP}$$

α	3.0°	10.0°	20.0°	30.0°	40.0°	60.0°	90.0°	180.0°
				ξ_0				
$\frac{F_{\min}}{F_{\max}}$								
0.0	0.03	0.15	0.36	0.65	0.92	1.15	1.10	1.02
0.05	0.03	0.14	0.32	0.58	0.83	1.04	0.99	0.92
0.1	0.03	0.12	0.29	0.52	0.75	0.93	0.89	0.83
0.2	0.02	0.10	0.23	0.41	0.59	0.74	0.70	0.65
0.3	0.02	0.07	0.18	0.31	0.40	0.57	0.54	0.50
0.4	0.01	0.06	0.13	0.23	0.33	0.41	0.39	0.37
0.5	0.01	0.04	0.09	0.16	0.23	0.29	0.28	0.26
0.6	0.01	0.03	0.06	0.10	0.15	0.18	0.17	0.16

6.4.23 Orifice Flow Loss

Name:

FUNCTION **PLORIF**(SR , PD , PL , OD , OL , F[:CNAME:]NODE)

SR	-	Double precision Roughness of pipewall
PD	-	Double precision Pipe diameter
PL	-	Double precision Pipe length
OD	-	Double precision Orifice diameter
OL	-	Double precision Orifice length
F[:CNAME:]NODE	-	Fluid node (upstream)

Type:

Double precision.

Purpose:

The function PLORIF computes the pressure loss coefficient for a circular orifice with bevelled edges facing the flow.

Example:

```
#
$CONDUCTORS
#
GP(10,11) = PLORIF(1.0D-5, FD10, FL10+FL11, 3.0D-3, 5.0D-3, F10);
#
```

Numerical Background:

The bevelled edges of the orifice are assumed to be at an angle of 40-60 °, and the ratio of orifice length to orifice diameter in the range 0.01-0.16. The computation procedure is based upon determination of friction coefficients for straight pipes, formula evaluation and one-dimensional table interpolation. The node number is used internally to obtain the average flow rate and viscosity of the node in order to evaluate the Reynolds number; the upstream node is appropriate for this.

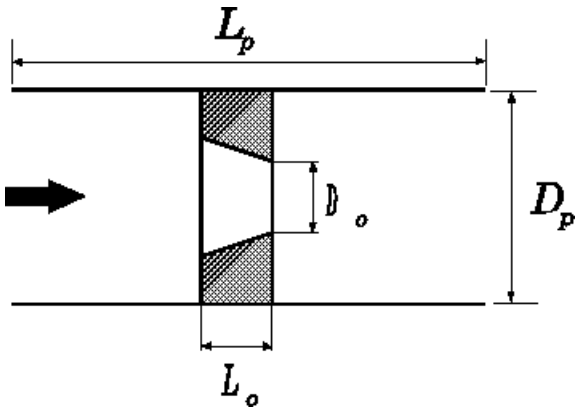


Figure 6-4. Orifice geometry

$$\xi = \left(1 + \sqrt{\xi'\left(1 - \frac{F_o}{F_p}\right)} - \frac{F_o}{F_p}\right)^2 \cdot \left(\frac{F_p}{F_o}\right)^2 + \lambda \frac{L_p}{D_p}$$

where

$$\xi = \frac{1}{GP}$$

$$F = \frac{\pi D^2}{4}$$

λ = friction factor

$$\xi' = f\left(\frac{L_o}{D_o}\right)$$

ξ' is obtained by interpolation from the following table:

$\frac{L_0}{D_0}$	0.01	0.04	0.08	0.16
ξ'	0.46	0.35	0.23	0.13

6.4.24 Butterfly Valve Loss

Name:

FUNCTION **PLBUVA**(DIA , ANGLE , KV , F[:CNAME:]NODE)

DIA	-	Double precision Diameter of valve
ANGLE	-	Double precision Valve angle (degrees)
KV	-	Double precision Leakage value
F[:CNAME:]NODE	-	Fluid node (upstream)

Type:

Double precision.

Purpose:

The function PLBUVA computes the pressure loss coefficient for a circular butterfly valve.

Example:

```
#
$CONDUCTORS
#
GP(10,11) = PLBUVA(FD10, 30.0D0, 0.97D0, F10);
#
```

Numerical Background:

The valve opening angle may be from 0° (fully open) to 90° (fully closed). The ratio between valve flap diameter and inner pipe diameter may be between 0.1 and 0.99; this ratio is called leakage value, KV. If values outside this range are defined then KV is set to the appropriate limiting value. The computation procedure is based upon formula evaluation. The node number is used internally to obtain the average flow rate and viscosity of the node in order to evaluate the Reynolds number; the upstream node is appropriate for this.

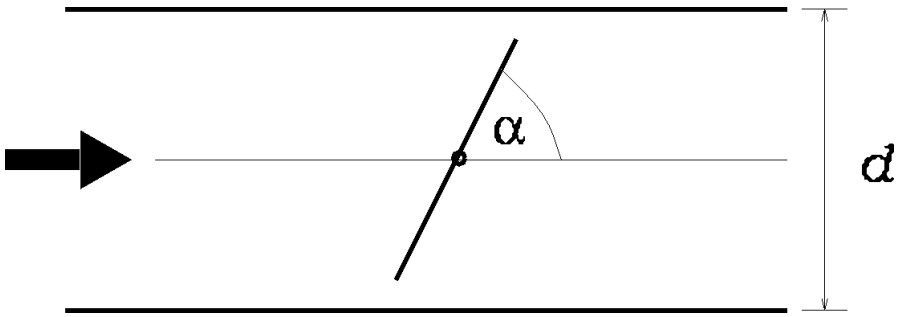


Figure 6-5. Butterfly valve geometry

$$\xi = \frac{120}{R_e} \cdot \frac{1 + (\overline{D}/2)(1 + \sin\alpha)}{(1 - \overline{D}^2 \sin\alpha)^2} + \left(1 - \frac{50}{R_e}\right)\left(\frac{1.56}{1 - \overline{D}^2 \sin\alpha}\right)$$

where:

$$\overline{D} = D_{\text{shutter}}/d < 1$$

$$\xi = \frac{1}{GP}$$

6.4.25 Slide Valve Flow Loss

Name:

FUNCTION **PLSLVA**(OPEN)

OPEN - Double precision Ratio of opening height to valve diameter

Type:

Double precision.

Purpose:

The function PLSLVA computes the pressure loss coefficient for a circular slide valve with the ratio open high of the shutter to the valve diameter (0 equals fully closed, 1 equals fully open).

Example:

```
#
$CONDUCTORS
#
GP(10,11) = PLSLVA(0.3D0);
#
```

Numerical Background:

The computation procedure is based upon table interpolation.

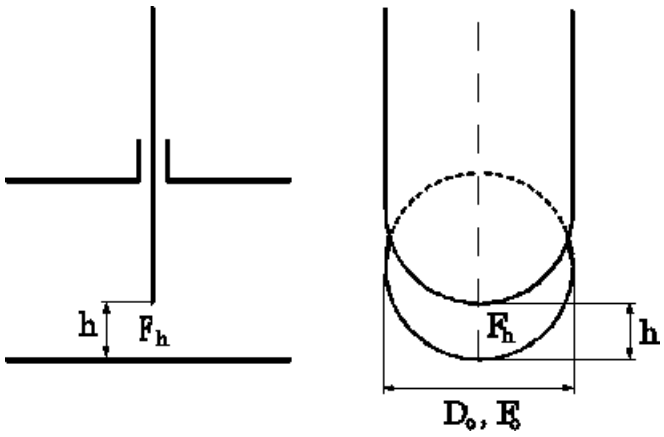


Figure 6-6. Slide valve geometry

F_0 is the total cross sectional area and is given by the diameter D_0. F_h denotes the actual open area.

$$\xi = f\left(\frac{h}{D_0}\right) = \tilde{f}\left(\frac{F_h}{F_0}\right)$$

where

$$\xi = \frac{1}{GP}$$

The following table is used for interpolation to obtain ξ.

h/D_0	0.0	0.125	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00
F_h/F_0	0.0	0.16	0.25	0.38	0.50	0.61	0.71	0.81	0.90	0.96	1.00
ξ	10^4	97.8	35.0	10.0	4.60	2.06	0.98	0.44	0.17	0.06	0.00

6.4.26 Fluid Property Database Access

Name:

FUNCTION **PROPS1**(FTYPE, PROP, DP1, ZERTS)
FUNCTION **PROPS2**(FTYPE, PROP, DP1, DP2, ZERTS)
FUNCTION **PROPS3**(FTYPE, PROP, DP1, DP2, DP3, ZERTS)

FTYPE	-	Character: Fluid type
PROP	-	Character: Property to be calculated
DP1	-	Double precision: Independent variable
DP2	-	Double precision: Independent variable
DP3	-	Double precision: Independent variable
ZERTS	-	Character: Description of independent variables

Type:

Double precision

Purpose:

These functions give the user direct access to the FHTS property routines without the constraint that the function arguments must pertain to conditions at a particular fluid node.

The user stipulates the fluid type (FTYPE), the property to be computed (PROP) and the fluid state. The state is defined by specifying the values of a sufficient number of independent thermodynamic variables as to be unambiguous; the values are passed as DP1, DP2 and DP3, their meaning being denoted by the character string ZERTS.

PROPS1 returns saturation temperature for a given pressure (PROP='TSAT', ZERTS='P'), or saturation pressure for a given temperature (PROP='PSAT', ZERTS='T').

PROPS2 can be used to calculate any of the following list of properties:-

- density (PROP='RHO')
- specific heat capacity (PROP='CP')
- viscosity (PROP='VISC')
- thermal conductivity (PROP='COND')
- surface tension (PROP='SIG')*
- specific enthalpy (PROP='ENTH')*
- vapour quality (PROP='QUAL')*
- temperature (PROP='TEMP')

*not for fluid types AIR and AIRW

The fluid state is specified using ZERTS in the same manner as for the nodal entity FST:-

'P&T'	-	pressure and temperature (single-phase)
'P&FE'	-	pressure and enthalpy
'P&VQ'	-	pressure and vapour quality (two-phase)
'T&VQ'	-	temperature and vapour quality (two-phase)

PROPS3 calculates the properties RHO, CP, VISC and COND for moist air (fluid type AIRW) only, this special case arising from the need to specify AIRW's state with three independent variables: pressure, temperature and relative humidity (ZERTS='P&T&PHI').

Note that DP1, DP2 and DP3 (where applicable) correspond to the first, second and third variables, respectively, listed in ZERTS, in whatever order they may be given.

The reason for the multiplicity of functions is to avoid the necessity of defining dummy values for any unused variables.

Example:

```
$VARIABLES2
C
C Calculate density of water at room
C temperature and pressure
C
      DENS = PROPS2('WATER', 'RHO', 20.D0, 1.0D5, 'T&P')
C
```

Restrictions:

PROPS*n* cannot be called from user-defined property functions, as this would lead to recursive calls which are explicitly forbidden in Fortran 77.

6.4.27 Valve Opening Fraction

Name:

SUBROUTINE **RLVALV** (DXVDT, CNAME, TDB, TSEN, TSET,
XVMAX, XVMIN)
SUBROUTINE **VLSTAT** (CNAME, VSTAT)

DXVDT	-	Double precision Rate of change of valve velocity w.r.t. temperature
CNAME	-	Sub-model name
TDB	-	Double precision Deadband temperature
TSEN	-	Double precision Sensor temperature
TSET	-	Double precision Set point temperature
XVMAX	-	Double precision Maximum valve velocity
XVMIN	-	Double precision Minimum valve velocity
VSTAT	-	Character Valve status - ON/OFF

Purpose:

These routines are used to control the setting of the valve open fraction for the standard FHTS element VALVE.

RLVALV sets the valve opening fraction calculated from the valve characteristic data supplied using the Rate Limiting valve method. For a three way valve, after the valve position for the main branch valve has been calculated from the equations, the other branch valve position is calculated as one minus the main branch fraction.

VLSTAT sets the valve opening fraction for ON/OFF states. The user defines the state of the valve VSTAT and the routine sets the fraction to 1 for ON or 0 for OFF. Again for the case of a three-way valve, the second branch opening fraction is calculated as one minus the main branch fraction.

Examples:

```
$VARIABLES2
         DOUBLE PRECISION DXVDT , TDB , TSEN , TSET ,
        &                                XVMAX , XVMIN
C Valve VALV
C Set xopen in rlvalv
C User input valve operating conditions
         DXVDT=0.002
         TDB=0.01
         TSEN=T7
         TSET=T7
         XVMAX= 0.004
         XVMIN=0.0001
C
         CALL RLVALV(DXVDT , VALV , TDB , TSEN , TSET ,
        &                                 XVMAX , XVMIN )
```

```
        $VARIABLES2
C Valve VALV
              CALL VLSTAT(VALV, 'ON')
```

Restrictions:

These routines can only be used with the standard FHTS element VALVE.

RLVALV can only be used for transient calculations. For steady state calculations, the substitutions data XOPEN should be set explicitly in the element definition.

The valve opening fraction must lie between zero and one if set by the user.

6.4.28 Pressure Source Assignment

Name:

FUNCTION **SETDP**(ZENT, VAL1)
FUNCTION **SETDPV**(ZENT, VAL1, VAL2)

ZENT	-	Character: mass flow link descriptor
VAL1	-	Double precision: pressure source
VAL2	-	Double precision: derivative of pressure source w.r.t. volumetric flow rate

Type:

Double precision.

Purpose:

These functions are used to assign a pressure source (Δp) to the specified mass flow link.

The format for ZENT is:

'M[:CNAME:]([CNAME:]*n*1, [CNAME:]*n*2)'

*n*1 and *n*2 being node numbers. A positive value for VAL1 will result in a pressure *increase* over the link.

If the derivative of Δp with respect to volumetric flow rate is known—for instance, the gradient of a pump characteristic—then this may be supplied with SETDPV. The effect of this is to aid convergence and stability of the solution by linearisation of the Δp term in the momentum equation.

Example:

```
#
$VARIABLES1
      DOUBLE PRECISION DPS, DPDVS
#
         DPS   = (RHOJ / RHO_REF) * (SPEEDR ** 2) * DPREF
         DPDVS = (RHOJ / RHO_REF) * SPEEDR * DPDVR
#
# Assign pressure source
#
         CALL SETDPV('M(1,2)' , DPS , DPDVS)
#
```

Restrictions:

Accepts mass flow links only.

6.4.29 Set node entity values

Name:

SUBROUTINE **SETNDI** (ZLABEL, ZENT, IVALUE, CNAME)
SUBROUTINE **SETNDR** (ZLABEL, ZENT, RVALUE, CNAME)
SUBROUTINE **SETNDZ** (ZLABEL, ZENT, ZVALUE, CNAME)

ZLABEL	-	Character: node label (see Section 6.1.3)
ZENT	-	Character: node entity (e.g. 'T', 'QA')
IVALUE	-	Integer: value to assign to node entity
RVALUE	-	Real: value to assign to node entity
ZVALUE	-	Character: value to assign to node entity
CNAME	-	Concatenated submodel name (see Section 6.1.3)

Purpose:

Sets the value of a node entity for all nodes matching ZLABEL and CNAME. The appropriate routine to call depends on the type of the entity: SETNDI for integer, SETNDR for real, or SETNDZ for character. The routines apply to any node entity, including user-defined ones.

Example:

```
CALL SETNDI('Hot Side', 'Cstate', 5, BIGBOX:TOP:ONLY)
CALL SETNDR(' ', 'T', 20.0D0, CURRENT)
CALL SETNDZ(' ', 'FT', 'WATER', CURRENT)
```

Here, Cstate is a user-defined entity.

6.4.30 Status Reporting/Setting

Name:

SUBROUTINE **STATST** (ZENT, ZSTAT)
FUNCTION **STATRP** (ZENT)

ZENT - Character string, entity descriptor
ZSTAT - Character string, status

Type:

STATRP: Character * 3

Purpose:

STATST is used to set the status of any entity (node or conductor) to ZSTAT.

The possible formats for ZENT are:

N[:CNAME:]*n* for a node;

GL|GR|GF|GV|M[:CNAME:]([CNAME:]*n*1, [CNAME:]*n*2[, *x*])
for a conductor;

M[:CNAME:] for a sub-model (STATRP only).

where *n, n*1 and *n*2 are node numbers and *x* is the sequence number of the conductor.

STATRP returns the existing status of an entity (node, conductor or sub-model). The status of a conductor or a sub-model may be either **ON** or **OFF**. The status of a node is **D**, **B** or **X** for a solid node, **F, J, R, K** or **H** for a fluid node. A node or conductor status, is for this purpose, irrespective of the status of the submodel in which it is contained.

Example:

```
      CHARACTER FLAG*3
C
      CALL STATST('N:SUBM1:23', 'B')
      CALL STATST('GL(2,SUBM1:5)', 'ON')
      CALL SETNDZ(' ', 'FT', 'WATER', CURRENT)
C
      FLAG = STATRP('N15')
      FLAG = STATRP('M:SUBM1:')
      FLAG = STATRP('M')
```

Restrictions:

The following restrictions apply:-

- The status of a sub-model may not be set in STATST (for defining a sub-model's status see Section 6.4.14).

- In STATRP, ZENT should be set to 'M' if the status of the current sub-model is required. 'M:CURRENT:' is NOT permitted.
- In STATST, ZSTAT must be either 'ON' or 'OFF' for the conductors and either 'D', 'B' or 'X' for solid nodes, 'F', 'J', 'R', 'K' or 'H' for fluid nodes.
- No error is recorded if any attempt is made to set an entity status to its existing status.

6.4.31 Store Nodal Min/Max Values

Name:

SUBROUTINE **STORMM**(ZENT, ENTMIN, TIMMIN, ENTMAX, TIMMAX)

ZENT - Character: nodal entity for which min/max are required
ENTMIN - Character: user-defined nodal entity to store minimum value
TIMMIN - Character: user-defined nodal entity to store time of minimum
ENTMAX - Character: user-defined nodal entity to store maximum value
TIMMAX - Character: user-defined nodal entity to store time of maximum

Purpose:

This routine is used to store the minimum and maximum values attained at each node in a model by a specified nodal entity during a transient analysis.

When called, the routine takes each node in turn and compares the current value of the entity given by ZENT with the values of those given by ENTMIN and ENTMAX. If the entity named in ZENT is less than that named in ENTMIN, then the latter is set to the value of the former and the current solution time (i.e. the value of the control constant TIMEN) is recorded in the entity given by TIMMIN. Similarly, if the entity named in ZENT is greater than that named in ENTMAX, then the latter is set to the value of the former and the current solution time is recorded in the entity given by TIMMAX.

Note that user-defined nodal entities are used for storage: ENTMIN, TIMMIN, ENTMAX and TIMMAX must contain names which have been declared in the global file (see Section 3.8.1). ZENT may be either an existing ESATAN entity name or a user-defined one.

STORMM should be called from one or both of the $VARIABLES2 and $OUTPUTS blocks. If the call is included in $VARIABLES2 then the calculation will be made at the end of every time step, whereas if called in $OUTPUTS it will be made before the first timestep and then only after every output interval (OUTINT). Note that if only $VARIABLES2 is used, then initial values which are maxima or minima will not be caught; if only $OUTPUTS is used, then the sampling of results will be coarse and the true maximum and minimum values over the transient may not be caught. For this reason it is recommended that STORMM is normally called in both $VARIABLES2 and $OUTPUTS. If maxima and minima are to be recorded only for a given time interval this can be achieved using appropriate logic in the operations block.

The entities in ENTMIN and ENTMAX need to be initialised appropriately before STORMM is called. This can be achieved by using one of the library routines SETNDR and SETNDI (Section 6.4.29), either in the $INITIAL block or in the $EXECUTION block prior to the solver being called.

The maximum and minimum values stored can be printed out using any output routine that supports user-defined nodal entities.

Note that where the maximum or minimum value is reached more than once, it is the time of the first occurrence that is recorded.

Example:

In the global file

```
$USER_NODE_ENTITIES
$REAL
MaxT;
MinT;
TimeMaxT;
TimeMinT;
MaxQI;
MinQI;
TimeMaxQI;
TimeMinQI;
```

In the model file:

```
$INITIAL
      CALL SETNDR(' ', 'MinT', 1.0D10, CURRENT)
      CALL SETNDR(' ', 'MaxT', -1.0D10, CURRENT)
      CALL SETNDR(' ', 'MinQI', 1.0D10, CURRENT)
      CALL SETNDR(' ', 'MaxQI', -1.0D10, CURRENT)
$OUTPUTS
      CALL STORMM('T','MinT','TimeMinT','MaxT','TimeMaxT')
$VARIABLES2
      CALL STORMM('T','MinT','TimeMinT','MaxT','TimeMaxT')
#
# Record QI min/max only for 1000.0 <= t <= 2000.0
#
      IF ((TIMEN .GE. 1.0D3) .AND. (TIMEN .LE. 2.0D3)) THEN
         CALL STORMM('QI','MinQI','TimeMinQI',
     &                'MaxQI','TimeMaxQI')
      END IF
$EXECUTION
      CALL SLFWBK  # Transient solution
      CALL PRNDTB(' ', 'MinT,TimeMinT,MaxT,TimeMaxT', CURRENT)
      CALL PRNDTB(' ', 'MinQI,TimeMinQI,MaxQI,TimeMaxQI',CURRENT
```

Restrictions:

The specified user-defined nodal entities must have been declared inthe global file (see Section 3.8.1) and their values initialised before STORMM is first called.

ZENT must name a real or integer nodal entity; character entities are not permitted.

Tip:

To print a table of maximum and minimum temperatures attained during a transient compared to hot and cold limits, define two further user entities to hold the limits and use PRNDTB to print out the results. For example:

```
CALL PRNDTB(' ', 'ColdLimit, MinT, MaxT, HotLimit', CURRENT)
```

6.4.32 Sink Temperature

Name:

FUNCTION **TSINK** (ZLABEL1, CNAME1, ZLABEL2, CNAME2, TYPE, ZERROR)

ZLABEL1	-	Character Thermal group (see Section 6.1.3)
CNAME1	-	Submodel name for thermal group (see Section 6.1.3)
ZLABEL2	-	Character Environment group (see Section 6.1.3)
CNAME2	-	Submodel name for environment group (see Section 6.1.3)
TYPE	-	Integer Type of sink temperature required
ZERROR	-	Integer Error code

Type:

Double precision

Purpose:

To calculate the sink temperature between a node (or group of nodes) of ZLABEL1 to an environment defined by a node (or group of nodes) of ZLABEL2. TYPE will define which kind of sink temperature calculation is to be used. It has to be one of the four following integers:-

1 Black body radiation sink temperature $T_{S,bbr}$
2 Grey body radiation sink temperature $T_{S,gbr}$
3 Radiative sink temperature $T_{S,r}$
4 Linear sink temperature $T_{S,l}$

The calculated value may be undefined according to the following error code ZERROR:-

0 No error
1 Undefined: wrong TYPE specified
2 Undefined: fluid/boundary node defined as input
3 Undefined: thermal and environment groups are not disjoint
4 Undefined: complex value obtained
5 Undefined: radiative conductances zero
6 Undefined: linear conductances zero

Example:

```
$VARIABLES2
C
C Calculation of the radiative sink temperature for the
C thermal group composed of nodes from 10 to 20 to the
C environment group composed of nodes from 50 to 60
C
      TS = TSINK('#10-20', CURRENT, '#50-60', CURRENT, 3,
     &           ZERROR)
C
      IF (ZERROR .EQ. 0) THEN
      . . .
         TS = TSINK('#SUB1:10-20', CURRENT,
                '#SUB2:50-60', CURRENT, 3, ZERROR)
      END IF
C
```

Restrictions:

Accepts thermal nodes only. Ignores conductors attached to inactive nodes in calculation. Boundary nodes can only be part of the environment group, not the thermal group. Requires 2 * NUMREF1 + 2 * NUMREF2 locations of dynamic core, where NUMREF1 is the number of nodes of ZLABEL1 and NUMREF2 is the number of nodes of ZLABEL2.

Numerical Background:

The **Black body radiation sink temperature $T_{S,bbr}$** of the thermal item (node i) with respect to the environment (group of nodes E) is defined by:

$$\sigma\varepsilon_i A_i(T_i^4 - T_{S,\,bbr}^4) = \sum_{j \in E} [\sigma GR_{ij}(T_i^4 - T_j^4)] - (QS_i + QA_i + QE_i)$$

This equation can easily be generalised by summing i over the total number of nodes in the thermal item.

The **Grey body radiation sink temperature $T_{S,gbr}$** of the thermal item (node i) with respect to the environment (group of nodes E) is defined by:

$$\left(\sum_{j \in E} [\sigma GR_{ij}]\right)(T_i^4 - T_{S,\,gbr}^4) = \sum_{j \in E} [\sigma GR_{ij}(T_i^4 - T_j^4)] - (QS_i + QA_i + QE_i)$$

This equation can easily be generalised by summing i over the total number of nodes in the thermal item. Note that the grey body radiation sink temperature of the thermal item is independent of any temperature Ti of this thermal item.

The **Radiative sink temperature $T_{S,r}$** of the thermal item (node i) with respect to the environment (group of nodes E) is defined by:

$$\left(\sum_{j \in E} [\sigma GR_{ij}] \right) (T_i^4 - T_{S,r}^4) = \sum_{j \in E} [\sigma GR_{ij} (T_i^4 - T_j^4)]$$

This equation can easily be generalised by summing i over the total number of nodes in the thermal item.

The **Linear sink temperature $T_{S,l}$** of the thermal item (node i) with respect to the environment (group of nodes E) is defined by:

$$\left(\sum_{j \in E} GL_{ij} \right) (T_i - T_{S,l}) = \sum_{j \in E} [GL_{ij} (T_i - T_j)]$$

This equation can easily be generalised by summing i over the total number of nodes in the thermal item.

6.4.33 Volume-Change Status

Name:

SUBROUTINE **VOLST** (F*n* , ZSTAT)

F*n*	-	Fluid node reference
ZSTAT	-	Character string, status

Purpose:

To set volume-change status of a fluid node. This controls whether the node's flow area (ZSTAT = 'A') or its length (ZSTAT = 'L') is to be updated during solution if a change in volume is calculated. At the start of solution the former is assumed. Change in volume is determined by the nodal entities CMP (compliance) and VDT (time-derivative of volume); See Section 3.5.2

Example:

```
CALL VOLST(F10, 'A')
CALL VOLST(F:SUBM1:4, 'L')
```

6.4.34 Water Vapour Sink

Name:

SUBROUTINE **WVSINK** (F*n* , *r-exp*)

F*n*	-	Fluid node reference
r-exp	-	Double precision expression

Purpose:

To set sink rate for water vapour at an air/vapour node. Causes pure water vapour to be extracted during solution at a rate of *r-exp* kg/s; cf. defining a sink via the nodal entity FM, which would remove a mixture of air and water-vapour.

Example:

```
CALL WVSINK(F10, 0.001D0)
CALL WVSINK(F:SUBM1:5, 0.002D0)
```

Restrictions:

Can only be used in conjunction with solution routine FLTNSS. Applicable to nodes of fluid type AIRW only. *r-exp* must be non-negative.

6.4.35 Phase Separation

Name:

SUBROUTINE **PHSDST** (ZENT, ZPHSD)
SUBROUTINE **PHSXST** (ZENT, ZPHSD, VAPQ)

ZENT	-	Character: Mass flow link descriptor
ZPHSD	-	Character: Phase separation direction
VAPQ	-	Double precision: Phase separation direction vapour quality

Purpose:

PHSDST activates phase separation in a given direction for a mass flow link (see Section 4.6). The direction specified by ZPHSD may be:

forward	('FWD')
reverse	('REV')
both	('BOTH')
none	('NONE')

The nominal forward direction is given by the definition of the mass flow link from the first node to the second node. Reverse flow occurs when the flow goes from the second to the first node.

The default value for a momentum link is 'NONE', i.e. no phase separation.

PHSXST sets vapour quality of the phase separation: liquid (0.0), vapour (1.0) or two-phase (0.0–1.0). Thus, if VAPQ = 0.0 and ZPHSD = 'FWD' then only liquid will be allowed to pass through the link (unless the upstream node is pure vapour).

Example:

In this example, link M(1, 2) will allow only liquid to pass in the forward direction, but will allow both liquid and vapour to pass backwards. Link M(5, 9) will accept a 50% mixture of liquid and vapour for forward flow, but only pure vapour for reverse.

```
$CONDUCTORS
#
M(1, 2) = 0.0 ;
M(9, 9) = 0.0 ;
#
:
:
$INITIAL
      CALL PHSDST('M(1, 2)', 'FWD')
      CALL PHSXST('M(1, 2)', 'FWD', 0.0D0)
      CALL PHSDST('M(5, 9)', 'BOTH')
      CALL PHSXST('M(5, 9)', 'FWD', 0.5D0)
      CALL PHSXST('M(5, 9)', 'REV', 1.0D0)
```

Restrictions:

The specified mass flow link must be a momentum type (see Section 6.4.36). PHSDST has to be called before PHSXST with the same direction specified.

Can only be used in conjunction with solution routine FGENSS or FGENFI.

6.4.36 Mass flow link type

Name:

SUBROUTINE **MTYPST** (ZENT, ZMFLT)

ZENT	-	Character string, Mass flow link descriptor
ZMFLT	-	Character string, Mass flow link type

Purpose:

MTYPST sets a given mass flow link as an evaporative link (’EVAP’) or a momentum link (’MOM’). A momentum link obeys the normal conservation of momentum equation. An evaporative link (Section 4.5) induces a flow rate according to an imposed heat flux, for instance as in a capillary evaporator. The evaporative heat flux is applied via subroutine EVLQST (Section 6.4.37).

Mass flow links are of type ’MOM’ by default.

Example:

```
$CONDUCTORS
#
M(1, 2) = 0.0 ;
M(3, 4) = 0.0 ;
#
$INTIAL
#
      CALL MTYPST(’M(1, 2)’, ’MOM’)
#
      CALL MTYPST(’M(3, 4)’, ’EVAP’)
#
```

Restrictions:

Can only be used in conjunction with solution routine FGENSS or FGENFI.

6.4.37 Evaporative link heat flux

Name:

SUBROUTINE **EVLQST** (ZENT, HF)

ZENT	-	Character string, Mass flow link descriptor
HF	-	Double precision, Heat flux

Purpose:

EVLQST sets the heat flux on an evaporative mass flow link (Section 4.5), for modelling evaporative pumping (e.g. in a capillary evaporator).

Example:

```
$CONDUCTORS
#
M(1, 2) = 0.0 ;
M(3, 4) = 0.0 ;
#
$INTIAL
#
      DOUBLE PRECISION HF
      HF = 200.0D0
      CALL MTYPST('M(1, 2)', 'EVAP')
      CALL EVLQST('M(1, 2)', 100.0D0)
      CALL MTYPST('M(3, 4)', 'EVAP')
      CALL EVLQST('M(3, 4)', HF)
```

Restrictions:

The specified mass flow link must be of evaporative type. This means that subroutine MTYPST (Section 6.4.36) must be called before EVLQST and the mass flow link set to evaporative.

Can only be used in conjunction with solution routine FGENSS or FGENFI.

6.4.38 Nodal flow rate

Name:

FUNCTION **NDMFL**(NODE)

NODE	-	Fluid node reference

Type:

Double precision.

Purpose:

NDMFL calculates the nodal flow rate in the specified node by averaging the flow into and the flow out of the fluid node.

It should be noted that the node reference should be in the form 'F*n*', irrespective of the fluid node status.

Example:

```
$VARIABLES2
      DOUBLE PRECISION NDM
#
      NDM = NDMFL(F:PUMP1:1)
```

Restrictions:

This function can only be used with fluid nodes.

6.4.39 Fluid Property Interface

Name:

FUNCTION **XCOND**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XCONDS**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XCP**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XCPS**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XENTH**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XENTHS**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XJT**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XJTS**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XKT**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XKTS**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XPRES**(TEMP, ENTH, QUAL, FTI, NODE)
FUNCTION **XPRESS**(TEMP, FTI, NODE)
FUNCTION **XRHO**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XRHOS**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XSIG**(PRES, TEMP, FTI, NODE)
FUNCTION **XSIGS**(PRES, TEMP, FTI, NODE)
FUNCTION **XTEMP**(PRES, ENTH, QUAL, FTI, NODE)
FUNCTION **XTEMPS**(PRES, FTI, NODE)
FUNCTION **XVISC**(PRES, TEMP, QUAL, FTI, NODE)
FUNCTION **XVISCS**(PRES, TEMP, QUAL, FTI, NODE)

PRES	-	Double precision: Pressure (model units)
TEMP	-	Double precision: Temperature (model units)
ENTH	-	Double precision: Enthalpy
QUAL	-	Double precision: Vapour quality
FTI	-	Integer: Fluid-type number
NODE	-	Node reference (or zero)

Type:

Double precision

Purpose:

These functions allow evaluation of fluid properties at an arbitrary fluid state. The properties referenced are those defined in the currently selected library (Section 4.2.2).

The quantity returned by each function is as follows:

XCOND	-	Thermal conductivity (single- or two-phase)
XCONDS	-	” (saturation)
XCP	-	Isobaric specific heat (single- or two-phase)
XCPS	-	” (saturation)

XENTH	-	Specific enthalpy (single- or two-phase)
XENTHS	-	” (saturation)
XJT	-	Joule-Thompson* coefficient (single- or two-phase)
XJTS	-	” (saturation)
XKT	-	Isothermal compressibility (single- or two-phase)
XKTS	-	” (saturation)
XPRES	-	Pressure (single- or two-phase)
XPRESS	-	” (saturation)
XRHO	-	Density (single- or two-phase)
XRHOS	-	” (saturation)
XSIG	-	Surface tension (single- or two-phase)
XSIGS	-	” (saturation)
XTEMP	-	Temperature (single- or two-phase)
XTEMPS	-	” (saturation)
XVISC	-	Dynamic viscosity (single- or two-phase)
XVISCS	-	” (saturation)

Note that the properties are calculated from the given fluid state, not simply returned as the current value at a specified node. Vapour quality, QUAL, should be set to zero for subcooled or saturated liquid, and to unity for saturated or superheated vapour.

Input values of pressure and temperature, and the return values of XPRES/ XPRESS and XTEMP/XTEMPS, are in model units, i.e. on the scale determined by the control constants `PABS` and `TABS`.

The fluid-type number, FTI, is obtained from the fluid-type name via the function FTYPEI (Section 6.4.40). The user should not set FTI directly, as the fluid number will vary depending on which fluid property library is being used.

The node reference, NODE, is passed to the functions purely for information, e.g. for use in error messages. If it is unknown or not relevant, this parameter should be set to zero.

Note that, for saturation properties, all input state variables should be consistent with the saturation state. For example, if saturated liquid viscosity is required at a certain pressure, then the corresponding saturation *temperature* should be calculated first and passed in to XVISCS.

* Or Joule-Kelvin

Example:

```
$VARIABLES2
#
      DOUBLE PRECISION DENS, MUL, SPH, TS
      INTEGER FTI
#
# Calculate properties of water:-
#
      FTI = FTYPEI('WATER')
#
# - Density at room temperature & pressure (liquid)
#   25 deg C, 1.01325D5 Pa, 0 %
#
      DENS = XRHO(1.01325D5, 25.0D0, 0.0D0, FTI, 0)
#
# - Specific heat of vapour at pressure of node F3 and
#   250 deg C
#
      SPH = XCP(P3, 250.0D0, 1.0D0, FTI, F3)
#
# - Saturation temperature at F3
#
      TS = XTEMPS(P3, FTI, F3)
#
# - Saturated liquid viscosity at pressure of F3
#
      MUL = XVISCS(P3, TS, 0.0D0, FTI, F3)   # Note T_s used
#
```

Restrictions:

In order to be evaluated, a fluid property must have been defined in the current fluid library.

6.4.40 Fluid Type Mapping

Name:

FUNCTION **FTYPEC**(FTI)
FUNCTION **FTYPEI**(FTC)

FTI	-	Integer: Fluid-type number
FTC	-	Character: Fluid-type name

Type:

Character
Integer

Purpose:

These are utility functions to be used in conjunction with the fluid property interface functions (Section 6.4.39). They allow conversion between the fluid-type name specified in a model and the corresponding number used internally. (Denoting the fluid type by an integer value is more efficient in terms of CPU-time than using a character string when several properties are to be evaluated for the same fluid.)

FTYPEC returns the fluid-type name associated with the given number.

FTYPEI returns the fluid-type number associated with the given name.

Note that the mapping between fluid names and numbers will vary depending on which fluid property library is being used (Section 4.2.2). The actual value of a fluid number should not be of concern to the user.

Example:

```
$SUBROUTINES
#
# Compute density:
#   - WATER: at room temperature & pressure
#   - Other: at STP
#
      SUBROUTINE DENSY(FTI, DENS)
#
      CHARACTER FTC * 24
      DOUBLE PRECISION DENS
      INTEGER FTI
#
      FTC = FTYPEC(FTI)   # Convert fluid number to name
#
      IF (FTC .EQ. 'WATER') THEN
         DENS = XRHO(1.01325D5, 25.0D0, 0.0D0, FTI, 0)
      ELSE
         DENS = XRHO(1.01325D5, 0.0D0, 0.0D0, FTI, 0)
      END IF
#
```

```
      END
#
$EXECUTION
      DOUBLE PRECISION DENS
      INTEGER FTI
#
      FTI = FTYPEI('AMMONIA') # Convert fluid name to number
#
      CALL DENSY(FTI, DENS)   # User-defined subroutine
#
```

Restrictions:

In order for the mapping to exist, at least one property for a given fluid must have been defined in the current fluid library.

6.4.41 Get Nodal Entity

Name:

FUNCTION **GETA**(NODE)
FUNCTION **GETALP**(NODE)
FUNCTION **GETC**(NODE)
FUNCTION **GETCMP**(NODE)
FUNCTION **GETEPS**(NODE)
FUNCTION **GETFD**(NODE)
FUNCTION **GETFE**(NODE)
FUNCTION **GETFF**(NODE)
FUNCTION **GETFH**(NODE)
FUNCTION **GETFL**(NODE)
FUNCTION **GETFLA**(NODE)
FUNCTION **GETFM**(NODE)
FUNCTION **GETFQ**(NODE)
FUNCTION **GETFR**(NODE)
FUNCTION **GETFRG**(NODE)
FUNCTION **GETFST**(NODE)
FUNCTION **GETFT**(NODE)
FUNCTION **GETFW**(NODE)
FUNCTION **GETFX**(NODE)
FUNCTION **GETFY**(NODE)
FUNCTION **GETFZ**(NODE)
FUNCTION **GETL** (NODE)
FUNCTION **GETP**(NODE)
FUNCTION **GETPHI**(NODE)
FUNCTION **GETQA**(NODE)
FUNCTION **GETQAI**(NODE)
FUNCTION **GETQE**(NODE)
FUNCTION **GETQEI**(NODE)
FUNCTION **GETQI**(NODE)
FUNCTION **GETQR**(NODE)
FUNCTION **GETQS**(NODE)
FUNCTION **GETQSI**(NODE)
FUNCTION **GETT**(NODE)
FUNCTION **GETVDT**(NODE)
FUNCTION **GETVOL**(NODE)
FUNCTION **GETVQ**(NODE)

NODE - Node reference

Type:

Double precision
Character

Purpose:

Return the current value of an entity at a given node. The node reference is either the internal node number or the node identifier (e.g. Dn).

Access to ESATAN nodal entities is provided for use in situations requiring a calculation for an arbitrary node, such as when looping over a list of nodes or in a subroutine where a node reference is passed in as a parameter.

The functions are named in the obvious way from the nodal entity concerned.

Example:

```
...
$CONDUCTORS
GL(1, 101) = MYHTC(F1, D101);
...
$SUBROUTINES
      DOUBLE PRECISION FUNCTION MYHTC(FNODE, TNODE)
#
# Calculate heat-transfer coefficient using Mungo correlation
#
      DOUBLE PRECISION CND, DIA, FLAREA, HTAREA, HTC, MFLOW, PR,
     &                 PRESS, QUALT, RE, SPHT, TEMP, TWALL, VSC
      INTEGER FTI, FNODE, TNODE
      CHARACTER FTYPE * 24
#
# Get fluid node entities and flow rate
#
      DIA = GETFD(FNODE)     # Hydraulic diameter
      FTYPE = GETFT(FNODE)   # Fluid type
      PRESS = GETP(FNODE)    # Pressure
      QUALT = GETVQ(FNODE)   # Vapour quality
      TEMP = GETT(FNODE)    # Temperature
      FLAREA = GETFLA(FNODE) # Flow area
      HTAREA = GETA(FNODE)  # Heat-transfer area
      MFLOW = NDMFL(FNODE)   # Mass flow rate
#
# Get wall temperature
#
      TWALL = GETT(TNODE)
#
# Calculate fluid properties
#
      FTI = FTYPEI(FTYPE)
      VSC = XVISC(PRESS, TEMP, QUALT, FTI, FNODE)
      SPHT = XCP(PRESS, TEMP, QUALT, FTI, FNODE)
      CND = XCOND(PRESS, TEMP, QUALT, FTI, FNODE)
#
# Reynolds & Prandtl numbers
```

```
#
      RE = MFLOW * DIA / (FLAREA * VSC)
      PR = VSC * SPHT / CND
#
# Mungo correlation
#
      HTC = 0.5D0 * (RE ** 0.2D0) * (PR ** 0.6D0) * CND / DIA
            * (ABS(TWALL - TEMP) ** 0.03D0)
#
# Multiply by heat-transfer area to get conductance
#
      MYHTC = HTC * HTAREA
#
      END
```

Restrictions:

The type of the node specified must be appropriate to the entity being requested; i.e. a fluid nodal entity value will not be returned for a thermal node, and vice versa.

User defined nodal entities are currently not supported.

6.4.42 Get Control Constant

Name:

FUNCTION **GETCCR**(CCR)

CCR - Character: Real control constant name

Type:

Double precision

Purpose:

Return the current value of a real control constant.

Access to a limited number of control constants is provided primarily for use in external Fortran and LANG = FORTRAN subroutines.

This function was developed to facilitate the calculation of user-defined heat-transfer coefficients; hence, the control constants supported are restricted to the following:

GRAVX, GRAVY, GRAVZ
PABS
TABS

Example:

Suppose the user has defined an external function to calculate the density of an ideal gas, in the file uigrho.f:

```
      DOUBLE PRECISION FUNCTION UIGRHO(PRES, TEMP, RGAS)
C
C Calculate density of an ideal gas.
C Need to convert input pressure & temperature from
C model units to absolute
C
      DOUBLE PRECISION GETCCR, PABS, PRES, RGAS, TABS, TEMP
C
      PABS = GETCCR('PABS')
      TABS = GETCCR('TABS')
C
      UIGRHO = (PRES + PABS) / (RGAS * (TEMP + TABS))
C
      END
```

Restrictions:

Not all control constants are currently supported.

6.4.43 Get Conductor Value

Name:

FUNCTION **GETGF**(CNAME, NODE1, NODE2)
FUNCTION **GETGF2**(CNAME, NODE1, NODE2, CSQ)
FUNCTION **GETGL**(CNAME, NODE1, NODE2)
FUNCTION **GETGL2**(CNAME, NODE1, NODE2, CSQ)
FUNCTION **GETGP**(CNAME, NODE1, NODE2)
FUNCTION **GETGR**(CNAME, NODE1, NODE2)
FUNCTION **GETGR2**(CNAME, NODE1, NODE2, CSQ)
FUNCTION **GETGV**(CNAME, NODE1, NODE2)
FUNCTION **GETGV2**(CNAME, NODE1, NODE2, CSQ)
FUNCTION **GETM**(CNAME, NODE1, NODE2)

CNAME	-	Concatenated submodel name
NODE1	-	Node1 reference
NODE2	-	Node2 reference
CSQ	-	Conductor sequence number

Type:

Double precision

Purpose:

Return the current value of the conductor defined between the two given nodes, given the submodel where this conductor has been defined and, optionally given its sequence number. The node reference is either the internal node number or the node identifier (e.g. Dn).

Access to conductors values is provided for use in situations requiring a calculation for an arbitrary node, such as when looping over a list of conductors or in a subroutine where a node reference is passed in as a parameter.

The functions are named in the obvious way from the type of conductor concerned.

The suffix "2" indicates that the function allows users to address a specific sequence number. Basic functions assume CSQ = 1.

Example:

```
...
$CONDUCTORS
GL(1, 101) = 1.23456;
GL(1, 101) = 0.00444;
...
$INITIAL
```

```
INTEGER N1
DOUBLE PRECISION MYGL1, MYGL2

N1 = INTNOD(CURRENT, 1)

MYGL1 = GETGL (CURRENT,N1,D101)
MYGL2 = GETGL2(CURRENT,N1,D101,2)
```

Restrictions: -

6.4.44 Set Nodal Entity

Name:

SUBROUTINE **SETA**(NODE, VAL)
SUBROUTINE **SETALP**(NODE, VAL)
SUBROUTINE **SETC**(NODE, VAL)
SUBROUTINE **SETCMP**(NODE, VAL)
SUBROUTINE **SETEPS**(NODE, VAL)
SUBROUTINE **SETFD**(NODE, VAL)
SUBROUTINE **SETFE**(NODE, VAL)
SUBROUTINE **SETFF**(NODE, VAL)
SUBROUTINE **SETFH**(NODE, VAL)
SUBROUTINE **SETFL**(NODE, VAL)
SUBROUTINE **SETFLA**(NODE, VAL)
SUBROUTINE **SETFM**(NODE, VAL)
SUBROUTINE **SETFQ**(NODE, VAL)
SUBROUTINE **SETFR**(NODE, VAL)
SUBROUTINE **SETFRG**(NODE, CHAR)
SUBROUTINE **SETFST**(NODE, CHAR)
SUBROUTINE **SETFT**(NODE, CHAR)
SUBROUTINE **SETFW**(NODE, VAL)
SUBROUTINE **SETFX**(NODE, VAL)
SUBROUTINE **SETFY**(NODE, VAL)
SUBROUTINE **SETFZ**(NODE, VAL)
SUBROUTINE **SETL** (NODE, CHAR)
SUBROUTINE **SETP**(NODE, VAL)
SUBROUTINE **SETPHI**(NODE, VAL)
SUBROUTINE **SETQA**(NODE, VAL)
SUBROUTINE **SETQAI**(NODE, VAL)
SUBROUTINE **SETQE**(NODE, VAL)
SUBROUTINE **SETQEI**(NODE, VAL)
SUBROUTINE **SETQI**(NODE, VAL)
SUBROUTINE **SETQR**(NODE, VAL)
SUBROUTINE **SETQS**(NODE, VAL)
SUBROUTINE **SETQSI**(NODE, VAL)
SUBROUTINE **SETT**(NODE, VAL)
SUBROUTINE **SETVDT**(NODE, VAL)
SUBROUTINE **SETVOL**(NODE, VAL)
SUBROUTINE **SETVQ**(NODE, VAL)

NODE	-	Node reference
CHAR	-	Character value
VAL	-	Double precision value

Purpose:

Set the given value of an entity at a given node. The node reference is either the internal node number or the node identifier (e.g. Dn).

Access to ESATAN nodal entities is provided for use in situations requiring a calculation for an arbitrary node, such as when looping over a list of nodes or in a subroutine where a node reference is passed in as a parameter.

The functions are named in the obvious way from the nodal entity concerned.

Example:

```
...
$NODES
...
FOR KL1 = 1 TO 100 DO
FKL1='Pipe1',
   A   =  0.02  , FD  =  0.05  , FL  =  1.00,
   P   =  1.00E5, FE  =  0.00  , T   = 14.00,
   FF  =  1.0E-2, FQ  =  1.00  , FM  =  0.02,
   FH  =  1.00  , FR  = 16.00  , FX  =  0.10,
   FY  =  0.20  , FZ  =  0.30  , FT  ='R114',
   VQ  =  0.50  , FRG =  0.02  , FLA =  0.01,
   VOL =  0.01  , PHI =  1.00  , FW  =  0.60,
   CMP = 60.00  , VDT =  6.0E-3, FST = 'P&T';
END DO
...
$CONDUCTORS
...
$INITIAL
        INTEGER I, N
        I = D1
        WHILE (I .LE. D100)
         #
         CALL SETVQ(I,0.D0)
         CALL SETA(I,1.D-4)
         CALL SETFD(I,1.D-2)
         CALL SETFL(I,0.02D0)
         CALL SETP(I,1.D5)
         CALL SETFE(I,0.D0)
         CALL SETT(I,20.D0)
         CALL SETFF(I,0.D0)
         CALL SETFQ(I,0.D0)
         CALL SETFM(I,1.D-4)
         CALL SETFH(I,0.D0)
         CALL SETFR(I,0.D0)
         CALL SETFX(I,0.D0)
         CALL SETFY(I,0.D0)
         CALL SETFZ(I,0.D0)
```

```
            CALL SETFT(I,'AMMONIA')
            CALL SETFRG(I,'LIQ')
            CALL SETFLA(I,0.D0)
            CALL SETVOL(I,0.D0)
            CALL SETPHI(I,0.D0)
            CALL SETFW(I,0.D0)
            CALL SETCMP(I,0.D0)
            CALL SETVDT(I,0.D0)
            CALL SETFST(I,'P&FE')
#
            I = I + 1
          END  WHILE
...
```

Restrictions:

The type of the node specified must be appropriate to the entity being requested; i.e. a fluid nodal entity value will not be returned for a thermal node, and vice versa.

User defined nodal entities are currently not supported.

Some set values might be overwritten by the solution routine (T, VOL, VQ, ...).

6.4.45 Set Conductor Value

Name:

SUBROUTINE **SETGF**(CNAME, NODE1, NODE2, VAL)
SUBROUTINE **SETGF2**(CNAME, NODE1, NODE2, CSQ, VAL)
SUBROUTINE **SETGL**(CNAME, NODE1, NODE2, VAL)
SUBROUTINE **SETGL2**(CNAME, NODE1, NODE2, CSQ, VAL)
SUBROUTINE **SETGP**(CNAME, NODE1, NODE2, VAL)
SUBROUTINE **SETGR**(CNAME, NODE1, NODE2, VAL)
SUBROUTINE **SETGR2**(CNAME, NODE1, NODE2, CSQ, VAL)
SUBROUTINE **SETGV**(CNAME, NODE1, NODE2, VAL)
SUBROUTINE **SETGV2**(CNAME, NODE1, NODE2, CSQ, VAL)
SUBROUTINE **SETM**(CNAME, NODE1, NODE2, VAL)

CNAME	-	Concatenated submodel name
NODE1	-	Node1 reference
NODE2	-	Node2 reference
CSQ	-	Conductor sequence number
VAL	-	Double precision value

Purpose:

Set the value of the conductor defined between the two given nodes, given the submodel where this conductor has been defined and, optionally given its sequence number. The node reference is either the internal node number or the node identifier (e.g. Dn).

Access to conductors values is provided for use in situations requiring a calculation for an arbitrary node, such as when looping over a list of conductors or in a subroutine where a node reference is passed in as a parameter.

The subroutines are named in the obvious way from the type of conductor concerned.

The suffix "2" indicates that the function allows users to address a specific sequence number. Basic functions assume CSQ = 1.

Example:

```
...
$CONDUCTORS
GL(1, 101) = 0.0D0;
GL(1, 101) = 0.0D0;
...
$INITIAL

        CALL SETGL (CURRENT,D1,D101, 1.23456D0)
        CALL SETGL2(CURRENT,D1,D101,2, 0.00044D0)
```

Restrictions:

None

6.4.46 Evaluate Thermal Frequency Response

Name:

SUBROUTINE EVALFR(NODE1, NODE2, FREQ, GAIN, PHASE)

NODE1	-	Node reference: input node
NODE2	-	Node reference: output node
FREQ	-	DOUBLE PRECISION: Frequency (Hz)
GAIN	-	DOUBLE PRECISION: Calculated gain of frequency response (K/W or dimensionless; see below)
PHASE	-	DOUBLE PRECISION: Calculated phase of frequency response (deg)

Purpose:

Evaluate the thermal frequency response between given nodes at a specified frequency. (Thermal nodes only.)

The frequency response transfer function computed by SLFRTF enables the prediction of how temperature will behave as a result of variations in boundary conditions. A variation could be applied to either a heat load or a boundary temperature (input). The transfer function relates this to the resulting fluctuation in temperature (output) at any diffusion node, assuming the input to be a sinusoidal oscillation at a given frequency. An arbitrary input fluctuation can be decomposed into sinusoidal components of different frequencies, according to Fourier analysis.

If a diffusion node is specified as input, it is assumed to imply a heat load variation of some sort, i.e. a QA, QE, QI, QR or QS. A boundary node as input implies a temperature variation. For output, which is always a temperature variation, it only makes sense to specify a diffusion node.

The frequency response is returned as the gain and phase of the specified output over the specified input. Gain is in either K/W for a heat-load input or dimensionless for a temperature input; phase is in degrees.

Example:

```
$MODEL DEMO
   $MODEL SUB1
   $NODES
   D5, T = 0.0, C = 0.456, QR = 25.0;
   D15, T = 0.0, C = 0.0;
   D25, T = 0.0, C = 2.8;
   ...
   $ALIAS
   Top_left = D5;
   Bottom_right = D15;
   Middle = D25;
```

```
   ...
   $ENDMODEL SUB1
$NODES
B1, T = 0.0;
B2, T = 0.0;
...
D300, T = 0.0, C = 1.23;
...
$EXECUTION
      DOUBLE PRECISION GAIN, PHASE, PSDIN, PSDOUT

      PSDIN = 0.026

      CALL SOLVFM  # Steady-state solution

      CALL SLFRTF  # Frequency-response transfer function

      CALL EVALFR(B1, D300, 1.0D-4, GAIN, PHASE)
      PSDOUT = (GAIN ** 2) * PSDIN
      ...
      CALL EVALFR(D:SUB1:Top_left, D:SUB1:Middle, 1.0D-4,
     &   GAIN, PHASE)
      PSDOUT = (GAIN ** 2) * PSDIN
      ...
$ENDMODEL DEMO
```

This will compute a) the response of temperature at node 300 resulting from a temperature fluctuation at node 1 of frequency 10^{-4} Hz; and b) the response of temperature at node Middle resulting from a variation in heat load applied to node Top_left, both within submodel SUB1, again at a frequency of 10^{-4} Hz.

Restrictions:

EVALFR should be called only from the $EXECUTION block.

The frequency response is calculated for thermal nodes and non-fluidic conductors only; fluidic conductors (GFs), fluid nodes and fluid conductors are ignored.

6.4.47 Dynamic Update from ACD File

Name:

SUBROUTINE ACDDYU

Purpose:

Update dynamic quantities in the model from data in the ACD file. Currently, 'dynamic' means time- or wavelength-dependent. See Section 3.10 for information on the ACD file.

This routine can be called from an operations block to force quantities such as time-dependent heat fluxes to be recalculated. Typical usage would be where a series of steady-state simulations are required at specific time points, as illustrated in the example below. The value of control constant TIMEM is used as the current analysis time.

Note that dynamic quantities are updated automatically from ACD data during steady-state or transient solution.

Example:

```
$EXECUTION
      # Solve for steady state at 50s and 100s

      TIMEM = 50.0D0
      CALL ACDDYU
      CALL SOLVFM

      TIMEM = 100.0D0
      CALL ACDDYU
      CALL SOLVFM
```

Restrictions:

None.

6.4.48 Thermostatic Heaters

Name:

SUBROUTINE **HEATER_DEFINE**(NAME, ZLABEL_TEMP, ON_TEMP, OFF_TEMP, MEASURE_METHOD, ZLABEL_HL, NODE_FACTORS_HL, HL_DEPEND, HL_NOMINAL_CONST, HL_NOMINAL_VAR, HL_PERIOD, HL_MULTIPLIER, MODE_TRANSIENT, INITIAL_STATE_TRANS, MODE_STEADY, LOAD_FRACTION_STEADY, DAMP_STEADY, SET_POINT_STEADY)

SUBROUTINE **HEATER_RESET**(NAME, CNAME)

SUBROUTINE **HEATER_STATUS**(NAME, CNAME, MEASURED_TEMP, STATE, POWER_APPLIED, NUM_SWITCH_ONS, TIME_ON, DUTY_CYCLE)

NAME	-	CHARACTER: Name of heater (or blank)
ZLABEL_TEMP	-	CHARACTER: Node list for temperature measurement
ON_TEMP	-	DOUBLE PRECISION: Thermostat on-temperature
OFF_TEMP	-	DOUBLE PRECISION: Thermostat off-temperature
MEASURE_METHOD	-	INTEGER: Temperature-measurement method flag
ZLABEL_HL	-	CHARACTER: Node list for heat load
NODE_FACTORS_HL	-	*ARRAY REFERENCE:* Array of factors for applying heat load to each node [or zero]
HL_DEPEND	-	INTEGER: Heat-load dependence flag
HL_NOMINAL_CONST	-	DOUBLE PRECISION: Nominal constant heat load (HL_DEPEND = 0)
HL_NOMINAL_VAR	-	*ARRAY REFERENCE:* Array of heat load vs time/temperature [or zero]
HL_PERIOD	-	DOUBLE PRECISION: Period for cyclically varying heat load
HL_MULTIPLIER	-	DOUBLE PRECISION: Overall heat-load multiplying factor
MODE_TRANSIENT	-	INTEGER: Operating-mode flag for transient
INITIAL_STATE_TRANS	-	INTEGER: Initial state for transient
MODE_STEADY	-	INTEGER: Operating-mode flag for steady state
LOAD_FRACTION_STEADY	-	DOUBLE PRECISION: Load fraction for steady-state Fixed mode
DAMP_STEADY	-	DOUBLE PRECISION: Damping factor for steady-state Proportional mode
SET_POINT_STEADY	-	DOUBLE PRECISION: Target temperature for stead-state Set-point mode
CNAME	-	*MODEL REFERENCE:* Submodel in which heater is defined
MEASURED_TEMP	-	DOUBLE PRECISION: Temperature measured by thermostat
STATE	-	INTEGER: Current state of heater
POWER_APPLIED	-	DOUBLE PRECISION: Total power currently applied by heater
NUM_SWITCH_ONS	-	INTEGER: Number of times heater has switched on (transient)

TIME_ON	-	DOUBLE PRECISION: Total amount of time heater has been switched on (transient)
DUTY_CYCLE	-	DOUBLE PRECISION: Current estimate of duty cycle

Purpose:

HEATER_DEFINE defines a thermostatically controlled heater, and would normally be called from the $INITIAL block. The status of every heater defined in the model is automatically updated during solution in $VARIABLES1/2 (Section 6.1.1): the heat load is calculated according to the measured temperature, and applied to each node as a QI.

Physically, such a heater is a transient device, normally operating in an ‘on/off’ or ‘bang-bang’ fashion (there is also a ‘proportional’ mode); however, steady-state simulation can be useful for preconditioning a TMM prior to running a transient solution, or for preliminary heater sizing or estimation of duty cycles; three suitable modes are supported. HEATER_DEFINE specifies the behaviour of the heater in both transient and steady state solutions.

For a transient solution the following modes can be selected:

- On/Off (MODE_TRANSIENT = 1): If the measured temperature is at or below ON_TEMP, full heat load is applied. If it is at or above OFF_TEMP, the applied heat load is zero. Between these limits the heat load is either full or zero depending on the heater's recent history.
- Proportional (MODE_TRANSIENT = 2): Below ON_TEMP full heat load is applied, and above OFF_TEMP zero heat load is applied. Between these limits the heat load varies linearly.

In steady-state solution, the following modes are available:

- Fixed (MODE_STEADY = 1): A constant fraction of the heat load is applied (LOAD_FRACTION_STEADY). The measured temperature is disregarded.
- Proportional (MODE_STEADY = 2): As for transient. A damping factor (DAMP_STEADY) is required to promote solution convergence; 0.01 should be a suitable value.
- Set-Point (MODE_STEADY = 3): In this mode, the heat-load nodes are held at a constant temperature (SET_POINT_STEADY) during solution, and the heat required to achieve this condition is calculated. The measured temperature is disregarded. NB: The smaller the thermal distance between the temperature nodes and the heat-load nodes, the more accurate this method is for heater sizing; ideally, they should be coincident. The predicted heater power should be validated by use of one of the other modes (steady state or transient).

The temperature may be evaluated across several nodes, in which case the average (MEASURE_METHOD = 1), minimum (MEASURE_METHOD = 2) or maximum

(MEASURE_METHOD = 2) will be used. (Of course, for a single temperature node, this parameter is irrelevant.)

The applied heat load is distributed to the nodes according to the elements of array NODE_FACTORS_HL. If the given nominal heat load is a total power (W), then these factors are fractions which should sum to 1; if flux (Wm^{-2}), then they are nodal areas (m^2). If the nominal heat load is a power to be applied to each node (W/node) then each factor should equal 1.0. As a special case, a value of 0 may be passed into the routine instead of an array reference to indicate a total power to be spread equally across the nodes (equal fractions).

The nominal heat load may be constant (HL_DEPEND = 0), time dependent (HL_DEPEND = 1), cyclic time dependent (HL_DEPEND = 2), temperature dependent (HL_DEPEND = 3), or time and temperature dependent (HL_DEPEND = 4). If constant, the value is given by HL_NOMINAL_CONST; otherwise, an appropriate array must be referenced as HL_NOMINAL_VAR (a 2D table array for the last option, otherwise 2D real array). For option 2, the period must be given in HL_PERIOD.

HL_MULTIPLIER scales the applied heat load by the given factor; a value of 1.0D0 is usually appropriate.

INITIAL_STATE_TRANS is relevant only to the transient On/off mode, and only if the initial measured temperature is within the on/off-temperature band. Passing in 0 or 1 instructs the heater to start in the ‘off’ or ‘on’ state, respectively.

Where an array reference is not required, the integer value 0 should be passed to the routine instead.

Note that an existing heater may be redefined (modified) by calling HEATER_DEFINE a second time with the same name, e.g. between solutions in order to compare different nominal powers.

HEATER_RESET zeroes the ‘accumulated’ output parameters of the specified heater – number of switch-ons, total time on, duty cycle. As a special case, giving the heater name as blank resets all heaters in the specified submodel (and included submodels).

HEATER_STATUS returns performance parameters for the specified heater. The state of the heater is given by an integer value: STATE = 0 for ‘off’ and 1 for ‘on’. POWER_APPLIED is the total power (W) aggregated over all the heat-load nodes. The duty cycle for a transient solution is the ratio of the time on to the total solution time; for steady state, this is approximated as the ratio of applied power to available power.

Example:

```
. . .
$ARRAYS
$REAL
nodeFactors(2) = 0.4, 0.6;
```

```
. . .
$INITIAL
      CALL HEATER_DEFINE('htr1',
     &   '#13,15,17', 25.0D0, 30.0D0, 1,   # Temperature
     &   '#123,124', nodeFactors, 0, 50.0D0, 0, 0.0D0, 1.0D0,  # Heat load
     &   1, 0,                          # Transient mode - on/off
     &   2, 0.0D0, 0.01D0, 0.0D0)       # Steady-state mode - proportional
. . .
$VARIABLES2
      DOUBLE PRECISION htr1_temp, htr1_power, htr1_timeOn, htr1_dutyCycle
      INTEGER htr1_state, htr1_numSwitchOns

      CALL HEATER_STATUS('htr1', CURRENT,
     &   htr1_temp, htr1_state, htr1_power,
     &   htr1_numSwitchOns, htr1_timeOn, htr1_dutyCycle)

      WRITE(55, *) TIMEN, htr1_temp, htr1_state, htr1_power,
     &   htr1_numSwitchOns, htr1_timeOn, htr1_dutyCycle
```

Restrictions:

Only solid nodes are allowed for the temperature and heat load node lists (ZLABEL_TEMP and ZLABEL_HL); i.e. fluid nodes are not supported.

If steady-state Set-point mode is used, no other internal heat sources should be applied to the same nodes.

6.5 Heat Transfer

ALRADI	Calculation of total albedo heat source
ERRADI	Calculation of total earth heat source
FLUXF	Calculate total fluidic heat flow
FLUXGF	Calculate total fluidic heat flow
FLUXGL	Calculate total linear heat flow
FLUXGR	Calculate total radiative heat flow
FLUXGT	Calculate total heat flow
FLUXL	Calculate total linear heat flow
FLUXMF	Calculate total fluidic heat flow
FLUXML	Calculate total linear heat flow
FLUXMR	Calculate total radiative heat flow
FLUXR	Calculate total radiative heat flow
FLUXT	Calculate total heat flow
SLRADI	Calculation of total solar heat source
THRMST	Simulates thermostat
VCHPD	Design module for VCHP
VCHP	Control routine for VCHP
VFAC	Calculate radiative conductors from GV

HTCOEF	Calculate linear conductance due to convection
NUVRE	Calculate linear conductance due to convection
STVRE	Calculate linear conductance due to convection

HTBOKR	Boyko-Kruzhilin HTC
HTCHAT	Chato HTC
HTCHEN	Chen HTC
HTDIBO	Dittus-Boelter HTC
HTDOBS	Dobson HTC
HTDORO	Dougall-Rohsenow HTC
HTROMY	Rosson-Myers HTC
HTTRAV	Traviss HTC

6.5.1 Absorbed Heat Flux

Name:

SUBROUTINE **ALRADI**
SUBROUTINE **ERRADI**
SUBROUTINE **SLRADI**

Purpose:

These subroutines calculate the total heat flux absorbed at each node within an enclosure, taking into account all the reflective terms. The model must be a view factor submodel.

ALRADI calculates the total albedo heat source absorbed.
ERRADI calculates the total earth heat source absorbed.
SLRADI calculates the total solar heat source absorbed.

The incident heat source at each node within the enclosure has to be specified using the nodal entities QAI, QEI, QSI for albedo, earth, solar respectively. If no value is given zero is assumed. The solar absorptivity for each node has to be specified for SLRADI whilst for ERRADI and ALRADI emissivity has to be defined. Also the area for each node and all line-of-sight view factors given as view factor conductors in the $CONDUCTORS block, must be defined for all three routines.

These routines calculate the actual view factor taking into account multiple reflections. Using the incident heat flux at each node the total heat flux is calculated and assigned to the corresponding nodal entity. That is for the earth incident heat flux at a node, QEI, the total heat flux is calculated by calling ERRADI and this value is assigned to the nodal entity QE for that node. For albedo incident heat flux at a node, QAI, the total heat flux is calculated by calling ALRADI and this value is assigned to the nodal entity QA for that node. Finally for solar incident heat flux at a node, QSI, the total heat flux is calculated by calling SLRADI and this value is assigned to the nodal entity QS for that node.

Example:

```
$INITIAL
      CALL ALRADI
      CALL ERRADI
      CALL SLRADI
```

Restrictions:

These routines must be called from the $INITIAL or $VARIABLES1 operations block of the submodel to be processed. Clearly this must be a view factor submodel. All three routines require (3 * NNS ** 2 + NNS) locations of dynamic core, where NNS is the number of nodes in the submodel.

All radiative heat transfer information must be defined at the time of a call to any of these routines, i.e. nodal area, emissivity and absorptivity (even if there is no

incident heat flux), view factor conductances and the incident heat sources at each node. If any of these are assigned to variable expressions, then the GENMOR statement (see Section 3.6.3) must be used in VARIABLES1 at a point preceding the call to these routines in order to execute the generated Mortran resulting from such definitions.

6.5.2 Heat Flow - Submodels and Node Ranges

Name:

FUNCTION **FLUXML (**CNAME1, CNAME2)
FUNCTION **FLUXMF** (CNAME1, CNAME2)
FUNCTION **FLUXMR** (CNAME1, CNAME2)
FUNCTION **FLUXL** (D|F[:CNAME1:]NODE1, D|F[:CNAME2:]NODE2, D|F[:CNAME3:]NODE3, D|F[:CNAME4:]NODE4)
FUNCTION **FLUXF** (D|F[:CNAME1:]NODE1, D|F[:CNAME2:]NODE2, D|F[:CNAME3:]NODE3, D|F[:CNAME4:]NODE4)
FUNCTION **FLUXR** (D|F[:CNAME1:]NODE1, D|F[:CNAME2:]NODE2, D|F[:CNAME3:]NODE3, D|F[:CNAME4:]NODE4)
FUNCTION **FLUXT** (D|F[:CNAME1:]NODE1, D|F[:CNAME2:]NODE2, D|F[:CNAME3:]NODE3, D|F[:CNAME4:]NODE4)

CNAME*n*	-	sub-model names (see Section 6.1.3)
D[:CNAME:]NODE*n*	-	[sub-]model node numbers

Type:

Double precision.

Purpose:

It is important to note that FLUXL, FLUXF, FLUXR and FLUXT will be made redundant in future versions of ESATAN since they have been obsoleted by FLUXGL, FLUXGF, FLUXGR and FLUXGT (Section 6.5.3).

FLUXML, FLUXMF and FLUXMR are calculated as the total linear, fluidic and radiative heat flows, respectively, between CNAME1 and CNAME2. If CNAME2 is an included sub-model of CNAME1 then CNAME1 is regarded as all sub-models within CNAME1 except those within CNAME2.

FLUXL, FLUXF, FLUXR and FLUXT are calculated as the linear, fluidic, radiative and total heat flows, respectively, between two node bands. The node bands are specified using the notation shown, not in quotes.

If *I* is the set of nodes in the first sub-model or node band and *J* is the set of nodes in the second sub-model or node band then the calculation of flux is as follows:

for linear flow

$$Q = \sum_{i \in I} \sum_{j \in J} GL_{i,j}(T_j - T_i)$$

for radiative flow

$$Q = \sum_{i \in I} \sum_{j \in J} GR_{i,j}\sigma(T_j^4 - T_i^4)$$

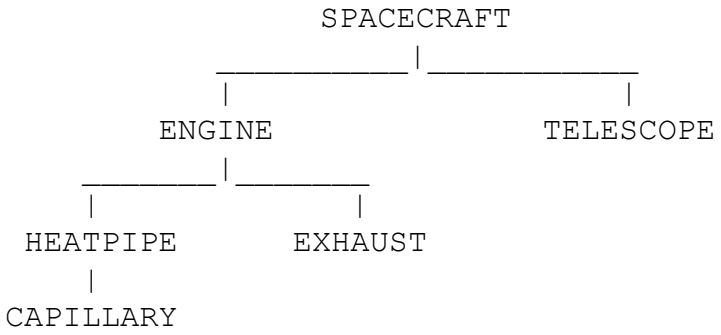
and for one way flow

$$Q = \sum_{i \in I} \sum_{j \in J, upstream} GF_{i,j}(T_j) - \sum_{i \in I, upstream} \sum_{j \in J} GF_{i,j}(T_i)$$

The temperature T is the absolute temperature. Note that a positive value returned by one of these functions represents a net flow of heat into the first named sub-model or node band.

Examples:

Consider the following sub-model structure:-



Submodel	Nodes
ENGINE	5, 10, 15
HEATPIPE	1, 3, 5, 7
CAPILLARY	50, 100, 1000
EXHAUST	50
TELESCOPE	10, 20, 30

```
HF = FLUXML (ENGINE, HEATPIPE)
```

This instruction sets HF equal to the total linear heat flow between nodes defined in the $NODES block of both ENGINE and EXHAUST and nodes defined in the $NODES block of both HEATPIPE and CAPILLARY.

```
HF = FLUXR(D:ENGINE:10, D:CAPILLARY:100,
          D:EXHAUST:50,D:TELESCOPE:10)
```

Here HF is set equal to the total radiative heat flow between the two defined node bands. The first node band comprises the nodes ENGINE:10,15,

HEATPIPE:1,3,5,7 and CAPILLARY:50,100. The second node band comprises the nodes EXHAUST:50 and TELESCOPE:10

Restrictions:

Overlapping node bands e.g. (ENGINE:5, CAPILLARY:1000, HEATPIPE:1, HEATPIPE:7) are **not** allowed. No account of heat flow via mass flow links is taken into account.

6.5.3 Heat Flow - Groups

Name:

FUNCTION **FLUXGL (**ZLABEL1, CNAME1, ZLABEL2, CNAME2)
FUNCTION **FLUXGF (**ZLABEL1, CNAME1, ZLABEL2, CNAME2)
FUNCTION **FLUXGR (**ZLABEL1, CNAME1, ZLABEL2, CNAME2)
FUNCTION **FLUXGT (**ZLABEL1, CNAME1, ZLABEL2, CNAME2)

ZLABEL*n*	-	character node list (see Section 6.1.3)
CNAME*n*	-	concatenated submodel name (see Section 6.1.3)

Type:

Double precision.

Purpose:

FLUXGL, FLUXGF, FLUXGR and FLUXGT are calculated as the linear, fluidic, radiative and total heat flows, respectively, between two ZLABEL groups of nodes. The node groups are specified using the ZLABEL/CNAME convention described in Section 6.1.3.

A ZLABEL2/CNAME2 combination of ' '/CURRENT is interpreted as requesting the flux from ZLABEL1/CNAME1 to the rest of the model.

If I is the set of nodes in the first ZLABEL group and J is the set of nodes in the second ZLABEL group then the calculation of flux is as follows:

for linear flow

$$Q = \sum_{i \in I} \sum_{j \in J} GL_{i,j}(T_j - T_i)$$

for radiative flow

$$Q = \sum_{i \in I} \sum_{j \in J} GR_{i,j}\sigma(T_j^4 - T_i^4)$$

and for one way flow

$$Q = \sum_{i \in I} \sum_{j \in J,\, upstream} GF_{i,j}(T_j) - \sum_{i \in I,\, upstream} \sum_{j \in J} GF_{i,j}(T_i)$$

The temperature *T* is the absolute temperature. Note that a positive value returned by one of these functions represents a net flow of heat into the first named sub-model or node band.

Examples:

```
$CONSTANTS
$CHARACTER
LIST1 = '#1,3;sub1:4-6;sub2:3,4';
...
$OUTPUTS
      HF = FLUXGL(LIST1, CURRENT, '#1-10', sub3)
```

Restrictions:

With the exception of the case of ZLABEL2/CNAME2 being ' '/CURRENT, the same node may **not** appear in both ZLABEL/CNAME combinations. No account of heat flow via mass flow links is taken into account.

6.5.4 Convective Heat Transfer

Name:

FUNCTION **HTCOEF**(F[:CNAME:]NODE, D[:CNAME:]NODE)
FUNCTION **NUVRE**(F[:CNAME:]NODE, ARRNAM)
FUNCTION **STVRE**(F[:CNAME:]NODE, ARRNAM)

F[:CNAME:]NODE	-	Fluid node reference
D[:CNAME:]NODE	-	Thermal node reference
ARRNAM	-	Real ESATAN array

Type:

Double precision.

Purpose:

Each function returns a linear thermal conductance value which can be used to define a convective link.

HTCOEF calculates a single- or two-phase heat-transfer coefficient, as appropriate, using correlations described in Appendix K. A call to this function is automatically supplied by the preprocessor for a conductor definition GL = *, and the user would not normally use it explicitly.

NUVRE and STVRE use interpolation to compute a heat-transfer coefficient. The array ARRNAM should be an ESATAN 2×n array containing Reynolds number/Nusselt number pairs (NUVRE) or Reynolds number/Stanton number pairs(STVRE).

In each case, physical dimensions and fluid properties are evaluated at the named nodes. The calculated heat-transfer coefficient is multiplied by the fluid node's heat-transfer area (entity A) to give the conductance.

Example:

Using the function NUVRE to define the convective heat transfer coefficient between fluid node number 6 and thermal node number 11.

```
$CONDUCTORS
# Convective heat transfer between fluid and pipe wall
GL(6,11)=NUVRE(F6,NRHT);
#
$ARRAYS
$REAL
# Array of Reynolds number/Nusselt number pairs
NRHT(2,8)=4000.0,  214.0, 8000.0,  296.0,
          12000.0, 373.0, 16000.0, 446.0,
          20000.0, 776.0, 40000.0, 1351.0,
          80000.0, 1992.0,130000.0,2000.0;
```

Restrictions:

These functions can only be used with linear conductors (GLs) and can only define the convective heat transfer between a fluid and a thermal node.

6.5.5 Thermostat Simulator

Name:

SUBROUTINE **THRMST** (T, TH, TL, STATE)

T	-	Double precision: Current temperature
TH	-	Double precision: High temperature
TL	-	Double precision: Low temperature
STATE	-	Integer status flag

Purpose:

THRMST simulates the action of a thermostat by setting STATE to 1 when T falls below TL, and 0 when T rises above TH. Note that no action is taken for TL < T < TH, STATE maintaining its previous value. Note: For modelling a thermostatically controlled heater, the user should consider using HEATER_DEFINE (Section 6.4.48).

Example:

```
$CONSTANTS
#
STATE = 0;
#
$VARIABLES2
#
        CHARACTER * 4 POWER
#
        CALL THRMST(T:SUB1:4, 25.0D0, 15.0D0, STATE)
#
        IF (STATE .EQ. 1) THEN
            POWER = 'ON'
        ELSE
            POWER = 'OFF'
        END IF
#
```

In this example, the STATE variable is defined as a user constant within the $CONSTANTS blocks to ensure that the STATE value is maintained between successive calls.

Restrictions:

Error recorded if TL > TH

6.5.6 VCHP Control

Name:

SUBROUTINE **VCHP**
SUBROUTINE **VCHPD** (CNAME, T*n*, VAL, TOL)

CNAME	-	Model name
T*n*	-	Double precision: Nodal temperature reference
VAL	-	Double precision: Required temperature
TOL	-	Double precision: Tolerance on required temperature

Purpose:

VCHP is the variable conductance heat pipe (VCHP) control routine for both design and simulation modes of operation. A call to this routine must appear in the $VARIABLES1 block of each VCHP submodel after the GENMOR card. For a definition of a VCHP submodel, see Section 3.9.

VCHPD is the design module. Given a VCHP submodel name, a nodal temperature reference, a required temperature and a tolerance, VCHPD will calculate the number of moles of control gas needed in the VCHP in order to keep the temperature of the specified node within the bounds dictated by the required temperature and tolerance. The number of moles of control gas is stored in user constant MOLESG of the submodel, and the predicted vapour/gas front in the user constant VAPLEN, where VAPLEN is measured from the evaporator end of the pipe. VCHPD should only be called from the $EXECUTION block since it calls the steady state solution routine SOLVIT during the design procedure. In fact SOLVIT may be called several times during a single call of VCHPD, and this fact should be taken into consideration when writing the $OUTPUTS block which will be executed once for each call to SOLVIT. In addition, all control constants which are required by SOLVIT must be specified for a call to VCHPD.

In the design mode any user-supplied values for these user constants is ignored. However for a steady state simulation MOLESG must be specified, and a realistic estimate of VAPLEN may speed convergence. For a transient analysis both MOLESG and VAPLEN must be specified with correct initial values.

Examples:

```
$EXECUTION
      QR2=15.0
      CALL VCHPD(CURRENT,T2,50.0,0.5)
      QR2=5.0
      VAPLEN=0.3
      CALL SOLVIT
$VARIABLES1
      GENMOR
      CALL VCHP
```

In this example, the VCHP in the current submodel is designed such that the temperature of node 2 lies between 49.5 and 50.5 degrees Celsius. The heat input to node 2 is set to 15 Watt, in the statement before the call to VCHPD. Then the operation of the VCHP is found at a lower power level (5 Watt). An initial estimate of the vapour length is made to aid convergence. Notice the GENMOR statement before the call to VCHP in the $VARIABLES1 block.

Restrictions:

- VCHP should **not** be called from within a $VARIABLES1 block.
- VAPLEN and MOLESG must have defined values.
- The call to VCHP must be done in the $VARIABLES1 block of the VCHP model.

6.5.7 View Factor to GR

Name:

SUBROUTINE **VFAC**

Purpose:

This routine calculates radiative conductances for a view factor submodel. Such a submodel is always a model of a complete enclosure. The area and emissivity of each node must be specified in the $NODES data block, and all line-of-sight view factors given as view factor conductors in the $CONDUCTORS data block. VFAC then calculates the radiation conductances, taking into account multiple reflections, and enters them as GR conductors ready for solution.

Example:

```
$INITIAL
#
      CALL VFAC
#
```

Restrictions:

VFAC must be called from the $INITIAL or $VARIABLES1 operations block of the submodel to be processed. Clearly this must be a view factor submodel. VFAC requires (3*NNS**2+NNS) locations of dynamic core, where NNS is the number of nodes in the submodel.

All radiative heat transfer information must be defined at the time of a VFAC call, i.e. nodal area and emissivity (even if there are no GVs at the node), and view factor conductances. If any of these are assigned to variable expressions, then the GENMOR statement (see Section 3.6.3) must be used in VARIABLES1 at a point preceding the call to VFAC in order to execute the generated Mortran resulting from such definitions.

6.5.8 Heat Transfer Correlations

Name:

FUNCTION **HTBOKR**(PRES, TEMP, QUAL, MFLOW,
AREA, DIA, FTI, NODE)
FUNCTION **HTCHAT**(PRES, TEMP, QUAL, TWALL,
DIA, GRAV, FTI, NODE)
FUNCTION **HTCHEN**(PRES, TEMP, QUAL, MFLOW, TWALL,
AREA, DIA, FTI, NODE)
FUNCTION **HTDIBO**(PRES, TEMP, QUAL, MFLOW, TWALL,
AREA, DIA, FTI, NODE)
FUNCTION **HTDOBS**(PRES, TEMP, QUAL, MFLOW, TWALL,
AREA, DIA, GRAV, FTI, NODE)
FUNCTION **HTDORO**(PRES, TEMP, QUAL, MFLOW,
AREA, DIA, FTI, NODE)
FUNCTION **HTROMY**(PRES, TEMP, QUAL, MFLOW, TWALL,
AREA, DIA, GRAV, FTI, NODE)
FUNCTION **HTTRAV**(PRES, TEMP, QUAL, MFLOW,
AREA, DIA, FTI, NODE)

PRES	-	Double precision: Pressure
TEMP	-	Double precision: Temperature of fluid
QUAL	-	Double precision: Vapour quality
MFLOW	-	Double precision: Mass flow rate
TWALL	-	Double precision: Wall temperature
AREA	-	Double precision: Flow area
DIA	-	Double precision: Hydraulic diameter
GRAV	-	Double precision: Gravitational acceleration
FTI	-	Integer: Fluid-type number
NODE	-	Node reference (or zero)

Type:

Double precision

Purpose:

To compute heat-transfer coefficients using certain well-known correlations (see table below). Where applicable, subcooled boiling and superheated condensation are allowed for.

Input values of pressure and temperature are in model units, i.e. on the scale determined by the control constants `PABS` and `TABS`. If one of these functions is used in the $SUBROUTINES block with LANG = FORTRAN, or in an external Fortran routine, the values of `PABS` and `TABS`, as well as `GRAVX/Y/Z` if needed to calculate GRAV, can be obtained by calling GETCCR (Section 6.4.42).

Correlation	Function	Applicability
Boyko-Kruzhilin[18]	HTBOKR	Annular condensation
Chato[18]	HTCHAT	Stratified condensation
Chen[18]	HTCHEN	Forced-convection boiling
Dittus-Boelter[2][18]	HTDIBO	Turbulent single-phase
Dobson[18]	HTDOBS	Stratified wavy condensation
Dougall-Rohsenow[19]	HTDORO	Film boiling
Rosson-Myers[20]	HTROMY	Slug, plug and stratified wavy condensation
Traviss, et al.[18]	HTTRAV	Annular condensation

The fluid-type number, FTI, is obtained from the fluid-type name via the function FTYPEI (Section 6.4.40). The user should not set FTI directly, as the fluid number will vary depending on which fluid property library is being used.

Each HTC is evaluated at the given fluid state: the node reference, NODE, is passed to the functions purely for information, e.g. for use in error messages. If it is unknown or not relevant, this parameter should be set to zero.

Note that the value returned must be multiplied by the heat-transfer area for use as a conductance.

Example:

Consider a model containing a simple condenser. It is required to use the Rosson-Myers correlation for the condenser HTC.

```
$NODES
...
# Condenser nodes
F5, ...;
D105, ...;
...
#
$CONDUCTORS
GL(5, 105) = HTROMY(P5, T5, VQ5, NDMFL(F5), T105,
                    FLA5, FD5, GRAVZ, FTYPEI(FT5), F5)
             * A5;
#
```

Restrictions:

It is assumed that the correlation applied is appropriate to the fluid state specified: the flow regime is not checked.

6.5.9 Piecewise Grey-Body Radiation

Name:

SUBROUTINE **PWGBR**(NODE1, NODE2, RDATA, WBANDS, TPOINTS, TIME)

NODE1 - Node reference: First node of GR conductor

NODE2 - Node reference: Second node of GR conductor

RDATA - Array reference: Radiative coupling data (per waveband and time-point)

WBANDS - Array reference: Waveband limits

TPOINTS - Array reference: Time-points

TIME - DOUBLE PRECISION: Current solution time

Purpose:

This routine calculates radiative couplings for wavelength-dependent thermo-radiative analysis.

Normally, all thermal radiation is assumed to take place in the infra-red part of the spectrum, and thermo-optical properties (emissivity, absorptivity, etc.) are assumed to be constant within this range. However, in some situations this idealisation is not sufficiently accurate and it becomes necessary to account for the dependence of thermo-optical properties on the wavelength of the emitted radiation. One of the most practical methods for this is the piecewise grey-body (PWGB) method.

In the PWGB approach the radiation spectrum is split into a number of wavebands; for a particular radiative conductor, a value of conductance is supplied for each waveband, obtained by standard techniques assuming emissivities are constant for that waveband. These couplings are then combined as a weighted sum, the weights being derived from consideration of the black-body emissive power spectrum. This in fact results in an asymmetric radiative coupling; an effective (symmetric) GR can be derived by dividing the net heat exchange by the 4th-power temperature difference. (For mathematical details, see the Engineering Manual.)

The routine PWGBR computes the effective wavelength-dependent coupling and applies it to an existing GR conductor. The conductor (which must exist in the submodel from which the routine is called) is specified by giving the references of the appropriate nodes. The waveband conductance values are provided in a 1-d array, as are the wavelengths (in metres) defining the limits of each corresponding waveband.

It may be required that the radiative conductances vary with time, due to relative motion of different parts of the geometry; a spacecraft antenna, for example. In this case a number of time-points are input as a 1-d array and, within each waveband, the conductance at each time-point is supplied; conductances at intermediate times are found by linear interpolation.

The time-point array must always be given even if there is no time-dependence (in which case a single time-point should be defined), as must the current solution time.

The combined GR is dependent on temperature and so, for this to be properly taken into account by the solution, PWGBR should be called from the $VARIABLES1 block.

Example:

The first example illustrates a wavelength-dependent analysis with time-independent radiative coupling data using three wavebands:

```
$MODEL WAVY1
$NODES
D1, T = 0.0, C = 5.0;
D2, T = 0.0, C = 5.0;
...
$CONDUCTORS
GR(1, 2) = 0.0;
...
$ARRAYS
 $REAL
 timePoints(1) = 0.0;
 wavebands(4) = 0.0, 1.0E-6, 10.0E-6, 1.0E10;
 wrad_1_2(3) = 0.05, 0.04, 0.03;  # GR(1,2) for each   waveband
 ...
$VARIABLES1
 CALL PWGBR(D1, D2, wrad_1_2, wavebands, timePoints, TIMEM)
 ...
$EXECUTION
 CALL SOLVFM
$ENDMODEL WAVY1
```

The next example shows how time-varying radiative coupling data is defined for a wavelength-dependent analysis:

```
$MODEL WAVY2
$NODES
D1, T = 0.0, C = 5.0;
D2, T = 0.0, C = 5.0;
...
$CONDUCTORS
GR(1, 2) = 0.0;
...
$ARRAYS
 $REAL
 timePoints(2) = 0.0, 3600.0;
 wavebands(4) = 0.0, 1.0E-6, 10.0E-6, 1.0E10;
 wrad_1_2(3) =
         0.05, 0.075,  # GR(1,2): 1st waveband, 2 time-points
         0.04, 0.06,   # GR(1,2): 2nd waveband, 2 time-points
         0.03, 0.045;  # GR(1,2): 3rd waveband, 2 time-points
...
```

```
$VARIABLES1
 CALL PWGBR(D1, D2, wrad_1_2, wavebands, timePoints, TIMEM)
...
$EXECUTION
        CALL SOLVFM
        CALL SLCRNC
$ENDMODEL WAVY2
```

Note how a very large wavelength is given for the upper limit of the last waveband as an approximation for infinity.

Restrictions:

PWGBR should be called only from the $VARIABLES1 block of the model. There must be a corresponding GR conductor defined in $CONDUCTORS block. If two such conductors exist in parallel, the first one is used (sequence number 1). If the GR is in a submodel, PWGBR must be invoked in that submodel.

An array of time-points must always be supplied, even when the radiative couplings are not time-dependent; in such cases the array should consist of a single element.

Numerical Background:

The net radiative heat exchange from surface *i* to surface *j* is given by

$$Q_{ij} = \sigma \sum_{m=1}^{M} R_{ij}^{(m)} \left[f_{band}(\lambda_{m-1}, \lambda_m, T_i) T_i^4 - f_{band}(\lambda_{m-1}, \lambda_m, T_j) T_j^4 \right]$$

where the *m*th waveband lies between wavelengths λ_{m-1} and λ_m , $R_{ij}^{(m)}$ is the associated radiative coupling (obtained by the usual methods, e.g. Monte-Carlo ray-tracing, assuming constant emissivities), and $f_{band}(\lambda_{m-1}, \lambda_m, T)$ is the *black-body band fraction*. This last is defined in terms of the fractional black-body emissive power, $f(\lambda T)$:

$$f_{band}(\lambda_{m-1}, \lambda_m, T) \equiv f(\lambda_m T) - f(\lambda_{m-1} T)$$

where

$$f(\lambda T) = \frac{15}{\pi^4} \int_{hc/k\lambda T}^{\infty} \frac{\xi^3}{e^{\xi} - 1} d\xi$$

The derivation of this expression for $f(\lambda T)$ involves Planck's black-body radiation law and can be found in most text books on the subject.

Observe that by defining the weighted sum

$$\tilde{R_{ij}} \equiv \sum_{m=1}^{M} R_{ij}^{(m)} f_{band}(\lambda_{m-1}, \lambda_m, T_i)$$

The net radiative heat exchange equation above can be written as

$$Q_{ij} = \sigma(\tilde{R}_{ij} T_i^4 - \tilde{R}_{ji} T_j^4)$$

Hence $\tilde{R}_{ij}$ can be regarded as an *asymmetric* radiative coupling: $\tilde{R}_{ij} \neq \tilde{R}_{ji}$, $\tilde{R}_{ij} \neq -\tilde{R}_{ji}$

An effective radiative conductance, $\overline{R}_{ij}$, is defined by writing

$$Q_{ij} \equiv \sigma \overline{R}_{ij} (T_i^4 - T_j^4)$$

i.e.

$$\overline{R}_{ij} \equiv \frac{\tilde{R}_{ij} T_i^4 - \tilde{R}_{ji} T_j^4}{T_i^4 - T_j^4}$$

(which is symmetric). This quantity is then assigned to the appropriate GR conductor. The equation above becomes singular when $T_i = T_j$, however it can be shown that a finite limit is approached as $T_i \to T_j$. There is no simple expression for this limit, so the solution used is to approximate it by testing for $|T_i - T_j| < \varepsilon$, where ε is a small number, and replacing T_j by $T_{i \pm \varepsilon}$ before evaluating the equation above in this case. A tolerance of $\varepsilon = 10^{-6} K$ has been used.

6.6 Integration

INTGL1 Trapezoidal integration over array
INTGL2 Trapezoidal integration over table array
INTEGL Trapezoidal integration over two 1-dimensional arrays

6.6.1 Trapezoidal Integration

Name:

FUNCTION **INTGL1** (X1, X2, ARRNAM)
FUNCTION **INTGL2** (X1, X2, Y1, Y2, ARRNAM)
FUNCTION **INTEGL** (X1, X2, ARRX, ARRY)

X1, X2, Y1, Y2 - Double precision independent variable limits

ARRNAM, ARRX, ARRY - Array names (see below for type)

Type:

Double precision.

Purpose:

Trapezoidal integration over the given region, on the array(s) ARRNAM or ARRX and ARRY.

In INTGL1 the array may be of two types:-

i. Table array with one independent variable, e.g. *V(x)*

ii. Real array of dimensions (2, *n*) input as $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$

In INTGL2 the array must be a table array with two independent variables, e.g. *V(x, y)*.

For INTEGL, ARRX and ARRY should be 1-dimensional arrays containing the independent and corresponding dependent variable values, respectively.

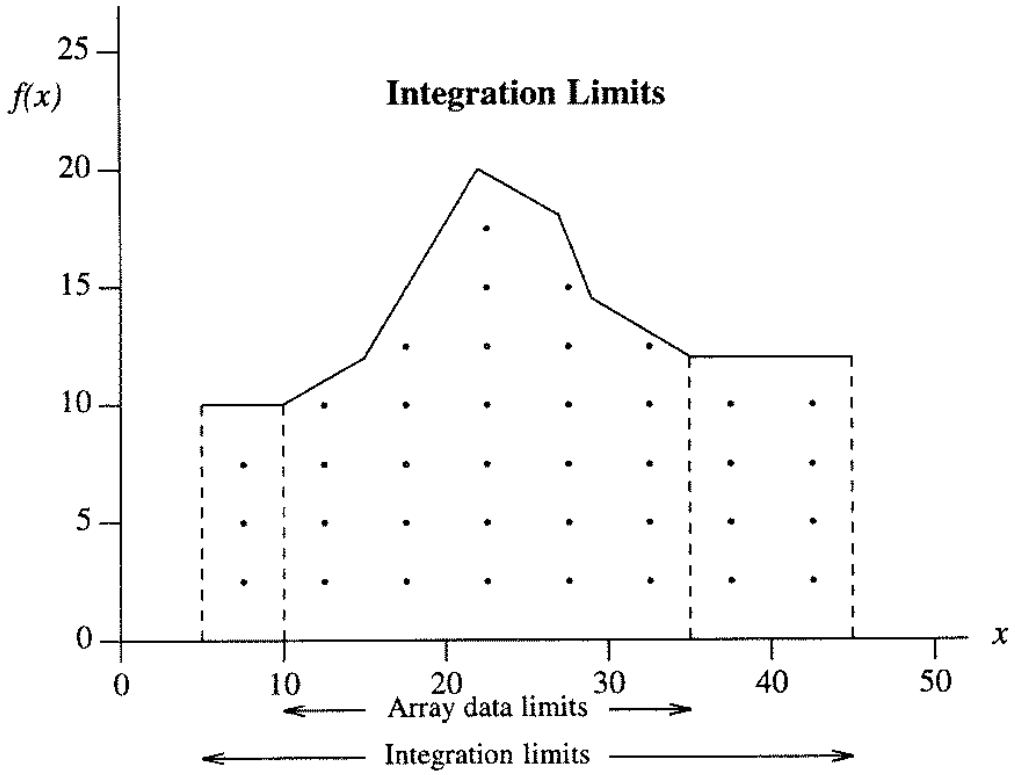
In both INTGL1 and INTEGL the integration is performed for $x_1 < x < x_2$. INTGL2 integrates over the region defined by $x_1 < x < x_2$ and $y_1 < y < y_2$.

Example:

```
$ARRAYS
$REAL
ARR1(2,6) = 10.0, 10.0, 15.0, 10.0, 22.0, 20.0,
            26.0, 18.0, 28.0, 14.0, 35.0, 12.0;
$TABLE
ARR2(X,Y)
X=0.0,  0.0, 7.5, 25.0, 9.5, 50.0, 9.9, 75.0, 8.9, 100.0, 8.0,
X=50.0, 0.0, 6.5, 25.0, 8.9, 50.0, 9.6, 75.0, 8.4, 100.0, 6.0,
X=100.0,0.0, 6.0, 25.0, 7.5, 50.0, 7.9, 75.0, 8.1, 100.0, 7.9;
...
$VARIABLES1
      DOUBLE PRECISION RES1, RES2
      RES1 = INTGL1(22.0D0, 47.0D0, ARR1)
      RES2 = INTGL2(15.0D0, 55.0D0, 25.0D0, 60.0D0, ARR2)
```

Restrictions:

In the diagram below, $y = f(x)$ is defined for $x = x_1, \ldots, x_n$ with X_1, X_2 lying out of this range. The shaded area shows how the integral is calculated.



The limits need not necessarily correspond to given points in the data because linear interpolation is used to find dependent variable values at these points. If, however, the limits lie outside the range of data then the value of the dependent variable at the closest point is used and a warning error is recorded.

6.7 Interpolation

INTCY1 Cyclic Lagrangian interpolation
INTCY2 Cyclic Lagrangian interpolation
INTCY3 Cyclic Lagrangian interpolation
INTCYC Cyclic Lagrangian interpolation over 2 arrays
INTERP Lagrangian interpolation over 2 arrays
INTRP1 Lagrangian interpolation
INTRP2 Lagrangian interpolation
INTRP3 Lagrangian interpolation
INTRPA Lagrangian interpolation

6.7.1 Cyclic Interpolation

Name:

FUNCTION **INTCY1** (X, ARR, N, PERIOD, OFFSET)
FUNCTION **INTCY2** (X, Y, ARR, N, PERIOD, OFFSET)
FUNCTION **INTCY3** (X, Y, Z, ARR, N, PERIOD, OFFSET)
FUNCTION **INTCYC** (X, ARRX, ARRY, N, PERIOD, OFFSET)

X, Y, Z	-	Double precision, independent variables
ARR, ARRX, ARRY	-	Array name (see below for type)
N	-	Integer, order of interpolation
PERIOD	-	Double precision, period of cycle
OFFSET	-	Double precision, offset

Type:

Double precision.

Purpose:

Cyclic Lagrangian interpolation of order N.

ARR, ARRX and ARRY are the names of arrays containing the data on which the interpolation is to be performed.

In INTCY1 the array ARR can be of these types:

i. Table array with one independent variable, e.g. *V(x)*

ii. Real array of dimensions (2, *n*) input as $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$

In INTCY2 and INTCY3 the array ARR must be a table array with two and three independent variables respectively e.g. *V(x, y)* and *V(x, y, z)*.

In INTCYC, ARRX is the name of a real array containing the independent variable values and ARRY the name of a real array containing the dependent variable values.

It is assumed that the dependent variable varies cyclically with one of the independent variables. Values of this independent variable in the appropriate array should lie in the range 0 to PERIOD. The value at which interpolation is performed is determined thus:

X1 = MOD(X + OFFSET, PERIOD),

where *X* is supposed to be the appropriate independent variable.

In the examples of table array input (see Section 3.5) Z would be the independent variable in which the dependent variable is cyclic, as there is a one-to-one

correspondence between them. (The examples show this most clearly.) The user need not, however, have input the variables in this particular order.

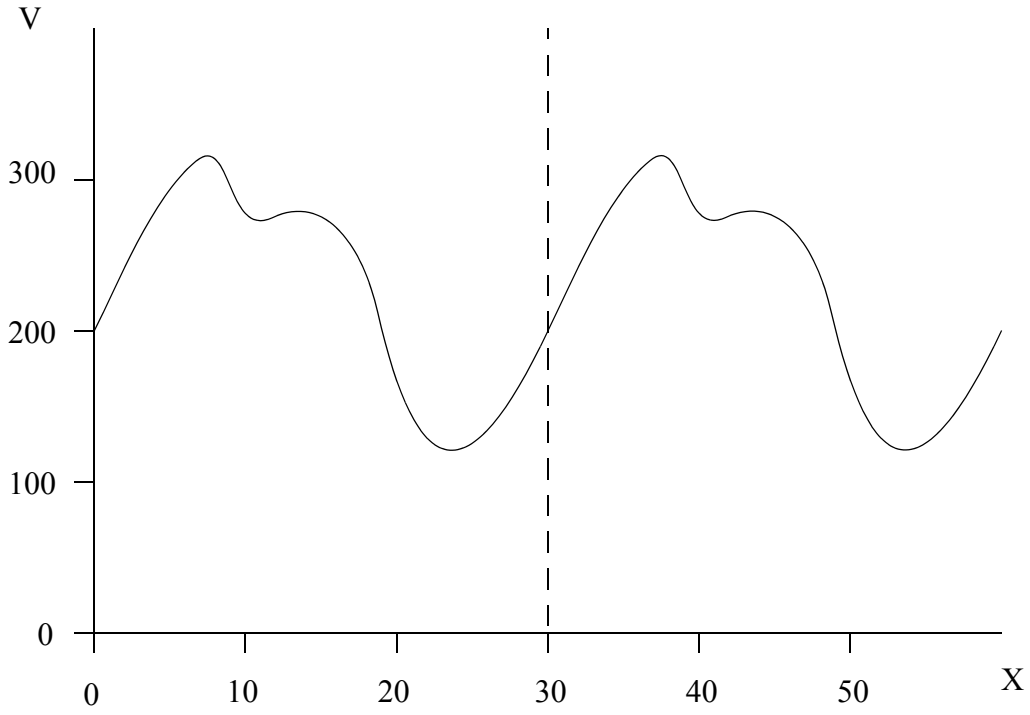
Should an attempt be made to interpolate at a value of independent variable which is out of range, then the nearest value is taken and a warning issued.

Note that interpolation of order 0 implies step changes in the dependent variable.

For higher-order interpolation (N>1), the routines interpolate on a consecutive set of data points which are centred on the input independent variable value. For example, for third-order interpolation—which requires four data points— if the independent variable value lies between the sixth and seventh pair of data values, then the fifth, sixth, seventh and eighth will be used for interpolation. If an odd number of points is required for interpolation, then an additional point after the independent variable value will be taken.

Example:

Consider the graph below where the dependent variable, *V*, is cyclic in independent variable *X* with a period of 30.0.



Suppose we have an array `ARR` which contains data for values of *X* in the range 0 to 30, and we wish to calculate *V* at *X*=45 with an offset of 5:

```
VVAL = INTCY1(45.0D0, ARR, 2, 30.0D0, 5.0D0)
```

The value of *X* at which interpolation is required is given by

X = MOD(45 + 5, 30) = 20

and hence, from the graph, we can see that VVAL takes the value of approximately 150.

Restrictions:

i. INTCYC, INTCY1: N = 0, 1, ..., *n-1*
where *n* is the number of independent variable pairs.
INTCY2: N = 0, 1, 2
INTCY3: N = 0, 1

Should N lie out of range it defaults to the nearest valid value and a warning is issued

ii. All sets of independent variable values must be in increasing order.

6.7.2 Lagrangian Interpolation

Name:

FUNCTION **INTRP1** (X, ARRNAM, N)
FUNCTION **INTRP2** (X, Y, ARRNAM, N)
FUNCTION **INTRP3** (X, Y, Z, ARRNAM, N)
FUNCTION **INTERP** (X, ARRX, ARRY, N)
FUNCTION **INTRPA** (X, ARRNAM, N)

X, Y, Z	-	Double precision, independent variables
ARRX, ARRY, ARRNAM	-	Array name (see below for type)
N	-	Integer, order of interpolation

Type:

Double precision.

Purpose:

Lagrangian interpolation of order N.

ARRNAM, ARRX and ARRY are the names of arrays containing the data on which the interpolation is to be performed.

In INTRP1 the array can be of these types:

i. Table array with one independent variable, e.g. *V(x)*

ii. Real array of dimensions (2, *n*) input as $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$

In INTRP2 and INTRP3 the array must be a table array with two and three independent variables respectively e.g. *V(x, y)* and *V(x, y, z)*.

In INTERP, ARRX is the name of a real array containing the independent variable values and ARRY the name of a real array containing the dependent variable values.

INTRPA is a fixed interval interpolation routine, which allows faster interpolation times due to the fixed format of the input data values. The array on which the interpolation is performed ARRNAM is input as a real array containing values xstart, deltax, y(1), y(2), ..., y(*n*), where:-

xstart	-	x-coordinate of the independent variable
deltax	-	fixed increment value between each data point
y(1), y(2), ..., y(*n*)	-	array of dependent values

Should an attempt be made during the calculations to interpolate at a value of independent variable which is out of range, then the nearest value is taken and a warning issued.

Note that interpolation of order 0 implies step changes in the dependent variable.

For higher-order interpolation (N>1), the routines interpolate on a consecutive set of data points which are centred on the input independent variable value. For example, for third-order interpolation—which requires four data points— if the independent variable value lies between the sixth and seventh pair of data values, then the fifth, sixth, seventh and eighth will be used for interpolation. If an odd number of points is required for interpolation, then an additional point after the independent variable value will be taken.

Example:

```
$ARRAYS
$REAL
ARR1(2,6) =     10.0, 10.0,
                15.0, 10.0,
                22.0, 20.0,
                26.0, 18.0,
                28.0, 14.0,
                35.0, 12.0;
#
$TABLE
ARR2(X,Y)
Y=0.0, 25.0, 50.0, 75.0, 100.0,
X=0.0, 7.5, 9.5, 9.9, 8.9, 8.0,
X=50.0, 6.5, 8.9, 9.6, 8.4, 6.0,
X=100.0, 6.0, 7.5, 7.9, 8.1, 7.9;
...
#
$VARIABLES1
#
      DOUBLE PRECISION RES1, RES2
#
      RES1 = INTRP1(12.0D0, ARR1, 1)
      RES2 = INTRP2(15.0D0, 27.0D0, ARR2, 1)
```

Restrictions:

The following restrictions apply:-

i. INTERP, INTRP1, INTRPA: N = 0, 1, ..., *n-1*
where *n* is the number of independent variable pairs
INTRP2: N = 0, 1, 2
INTRP3: N = 0, 1

Should N lie out of range it defaults to the nearest valid value and a warning is issued.

ii. All sets of independent variable values must be in increasing order.

6.8 Job Management

PROGHD Write out header to progress feedback file
PROGRS Write out data to progress feedback file
ZDAYDT Provide date
ZDAYTM Provide time

6.8.1 Progress Feedback

Name:

SUBROUTINE **PROGHD** (UNIT)
SUBROUTINE **PROGRS** (UNIT**)**

UNIT - INTEGER stream number

Purpose:

These routines are used to write progress feedback information into the file defined by stream number UNIT. PROGHD writes out column headers, whilst PROGRS writes out one line of data for each call. The data are written with fixed column widths as a comma-separated list suitable both for viewing as the analysis progresses and for import to a tool such as Microsoft Excel.

For thermal-only analyses the following data are output: Current real time, solution routine name, TIMEN, TIMEN/TIMEND%, DTIMEU, STEPCT, LOOPCT, NLOOP, RELXCC. For FHTS analyses the following additional data are output: SUMFLL, RELXMC, SUMTHL, FRLXCC.

Note that progress feedback is automatically provided in the *model*.MON file. These routines should only be used if additional output is required. Calling PROGRS with UNIT = 0 will write out to the *model*.MON file.

Example:

```
$INITIAL
      OPEN(UNIT=21, FILE='feedback.fbk')
      CALL PROGHD(21)
$VARIABLES1
      WRITE(21,...
$VARIABLES2
      CALL PROGRS(21)
```

Restrictions:

Output streams 1-20 are reserved for use by ESATAN system files.

6.8.2 Time and Date

Name:

FUNCTION **ZDAYDT()**
FUNCTION **ZDAYTM()**

Type:

ZDAYDT - character * 17
ZDAYTM - character * 8

Purpose:

Functions to return current date and time of day. These are obtained from system calls.

ZDAYDT returns the date in the form:-

Date Month Year

e.g. 25 DECEMBER 1986

ZDAYTM returns the time as:-

Hour:Minute:Second

e.g. 15:36:36

Example:

```
$OUTPUTS
      CHARACTER MYDATE*17, MYTIME*8
      MYDATE = ZDAYDT()
      MYTIME = ZDAYTM()
```

Restrictions:

None.

6.9 Matrix Functions

MATCML	Multiply columns by vector
MATCMN	Find minimum element of column
MATCMX	Find maximum element of column
MATCSM	Sum matrix columns
MATDIA	Set matrix to diagonal matrix
MATDTI	Determine determinant of integer matrix
MATDTR	Determine determinant of real matrix
MATIDN	Set matrix to identity matrix
MATINV	Inverse matrix
MATMLT	Multiply arrays
MATRML	Multiply rows by vector
MATRSM	Sum matrix rows
MATSMI	Sum integer matrix elements
MATSMR	Sum real matrix elements
MATTRN	Transpose matrix
MCAPRG	Set column to arithmetic progression
MCCNST	Set column to constant value
MCGPRG	Set column to geometric progression
MRAPRG	Set row to arithmetic progression
MRCNST	Set row to constant value
MRGPRG	Set row to geometric progression

All named arrays must be predefined, and of consistent arithmetic type.

6.9.1 Multiply Rows/Columns by Vector

Name:

SUBROUTINE **MATCML** (ANAME1, ANAME2, ANAME3)
SUBROUTINE **MATRML** (ANAME1, ANAME2, ANAME3)

ANAME1, ANAME2, ANAME3 - Real/Integer array names.

Purpose:

Multiply each column/row of ANAME1 by vector in ANAME2 (MATCML/MATRML respectively) and store resultant matrix in ANAME3.

Examples:

Given

$$\text{ANAME1} = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 2 \\ 1 & 0 & 1 \end{bmatrix}, \qquad \text{ANAME2} = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

then

```
CALL MATCML(ANAME1, ANAME2, ANAME3)
```

gives

$$\text{ANAME3} = \begin{bmatrix} 1 & 6 & 0 \\ 2 & 3 & 2 \\ 1 & 0 & 1 \end{bmatrix}$$

and

```
CALL MATRML(ANAME1, ANAME2, ANAME3)
```

gives

$$\text{ANAME3} = \begin{bmatrix} 1 & 2 & 0 \\ 6 & 3 & 6 \\ 1 & 0 & 1 \end{bmatrix}$$

Restrictions:

Arrays ANAME1 and ANAME3 must be two dimensional and ANAME2 a one dimensional array. The dimension of the vector ANAME2 must equal the number of columns/rows of ANAME1 using MATCML/MATRML respectively. The

declared size of ANAME3 must be greater than or equal to the number of elements in ANAME1.

Note:

Due to the FORTRAN convention for multi-dimensional arrays care has to be taken when defining matrices. The following array definition:-

```
REAL*ARR(3,3) =
       a11,a21,a31,
       a12,a22,a32,
       a13,a23,a33;
```

represents the matrix:-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.9.2 Maximum/Minimum Column Values

Name:

SUBROUTINE **MATCMX** (ANAME1, ANAME2)
SUBROUTINE **MATCMN** (ANAME1, ANAME2)

ANAME1, ANAME2 - Real/Integer array names.

Purpose:

Find maximum/minimum element value in column of ANAME1 (MATCMX/MATCMN respectively) and store resultant vector in ANAME2.

Examples:

Given

$$\text{ANAME1} = \begin{bmatrix} 1 & 2 & 5 \\ 2 & 4 & 0 \\ 1 & 9 & 1 \end{bmatrix},$$

then

```
CALL MATCMX(ANAME1, ANAME2)
```

gives

$$\text{ANAME2} = \begin{bmatrix} 2 \\ 9 \\ 5 \end{bmatrix}$$

and

```
CALL MATCMN(ANAME1, ANAME2)
```

gives

$$\text{ANAME2} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

Restrictions:

ANAME1 must be a two dimensional array and ANAME2 a one dimensional array. The size of array ANAME2 must be greater than or equal to the number of columns of ANAME1.

Note:

Due to the FORTRAN convention for multi-dimensional arrays care has to be taken when defining matrices. The following array definition:-

```
REAL*ARR(3,3) =
       a11,a21,a31,
       a12,a22,a32,
       a13,a23,a33;
```

represents the matrix:-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.9.3 Sum Matrix Columns/Rows

Name:

SUBROUTINE **MATCSM** (ANAME1, ANAME2)
SUBROUTINE **MATRSM** (ANAME1, ANAME2)

ANAME1, ANAME2 - Real/Integer array names

Purpose:

Evaluate the sum of element values for each column (MATCSM) or row (MATRSM) of ANAME1, storing the resultant vector in ANAME2.

Examples:

Given

$$\text{ANAME1} = \begin{bmatrix} 1 & 2 & 5 \\ 2 & 4 & 0 \\ 1 & 9 & 1 \end{bmatrix},$$

then

```
CALL MATCSM(ANAME1, ANAME2)
```

gives

$$\text{ANAME2} = \begin{bmatrix} 4 \\ 15 \\ 6 \end{bmatrix}$$

and

```
CALL MATRSM(ANAME1, ANAME2)
```

gives

$$\text{ANAME2} = \begin{bmatrix} 8 \\ 6 \\ 11 \end{bmatrix}$$

Restrictions:

ANAME1 must be a two dimensional array and ANAME2 a one dimensional array. The size of the vector ANAME2 must be larger than or equal to the number of columns/rows of ANAME1 when using MATCSM/MATRSM respectively.

Note:

Due to the FORTRAN convention for multi-dimensional arrays care has to be taken when defining matrices. The following array definition:-

```
REAL*ARR(3,3) =
       a11,a21,a31,
       a12,a22,a32,
       a13,a23,a33;
```

represents the matrix:-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.9.4 Diagonal Matrix

Name:

SUBROUTINE **MATDIA** (ANAME1, ANAME2)

ANAME1, ANAME2 - Real/Integer array names.

Purpose:

Create diagonal matrix ANAME2 from vector stored in ANAME1.

Example:

Given

$$\text{ANAME1} = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

then

```
CALL MATDIA(ANAME1, ANAME2)
```

gives

$$\text{ANAME2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Restrictions:

Array ANAME1 must be two dimensional and array ANAME2 must be one dimensional.

Note:

Due to the FORTRAN convention for multi-dimensional arrays care has to be taken when defining matrices. The following array definition:-

```
REAL*ARR(3,3) =
       a11,a21,a31,
       a12,a22,a32,
       a13,a23,a33;
```

represents the matrix:-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.9.5 Matrix Determinant

Name:

FUNCTION **MATDTR** (ARRNAM)
FUNCTION **MATDTI** (ARRNAM)

ARRNAM - Real/Integer array name

Type:

MATDTR - double precision
MATDTI - integer

Purpose:

Evaluate the determinant of a real/integer array (MATDTR/MATDTI respectively).

Examples:

```
RDET = MATDTR(RARR)
IDET = MATDTI(IARR)
```

Restrictions:

ARRNAM must be a two dimensional square matrix. It requires ((4 * D + 2) * D) locations of dynamic core (D = first dimension of ARRNAM).

Note:

Due to the FORTRAN convention for multi-dimensional arrays care has to be taken when defining matrices. The following array definition:-

```
REAL*ARR(3,3) =
       a11,a21,a31,
       a12,a22,a32,
       a13,a23,a33;
```

represents the matrix:-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.9.6 Identity Matrix

Name:

SUBROUTINE **MATIDN** (ARRNAM)

ARRNAM - Real/Integer array name

Purpose:

Set ARRNAM equal to the identity matrix.

Examples:

```
CALL MATIDN(IARR)
CALL MATIDN(RARR)
```

Restriction:

ARRNAM must be a square two-dimensional array.

6.9.7 Matrix Inverse

Name:

SUBROUTINE **MATINV** (ANAME1, ANAME2)

ANAME1, ANAME2 - Real array names

Purpose:

Invert ANAME1 and store in ANAME2. Uses Gaussian elimination.

Example:

```
CALL MATINV(ANAME1, ANAME2)
```

Restrictions:

Arrays ANAME1 and ANAME2 must be two dimensional and array ANAME1 must also be a square matrix. The declared size of array ANAME2 must be greater than or equal to the number of elements in array ANAME1. It requires ((4 * D + 2) * D) locations of dynamic core (D = first dimension of ANAME1).

Note:

Due to the FORTRAN convention for multi-dimensional arrays care has to be taken when defining matrices. The following array definition:-

```
REAL*ARR(3,3) =
       a11,a21,a31,
       a12,a22,a32,
       a13,a23,a33;
```

represents the matrix:-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.9.8 Matrix Multiplication

Name:

SUBROUTINE **MATMLT** (ANAME1, ANAME2, ANAME3)

ANAME1, ANAME2, ANAME3 - Real/Integer array names.

Purpose:

Multiply arrays ANAME1 and ANAME2, and store the result in ANAME3. This resultant matrix will be redimensioned, if appropriate, only if the declared size of array ANAME3 is greater than or equal to the resultant matrix. The dimensions of arrays ANAME1 and ANAME2 should be such that this multiplication is mathematically possible.

Example:

```
CALL MATMLT(ANAME1, ANAME2, ANAME3)
```

Restrictions:

Arrays must be two dimensional.

Note:

Due to the FORTRAN convention for multi-dimensional arrays care has to be taken when defining matrices. The following array definition:-

```
REAL*ARR(3,3) =
       a11,a21,a31,
       a12,a22,a32,
       a13,a23,a33;
```

represents the matrix:-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.9.9 Sum Matrix Elements

Name:

FUNCTION **MATSMR** (ARRNAM)
FUNCTION **MATSMI** (ARRNAM)

ARRNAM - Real/Integer array name

Type:

MATSMR - double precision
MATSMI - integer

Purpose:

Evaluate the sum of all elements of a real/integer array (MATSMR/MATSMI respectively).

Examples:

```
RSUM = MATSMR(RARR)
ISUM = MATSMI(IARR)
```

Restrictions:

ARRNAM must be a two dimensional array.

Note:

Due to the FORTRAN convention for multi-dimensional arrays care has to be taken when defining matrices. The following array definition:-

```
REAL*ARR(3,3) =
       a11,a21,a31,
       a12,a22,a32,
       a13,a23,a33;
```

represents the matrix:-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.9.10 Matrix Transpose

Name:

SUBROUTINE **MATTRN** (ANAME1, ANAME2)

ANAME1, ANAME2 - Real/Integer array names

Purpose:

Transpose matrix ANAME1 and store in ANAME2.

Example:

```
CALL MATTRN(ANAME1, ANAME2)
```

Restrictions:

Arrays ANAME1 and ANAME2 must be two dimensional arrays. The declared size of array ANAME2 must be greater than or equal to the number of elements in array ANAME1.

Note:

Due to the FORTRAN convention for multi-dimensional arrays care has to be taken when defining matrices. The following array definition:-

```
REAL*ARR(3,3) =
      a11,a21,a31,
      a12,a22,a32,
      a13,a23,a33;
```

represents the matrix:-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.9.11 Set Column/Row to AP/GP

Name:

SUBROUTINE **MCAPRG** (ARRNAM, N, VAL, INC)
SUBROUTINE **MRAPRG** (ARRNAM, N, VAL, INC)
SUBROUTINE **MCGPRG** (ARRNAM, N, VAL, INC)
SUBROUTINE **MRGPRG** (ARRNAM, N, VAL, INC)

ARRNAM	-	Real/Integer array name
N	-	Integer column/row value
VAL	-	Double precision/Integer starting value
INC	-	Double precision/Integer increment

Purpose:

Set all elements in column/row number N to arithmetic/geometric progression. For arithmetic progression, MCAPRG/MRAPRG for columns/rows respectively, likewise MCGPRG/MRGPRG for columns/rows in geometric progression.

Examples:

```
CALL MCAPRG(RARR, 3, 1.0D0, 0.5D0)
CALL MRAPRG(IARR, 2, 5, 2)
CALL MCGPRG(RARR, 1, 2.0D0, 0.5D0)
CALL MRGPRG(IARR, 4, 2, 3)
```

Restrictions:

VAL and INC must be of same arithmetic type as ARRNAM. Array ARRNAM must be two dimensional.

Note:

Due to the FORTRAN convention for multi-dimensional arrays care has to be taken when defining matrices. The following array definition:-

```
REAL*ARR(3,3) =
       a11,a21,a31,
       a12,a22,a32,
       a13,a23,a33;
```

represents the matrix:-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.9.12 Set Column/Row to Constant

Name:

SUBROUTINE **MCCNST** (ARRNAM, N, VAL)
SUBROUTINE **MRCNST** (ARRNAM, N, VAL)

ARRNAM	-	Real/Integer array name
N	-	Integer column/row value
VAL	-	Double precision/Integer value

Purpose:

Set all elements in column/row number N to VAL (MCCNST/MRCNST respectively).

Examples:

```
CALL MCCNST(RARR, 3, 1.0D0)
CALL MRCNST(IARR, 2, 5)
```

Restrictions:

ARRNAM and VAL must be of same arithmetic type. Array ARRNAM must be two dimensional.

Note:

Due to the FORTRAN convention for multi-dimensional arrays care has to be taken when defining matrices. The following array definition:-

```
REAL*ARR(3,3) =
       a11,a21,a31,
       a12,a22,a32,
       a13,a23,a33;
```

represents the matrix:-

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.10 Output

CSGDMP Print a CSG-dump of model
HUMDMP Humidity information dump
MODCHK Verify model and print warnings
PRHEAD Print page header information
PRNCSV Comma-separated list output of entities
PRNDBL Block output of entities
PRNDPT Table output of fluid properties
PRNDTB Table output of entities
PRPAGE Skip to new page
PRQBAL Print heat imbalance of node
PRQBOU Print heat flow over model boundary
PRQG Print heat flow between two nodes
PRQLIN Print heat flow over linear conductors
PRQNOD Print heat flow to node
PRTARR Print array elements
PRTGRP Print characteristic data for a group of nodes
PRTNAV Print average of entities
PRTNSM Print sum of entities
PRTTMD Print maximum temperature difference
PRTUC Print user constants
PTSINK Print sink temperatures
QRATES Print thermal balance for selected nodes

6.10.1 Introduction

Formatting

The user may choose the format with which double precision numbers are printed in all output routines other than PRNDTB. To do this he must set the character*8 global control constant, **FORMAT**. A default format of F10.2 is used if **FORMAT** is not set. **FORMAT** should not be assigned to any format resulting in an output field width of more than ten characters. Three other global control constants, **TFORM**, **TFORMF** and **THEAD** are available for formatting table output of node properties. These are described in Section 6.10.9.

Job Title

The global character control constant **HEADER** is available for user definition in the operations blocks, and is printed by the output routine PRPAGE at the heading of each page of output.

Sub-Model Code

When printing values of conductances and heat flow a sub-model code is used. Rather than giving the full concatenation of a node's sub-model it is printed in a coded form which comprises a letter "Z" immediately followed by an integer. A key for this code is given at the end of the print-out. A typical output would be:

```
HEAT FLOW IN CONDUCTORS ATTACHED TO NODES OF ZLABEL = "TOP"
ENGINE : EXHAUST
LINEAR HEAT FLOW:
(10, 20) - 0.1 , (10, Z2 : 150) - 0.2 , (50, Z3 : 10) - 1.5
KEY FOR SUB-MODEL CODE:
Z2 =
ENGINE : EXHAUST : HEATPIPE
Z3 =
ENGINE : EXHAUST : CAPILLARY
```

NB: In this example, all nodes without a preceding sub-model descriptor are in submodel ENGINE:EXHAUST.

6.10.2 CSG Dump

Name:

SUBROUTINE **CSGDMP** (CNAME)

CNAME - Sub-model name (see Section 6.1.3)

Purpose:

Outputs capacitance, conductance values and CSG for all diffusion nodes of submodel CNAME (and all included submodels unless :ONLY is specified). Also outputs CSGMIN value and node number for each submodel. Where present, FHTS fluid nodes are included in the calculation.

Example:

```
CALL CSGDMP(SUBMOD:ONLY)
```

Restrictions:

VARIABLES1 must have been called at least once before a call to CSGDMP can be made, unless all relevant nodal parameters are defined explicitly.

6.10.3 Humidity Dump

Name:

SUBROUTINE **HUMDMP** (CNAME)

CNAME - Sub-model name (see Section 6.1.3)

Purpose:

Outputs relative humidity (PHI), specific humidity (SPHUM), and water-vapour source- and sink-rate (WVSRC and WVSNK, respectively) for each moist-air node in a given submodel, as well as condensation rate (MCOND) and latent-heat transfer rate (QCOND) along each GL conductor attached to these nodes.

Example:

```
CALL HUMDMP(LOOP1)
```

Restrictions:

Can only be used in conjunction with fluid type AIRW. The $VARIABLES1 block must have been called at least once before a call to HUMDMP can be made.

6.10.4 Model Check

Name:

SUBROUTINE **MODCHK** (ZLABEL, CNAME)

ZLABEL - Character (see Section 6.1.3)
CNAME - Sub-model name (see Section 6.1.3)

Purpose:

To print out warnings about unconnected solid and fluid nodes and any undefined nodal entities within a model definition. The routine also checks for the presence of a boundary node within any given fluid loop and ascertains whether all nodal fluid types within a single loop are identical. A warning is issued if mass sources are defined at boundary nodes and also if the net mass source within a loop is not balanced by commensurate mass sinks. Any undefined or zero values of FLA, GP or any negative values of fluid nodal volume will also result in warnings being given to the user. In the case of solid nodes, if the maximum value of conductance connected to a node is more than one hundred times the minimum conductance connected to it, then a warning is issued.

As well as the warnings which it writes out, modchk also writes out information on CSG and COURANT numbers within the model: the range of these values is determined and then divided into ten equal intervals, the routine then writes out details of these intervals and the number of nodes within each one.

MODCHK may be called from any of the operations blocks. However, the generated Mortran needs to be executed prior to the call to MODCHK so that all capacitance and conductance values are set up.

Examples:

```
$EXECUTION
      CALL VARIABLES1
      CALL MODCHK(' ',CURRENT)
```

or

```
$VARIABLES1
      GENMOR
      CALL MODCHK(' ', CURRENT)
```

or

```
$OUTPUTS
      CALL MODCHK(' ', CURRENT)
```

Restrictions:

At present MODCHK will only carry out its checking of fluid loops correctly if tee-piece elements are used in their definition.

6.10.5 Page and Table Headers

Name:

SUBROUTINE **PRPAGE**
SUBROUTINE **PRHEAD**

Purpose:

PRPAGE skips to a new page and prints the following: program name, version number, run date, run time and job name.

PRHEAD prints the following information:-

TIMEN	-	solution time (all solution routines)
MODULE	-	solution routine (all solution routines)
ENBALA	-	absolute energy balance (steady state)
ENBALR	-	relative energy balance (steady state)
LOOPCT	-	number of iterations (steady state)
DTIMEU	-	time step (transient)
CSGMIN	-	(transient)
NCSGMN	-	node at which CSGMIN is found (transient)
DTCOUR	-	Courant timestep limit (Fluid transient)
RELXMC	-	Mass imbalance ratio on network (Fluid transient)

PRPAGE and PRHEAD are called at the start of all other output routines and PRPAGE is called for every subsequent page.

Example:

```
CALL PRPAGE
CALL PRHEAD
```

6.10.6 Comma-Separated Value Output of ZENTS

Name:

SUBROUTINE **PRNCSV** (ZLABEL, ZENTS, CNAME, ORDER, FILE)

ZLABEL	-	Character (see Section 6.1.3)
ZENTS	-	Character (see Section 6.1.3)
CNAME	-	Sub-model name (see Section 6.1.3)
ORDER	-	Character Output order
FILE	-	Character Output file name

Purpose:

To print entities in ZENTS for all nodes of ZLABEL in submodel CNAME to the file FILE. The data are ordered according to ORDER. If ORDER is 'ENTITY' then the data are ordered by entity type, if ORDER is 'NODE' then the data are ordered by node number. Column headers are written, and the data are output as a comma-separated value list, appropriate for import into a post-processing tool such as a spreadsheet.

Examples:

```
CALL PRNCSV(' ', 'T', CURRENT, 'NODE', 'temperatures.csv')
```

This call requests a printout of temperature (T) for nodes in the current submodel and its included submodels, in increasing node number order.

```
 CALL PRNCSV('#1-10', 'GL,GR', SUB1, 'ENTITY',
&            'conductors.csv')
```

This outputs values of linear and radiative conductors connected to any of nodes 1 to 10 in submodel SUB1. The GL conductors are output first and then the GR conductors.

Restrictions:

This routine always requires NMOD locations of dynamic core, where NMOD = number of nodes selected from ZLABEL and CNAME. For conductances, a further 6 × (number of conductors) locations are required.

Note that the popular Microsoft Excel spreadsheet application is limited to 256 columns. PRNCSV is able to output files with greater than 256 columns, but a warning message may be issued in this case but is not output by default.

No more than 1000 files can be defined for CSV output in a single run.

The number of data columns in the output CSV file cannot exceed 100,000.

6.10.7 Block Output of ZENTS

Name:

SUBROUTINE **PRNDBL** (ZLABEL, ZENTS, CNAME)

ZLABEL	-	Character (see Section 6.1.3)
ZENTS	-	Character (see Section 6.1.3)
CNAME	-	Sub-model name (see Section 6.1.3)

Purpose:

To print entities in ZENTS for all nodes of ZLABEL in submodel CNAME.

Examples:

```
CALL PRNDBL(' ', 'T-INC, T, A, QE', CURRENT)
```

This call requests a printout of T, A and QE, in increasing order of temperature, for nodes in the current submodel and its included submodels.

If conductances are required in order then a conductor will appear twice if both linked nodes are referenced. A letter "X" after a conductance value indicates that the conductor is turned off (see also Section 6.1.3).

```
CALL PRNDBL(' ', 'TempMax-INC, TempMax, T, QE', CURRENT)
```

This call orders the output of TempMax, T and QE in increasing order of the spare nodal entity TempMax, which may be calculated by the user in any operations block.

Restrictions:

This routine always requires NMOD locations of dynamic core, where NMOD = number of nodes selected from ZLABEL and CNAME. For nodal entities, a further NMOD locations are required if either INC or DEC is used, and for conductances, a further 6 × (number of conductors) locations are required.

6.10.8 Output of Fluid Properties

Name:

SUBROUTINE **PRNDPT**(ZLABEL, ZPROP, CNAME)

ZLABEL	-	Character (see Section 6.1.3)
ZPROP	-	Character (see below)
CNAME	-	Sub-model name (see Section 6.1.3)

Purpose:

To print a table of properties for fluid nodes of ZLABEL in submodel CNAME. The following entities are available for inclusion in ZPROP:-

RHO	density
COND	thermal conductivity
CP	specific heat at constant pressure
VISC	dynamic viscosity
QUAL	vapour quality or dryness fraction
VFLO	volumetric flow rate
U	velocity
RE	Reynolds number

PRNDPT will use a default output format unless the control constants TFORM, TFORMF and THEAD are set. For an explanation of these, see the entry for PRNDTB (Section 6.10.9).

Examples:

```
$OUTPUTS
      CALL PRNDPT(' ' , 'RHO,COND,RE' , CURRENT)
      CALL PRNDPT('LOOP1' , 'CP,RHO' , TBUS:ONLY)
```

6.10.9 Table Output of ZENTS

Name:

SUBROUTINE **PRNDTB** (ZLABEL, ZENTS, CNAME)

ZLABEL	-	Character (see Section 6.1.3)
ZENTS	-	Character (see Section 6.1.3)
CNAME	-	Sub-model name (see Section 6.1.3)

Purpose:

To print a table of entities in ZENTS (node properties only) for all nodes of ZLABEL in submodel CNAME.

The user has a choice of setting a format for the output or allowing the program to construct one. To choose the first option he must set the character global control constants TFORM (Thermal nodes) and TFORMF (Fluid nodes) to the required format. If mixing fluid node properties with thermal node properties then TFORM is used for thermal nodes whilst TFORMF is used for fluid node properties. TFORM must include spaces for the fluid entities whilst TFORMF has to contain spaces for the thermal entities. This can be seen in the example.

If the user does set TFORM he must specify a format for the table heading in THEAD, also a character global control constant. It is important to note that these formats must include a format for the integer node number at the beginning of the line. Character control constants can only be set in the operations blocks. Care is required when setting these control constants as they are not checked by the preprocessor nor by the library. Should there be a mistake in ZENTS, however, these control constants will not be used in case of a possible Fortran I/O error. TFORM, TFORMF and THEAD, if previously set, should be reset to '*' if the default formats are required on a subsequent occasion.

All double precision numbers are output using F10.2 by default, and integer spare nodal entities using I10 by default.

Examples:

For a thermal-only network:

```
TFORM = '(1X, I5, 3(2X, F8.3), 2(2X, F8.5))'
THEAD = ' NODE       T          C         QA        EPS      ALP'
CALL PRNDTB(' ', 'T, C, QA, EPS, ALP, QE-DEC', THROTTLE:FUELINLET)
```

Or, for a combined thermal/fluid network:

```
TFORM = '(1X, I4, 2(5X, F8.4), 30X)'
TFORMF = '(1X, I4, 5X, F8.4, 13X, 2(5X, F10.6))'
THEAD = ' NODE       TEMP          QI         PRESS       FQ'
CALL PRNDTB(' ', 'T, QI, P, FQ', CURRENT)
```

Or, including user-defined nodal entities:

```
TFORM = '(1X, I5, 2X, F8.3, 2X, I8, 2X, 2X, A12, 12X)'
TFORMF = '(1X, I5, 2X, F8.3, 2X, I8, 2X, 2X, A12, 2X, F10.6)'
THEAD = ' NODE      TEMP    Int_Ent_Ab    Char_12chars   PRESS'
CALL PRNDTB(' ', 'T, Int_Ent_Ab, Char_12chars, PRESS', CURRENT)
```

Restrictions:

PRNDTB may only be used for the output of node property values i.e. no conductor may be specified in ZENTS. It requires (2 * NUMREF) locations of dynamic core if INC or DEC is used (NUMREF = number of nodes in ZLABEL). Note that FORMAT (see Section 6.10) will not be used in PRNDTB even if TFORM is not set.

6.10.10 Heat Imbalances

Name:

SUBROUTINE **PRQBAL** (ZLABEL, CNAME)

ZLABEL - Character (see Section 6.1.3)
CNAME - Sub-model name (see Section 6.1.3)

Purpose:

To print the heat imbalances of all nodes of ZLABEL in submodel CNAME. The heat imbalance on a node is calculated as the sum of all heat flows into the node.

Example:

```
$OUTPUTS
      CALL PRQBAL(' ', CURRENT)
```

6.10.11 Print Heat Flow

Name:

SUBROUTINE **PRQBOU** (CNAME)
SUBROUTINE **PRQLIN** (CNAME1, CNAME2)
SUBROUTINE **PRQG** (D[:CNAME1:]NODE1, D[:CNAME2:]NODE2)
SUBROUTINE **PRQNOD** (ZLABEL, CNAME)

ZLABEL	-	Character (see Section 6.1.3)
CNAME1, CNAME2	-	Sub-model name (see Section 6.1.3)
D[:CNAME:]NODE	-	[Sub-]model node number

Purpose:

PRQBOU prints the heat flows along all conductors in submodel CNAME which cross the submodel boundary.

PRQLIN prints the heat flows along all conductors linking CNAME1 and CNAME2. If CNAME2 is an included submodel of CNAME1 then CNAME1 is regarded as all submodels within CNAME1 except those within CNAME2.

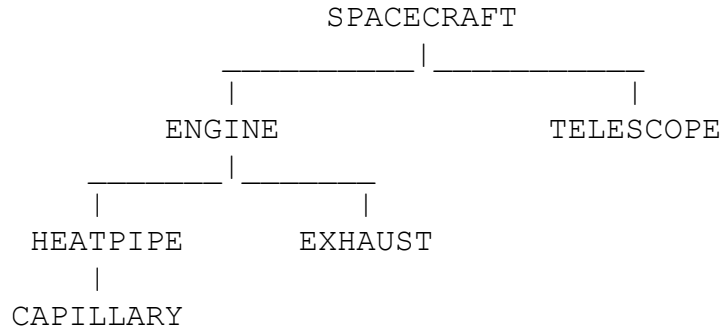
PRQG prints the heat flows along all conductors linking D[:CNAME1:]NODE1 and D[:CNAME2:]NODE2.

PRQNOD prints the heat flows along all conductors attached to nodes of ZLABEL in CNAME. If both nodes attached to a conductor are referenced by the call then the conductor will appear twice.

In all of these routines the convention employed is that heat flowing into the first named node is positive.

Examples:

Consider the following sub-model structure:-




```
$OUTPUTS
      CALL PRQBOU (ENGINE)
```

This call will print the heat flows along all conductors for which one attached node is in either ENGINE, HEATPIPE, CAPILLARY or EXHAUST, and the other is in either SPACECRAFT or TELESCOPE.

```
$OUTPUTS
      CALL PRQBOU (ENGINE : ONLY)
```

This call will print the heat flows along all conductors for which one attached node is in ENGINE and the other is in any sub-model except ENGINE.

```
$OUTPUTS
      CALL PRQLIN (ENGINE, HEATPIPE:ONLY)
```

Here, a conductor is referenced if one attached node is in HEATPIPE and the other is in ENGINE, CAPILLARY or EXHAUST.

```
$OUTPUTS
      CALL PRQG(D5, D:EXHAUST:3)
```

```
$OUTPUTS
      CALL PRQNOD('#1,5,11-15', TELESCOPE)
```

6.10.12 Print Array

Name:

SUBROUTINE **PRTARR** (ARRNAM)

ARRNAM - Array name

Purpose:

Print array elements. Element identifiers are at the start of every line and refer to the first element on the line.

Example:

```
$OUTPUTS
      CALL PRTARR(MYARR)
      CALL PRTARR(SUBMOD:MYARR2)
```

Restriction:

This routine cannot be used for reporting table arrays.

6.10.13 Output of Characteristic Data for Group of Nodes

Name:

SUBROUTINE **PRTGRP** (ZLABEL, CNAME)

ZLABEL - Character, node group (see Section 6.1.3)
CNAME - Concatenated model name (see Section 6.1.3)

Purpose:

To print out salient data for the nodes of a group—minimum and maximum temperature, total capacitance, total impressed heat flux by type, and total net heat flux between the group and the rest of the model by conductor type. For fluid nodes, minimum and maximum pressure will also be output.

Example:

```
$OUTPUTS
    CALL PRTGRP('#1-3;sub1:3', CURRENT)
```

6.10.14 Sums/Averages of Entities

Name:

SUBROUTINE **PRTNSM** (ZLABEL, ZENTS, CNAME)
SUBROUTINE **PRTNAV** (ZLABEL, ZENTS, CNAME)

ZLABEL	-	Character (see Section 6.1.3)
ZENTS	-	Character (see Section 6.1.3)
CNAME	-	Sub-model name (see Section 6.1.3)

Purpose:

PRTNSM prints the sums of the entities in ZENTS for nodes of ZLABEL in each submodel referenced by CNAME.

PRTNAV prints the arithmetic mean of the entities, with the mean of any integer spare nodal property expressed as real. In both of these routines a conductance will be accounted for once, and only once, if either or both of its attached nodes are referenced.

Examples:

```
$OUTPUTS
      CALL PRTNAV('SCOPE', 'T, A', CURRENT:ONLY)
```

This will print the average of the temperatures and the average of the areas of the nodes in the current submodel which have SCOPE as a substring of their labels.

Similarly,

```
$OUTPUTS
      CALL PRTNSM('SCOPE', 'A, Damage', CURRENT:ONLY)
```

will print the sums of the nodal areas (A) and of the user-defined nodal entity Damage.

6.10.15 Temperature Differences

Name:

SUBROUTINE **PRTTMD** (ZLABEL, CNAME)

ZLABEL - Character (see Section 6.1.3)
CNAME - Sub-model (see Section 6.1.3)

Purpose:

Prints the maximum temperature difference between nodes of ZLABEL in each submodel referenced by CNAME.

Example:

```
$OUTPUTS
      CALL PRTTMD(' ', SUBMOD:ONLY)
```

6.10.16 User Constants

Name:

SUBROUTINE **PRTUC** (CNAME)

CNAME - Sub-model name (see Section 6.1.3)

Purpose:

Prints names and values of all user constants defined in submodel CNAME.

Example:

```
$OUTPUTS
      CALL PRTUC(CURRENT)
```

6.10.17 Sink Temperature Output

Name:

SUBROUTINE **PTSINK** (ZLABEL1, CNAME1, ZLABEL2, CNAME2, TYPE)

ZLABEL1	-	Character Thermal group (see Section 6.1.3)
CNAME1	-	Submodel name for thermal group (see Section 6.1.3)
ZLABEL2	-	Character Environment group (see Section 6.1.3)
CNAME2	-	Submodel name for environment group (see Section 6.1.3)
TYPE	-	Integer Type of sink temperature required

Purpose:

To print the sink temperature between a node (or group of nodes) of ZLABEL1 to an environment defined by a node (or a group of nodes) of ZLABEL2. If undefined, the result is given as 'U'.

TYPE will define which kind of sink temperature calculation is to be used. It has to be one of the four following integers:-

1 Black body radiation sink temperature $T_{S,bbr}$
2 Grey body radiation sink temperature $T_{S,gbr}$
3 Radiative sink temperature $T_{S,r}$
4 Linear sink temperature $T_{S,l}$

For a definition and formulation of the sink temperature, see Section 6.4.32.

Example:

```
$OUTPUTS
#
# Calculation of the Grey body sink temperature for the
# thermal group composed of nodes from 1 to 12 to the
# environment group composed of nodes from 1 and 2 of
# submodel SUB2
#
      CALL PTSINK('#1-12', CURRENT, '#1-2', SUB2, 2)
#
```

Restrictions:

Accepts thermal nodes only. Ignores conductors attached to inactive nodes in calculation. Boundary nodes can only be part of the environment group, not the thermal group. Requires 2 * NUMREF1 + 2 * NUMREF2 locations of dynamic core, where NUMREF1 is the number of nodes of ZLABEL1 and NUMREF2 is the number of nodes of ZLABEL2.

6.10.18 Output of Thermal Balance

Name:

SUBROUTINE **QRATES** (ZLABEL, CNAME)

ZLABEL - Character (see Section 6.1.3)
CNAME - Sub-model name (see Section 6.1.3)

Purpose:

To print for each node temperatures, conductances, heat sources, and resultant heat fluxes giving the thermal balance for selected nodes of ZLABEL in CNAME. The user may change the output format using the control constant FORMAT though the output will only be aligned when the width of the format is 10.

Example:

```
$OUTPUTS
      CALL QRATES (' ' , CURRENT)
```

This call requests a print-out of temperature, conductances, heat sources and resultant heat fluxes for all nodes in the current submodel and its included submodels.

6.11 Polynomial Functions

LSTSQ Perform least square fit
POLYNM Calculate polynomial equation

6.11.1 Least Squares Fit

Name:

SUBROUTINE **LSTSQ** (ARR1, N, E, ARR2)

ARR1, ARR2	-	Array names
N	-	Integer order of least square fit
E	-	Double precision square error

Purpose:

Least square fit to points in array ARR1, to order N. This array may be of two types:-

i.Table array with one independent variable

iiReal array of dimensions (2, n) input as $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$.

The subroutine calculates $a_0, a_1, \ldots, a_n$ for the fit $f(x) = a_0 + a_1 x + \ldots + a_n x^n$ and puts the coefficients, in that order, into array ARR2.

The square error, $E = \sum_{i=1}^{n} (f(x_i) - y_i)^2$ is also returned.

Example:

```
$ARRAYS
$REAL
ARRIN(2,6) = 0.0, 10.0, 10.0, 15.0, 20.0, 27.0,
             30.0, 32.0, 45.0, 32.0, 60.0, 30.0;
ARROUT(4) = U;
...
$INITIAL
      DOUBLE PRECISION SQERR
      CALL LSTSQ(ARRIN, 3, SQERR, ARROUT)
```

Restrictions:

Array ARR2 must be a real array with size > N + 1.

6.11.2 Polynomial Equation

Name:

FUNCTION **POLYNM** (X, ARRNAM)

X	-	Double precision value to be substituted
ARRNAM	-	Array name

Type:

Double precision.

Purpose:

To calculate

$$f(x) = a_0 + a_1 x + \ldots + a_n x^n$$

where the coefficients $a_0, \ldots, a_n$ are stored in ARRNAM.

Example:

```
DOUBLE PRECISION VALUE
VALUE = POLYNM(5.7D0, ARRNAM)
```

Restrictions:

ARRNAM must be a real, one-dimensional array.

6.12 Scalar Array Functions

AAPRGL Set elements to arithmetic progression, with limit
AAPRG Set elements to arithmetic progression
ACONST Set all elements to a constant value
AELADD Add elements of two arrays
AELCPY Copy an array to another array
AELDIV Divide elements of two arrays
AELEQ Compare (=) elements of two arrays
AELGE Compare (≥) elements of two arrays
AELGT Compare (>) elements of two arrays
AELINV Copy reciprocal values of an array
AELLE Compare (≤) elements of two arrays
AELLT Compare (<) elements of two arrays
AELMLT Multiply elements of two arrays
AELNE Compare (≠) elements of two arrays
AELSUB Subtract elements of two arrays
AFUNCI Set elements of array to integer function value
AFUNCR Set elements of array to real function value
AFUNI2 As **AFUNCI**, but for two arrays
AFUNR2 As **AFUNCR**, but for two arrays
AGPRGL Set elements to geometric progression, with limit
AGPRG Set elements to geometric progression
AVADD Add elements of an array
AVDIV Divide elements of an array
AVEQ Compare (=) elements of an array
AVGE Compare (≥) elements of an array
AVGT Compare (>) elements of an array
AVLE Compare (≤) elements of an array
AVLT Compare (<) elements of an array
AVMLT Multiply elements of an array
AVNE Compare (≠) elements of an array
AVSUB Subtract elements of an array

All arrays in this section are nominally one-dimensional real or integer arrays, although predefined arrays of any order will be accepted. The names ANAME1, ANAME2 and ANAME3 are used to signify such arrays in the following descriptions. Arrays and values must be of like arithmetic type (Real/Integer) in each call.

6.12.1 Set Array to AP

Name:

SUBROUTINE **AAPRG** (ARRNAM, VAL, INC)
SUBROUTINE **AAPRGL** (ARRNAM, VAL, INC, LIM)

ARRNAM	-	Real/Integer array name
VAL	-	Double precision/Integer value
INC	-	Double precision/Integer increment
LIM	-	Double precision/Integer limit

Purpose:

To set array element values in arithmetic progression, starting with value VAL and increasing each element by INC. AAPRGL stops progression when successive element values encompass limit LIM; further element values remain unchanged.

Example:

```
      DOUBLE PRECISION RVALUE
      INTEGER IVALUE
      RVALUE=5.7
      IVALUE=4
C
      CALL AAPRG(RARR, RVALUE, 1.5D0)
      CALL AAPRGL(IARR, IVALUE, 2, 9)
```

Restrictions:

ARRNAM must be a one dimensional array. VAL, INC, LIM must all be of the same arithmetic type as array ARRNAM. LIM must be greater than zero.

6.12.2 Set Array to Constant

Name:

SUBROUTINE **ACONST** (ARRNAM, VAL)

ARRNAM	-	Real/Integer array name
VAL	-	Double precision/Integer value

Purpose:

To set all elements of an array to a constant value.

Example:

```
      INTEGER IVALUE
      IVALUE=4
C
      CALL ACONST(RARR, 1.5D0)
      CALL ACONST(IARR, IVALUE)
```

Restrictions:

VAL must be of the same arithmetic type as the array ARRNAM.

6.12.3 Vector Arithmetic

Name:

SUBROUTINE **AELADD** (ANAME1, ANAME2, ANAME3)
SUBROUTINE **AELSUB** (ANAME1, ANAME2, ANAME3)
SUBROUTINE **AELMLT** (ANAME1, ANAME2, ANAME3)
SUBROUTINE **AELDIV** (ANAME1, ANAME2, ANAME3)

ANAME1	-	Real/Integer array name
ANAME2	-	Real/Integer array name
ANAME3	-	Real/Integer array name

Purpose:

Add (AELADD), subtract (AELSUB), multiply (AELMLT), and divide (AELDIV) elements of ANAME1 with corresponding elements of ANAME2. Store resultant array in ANAME3.

Example:

```
CALL AELADD(RARR1, RARR2, RARR3)
CALL AELSUB(IARR1, IARR2, IARR3)
CALL AELMLT(RARR1, RARR2, RARR3)
CALL AELDIV(IARR1, IARR2, IARR3)
```

Restrictions:

ANAME1 and ANAME2 must be real/integer arrays of the same dimensions. The number of elements in array ANAME1 must equal the number of elements in array ANAME2. The declared size of array ANAME3 must be greater than or equal to the number of elements in ANAME1/ANAME2.

6.12.4 Array Copy

Name:

SUBROUTINE **AELCPY** (ANAME1, ANAME2)

ANAME1	-	Real/Integer array name
ANAME2	-	Real/Integer array name

Purpose:

Copy all values of elements of ANAME1 into ANAME2.

Example:

```
CALL AELCPY(IARR1, IARR2)
CALL AELCPY(RARR1, RARR2)
```

Restrictions:

The declared size of array ANAME2 must be greater than or equal to the number of elements in array ANAME1. Both arrays must be of the same type - either both real or both integer.

6.12.5 Vector Logical Operations

Name:

SUBROUTINE **AELGT** (ANAME1, ANAME2, ANAME3)
SUBROUTINE **AELLT** (ANAME1, ANAME2, ANAME3)
SUBROUTINE **AELEQ** (ANAME1, ANAME2, ANAME3)
SUBROUTINE **AELNE** (ANAME1, ANAME2, ANAME3)
SUBROUTINE **AELGE** (ANAME1, ANAME2, ANAME3)
SUBROUTINE **AELLE** (ANAME1, ANAME2, ANAME3)

ANAME1	-	Real/Integer array name
ANAME2	-	Real/Integer array name
ANAME3	-	Real/Integer array name

Purpose:

Compare elements of ANAME1 with the corresponding elements in ANAME2 using the logical operations greater than (AELGT), less than (AELLT), equal to (AELEQ), not equal to (AELNE), greater than or equal to (AELGE), and less than or equal to (AELLE). If the operation is true for an element, copy element value into corresponding position in ANAME3. If false, insert zero into ANAME3.

Example:

```
CALL AELGT(IARR1, IARR2, IARR3)
CALL AELLT(RARR1, RARR2, RARR3)
CALL AELEQ(IARR1, IARR2, IARR3)
CALL AELNE(RARR1, RARR2, RARR3)
CALL AELGE(IARR1, IARR2, IARR3)
CALL AELLE(RARR1, RARR2, RARR3)
```

Restrictions:

ANAME1 and ANAME2 must be real/integer arrays of the same dimensions.

6.12.6 Array Reciprocal Copy

Name:

SUBROUTINE **AELINV** (ANAME1, ANAME2)

ANAME1	-	Real/Integer array name
ANAME2	-	Real/Integer array name

Purpose:

Copy reciprocal of values of elements of ANAME1 into ANAME2.

Example:

```
CALL AELINV(ANAME1, ANAME2)
```

Restrictions:

Both arrays must be of type real. The declared size of array ANAME2 must be greater than or equal to the number of elements in array ANAME1.

6.12.7 Function Operations (1)

Name:

SUBROUTINE **AFUNCI** (IFUNC, ANAME1, ANAME2)
SUBROUTINE **AFUNCR** (DFUNC, ANAME1, ANAME2)

IFUNC	-	Integer function name
DFUNC	-	Double precision function name
ANAME1	-	Real/Integer array name
ANAME2	-	Real/Integer array name

Purpose:

Set ANAME2 elements equal to function of corresponding ANAME1 elements. Integer and double precision respectively. These subroutines are used for functions with one argument.

Example:

Using the function defined in the data blocks as:

```
DOUBLE PRECISION * FUNCTION * PLY(X) = 2 * X ** 3 + 1;
```

The following represents a valid input in the operations blocks.

```
EXTERNAL PLY
CALL AFUNCR(PLY , ARR1 , ARR2)
```

Restrictions:

The operations block calling these routines must contain the EXTERNAL statement specifying the function name. Arrays ANAME1 and ANAME2 must be of the same dimension and the declared size of ANAME2 must be greater than or equal to the number of elements in ANAME1.

6.12.8 Function Operations (2)

Name:

SUBROUTINE **AFUNI2** (IFUNC, ANAME1, ANAME2, ANAME3)
SUBROUTINE **AFUNR2** (DFUNC, ANAME1, ANAME2, ANAME3)

IFUNC	-	Integer function name
DFUNC	-	Double precision function name
ANAME1	-	Real/Integer array name
ANAME2	-	Real/Integer array name
ANAME3	-	Real/Integer array name

Purpose:

As AFUNCI/AFUNCR, but with two-argument functions. Argument values taken from ANAME1 and ANAME2, result stored in ANAME3. The subroutines are used for functions with two arguments.

Example:

Using the function defined in the data blocks as:

```
DOUBLE PRECISION * FUNCTION * PLY(X,Y) = 2*X**3 + Y**2;
```

The following represents a valid input in the operations blocks.

```
 EXTERNAL PLY
CALL AFUNCR(PLY , ARR1 , ARR2 , ARR3)
```

Restrictions:

The operations block calling these routines must contain the EXTERNAL statement specifying the function name. Arrays ANAME1 and ANAME2 must be of the same dimension and contain the same number of elements. The declared size of ANAME3 must be greater than or equal to the number of elements in ANAME1/ANAME2.

6.12.9 Set Array to GP

Name:

SUBROUTINE **AGPRG** (ARRNAM, VAL, FAC)
SUBROUTINE **AGPRGL** (ARRNAM, VAL, FAC, LIM)

ARRNAM	-	Real/Integer array name
VAL	-	Double precision/Integer value
INC	-	Double precision/Integer increment
LIM	-	Double precision/Integer limit

Purpose:

To set array element values in geometric progression, starting at value VAL increasing by FAC. AGPRGL stops progression when successive element values encompass limit LIM. Further element values remain unchanged.

Example:

```
      DOUBLE PRECISION RVALUE
      INTEGER IVALUE
      RVALUE=5.7
      IVALUE=4
C
      CALL AGPRG(RARR, RVALUE, 1.5D0)
      CALL AGPRGL(IARR, IVALUE, 2, 9)
```

Restrictions:

ARRNAM must be a one dimensional array. VAL, FAC, LIM must all be of the same arithmetic type as array ARRNAM.

6.12.10Scalar Arithmetic

Name:

SUBROUTINE **AVADD** (ANAME1, VAL, ANAME2)
SUBROUTINE **AVSUB** (ANAME1, VAL, ANAME2)
SUBROUTINE **AVMLT** (ANAME1, VAL, ANAME2)
SUBROUTINE **AVDIV** (ANAME1, VAL, ANAME2)

VAL	-	Double precision/Integer value.
ANAME1	-	Real/Integer array name
ANAME2	-	Real/Integer array name

Purpose:

Add (AVADD), subtract (AVSUB), multiply (AVMLT), and divide (AVDIV) elements of ANAME1 with VAL respectively and store resultant array in ANAME2.

Example:

```
      DOUBLE PRECISION RVALUE
      INTEGER IVALUE
      RVALUE=5.7
      IVALUE=4
C
      CALL AVADD(RARR1, RVALUE, RARR2)
      CALL AVSUB(IARR1, IVALUE, IARR2)
      CALL AVMLT(RARR1, 4.5D0, RARR2)
      CALL AVDIV(IARR1, 7, IARR2)
```

Restrictions:

ANAME1 and ANAME2 must be either one or two dimensional Real/Integer arrays. The declared size of array ANAME2 must be greater than or equal to the number of elements in array ANAME1.

6.12.11 Scalar Logical Operations

Name:

SUBROUTINE **AVGT** (ANAME1, VAL, ANAME2)
SUBROUTINE **AVLT** (ANAME1, VAL, ANAME2)
SUBROUTINE **AVEQ** (ANAME1, VAL, ANAME2)
SUBROUTINE **AVNE** (ANAME1, VAL, ANAME2)
SUBROUTINE **AVGE** (ANAME1, VAL, ANAME2)
SUBROUTINE **AVLE** (ANAME1, VAL, ANAME2)

VAL	-	Double precision/Integer value
ANAME1	-	Real/Integer array name
ANAME2	-	Real/Integer array name

Purpose:

Compare elements of ANAME1 with VAL using the logical operations greater than (AVGT), less than (AVLT), equal to (AVEQ), not equal to (AVNE), greater than or equal to (AVGE), and less than or equal to (AVLE). If operation is true for any element, copy element value into corresponding position in ANAME2. If false, insert zero into ANAME2.

Example:

```
      DOUBLE PRECISION RVALUE
      INTEGER IVALUE
      RVALUE=5.7
      IVALUE=4
C
      CALL AVGT(RARR1, RVALUE, RARR2)
      CALL AVLT(IARR1, IVALUE, IARR2)
      CALL AVEQ(RARR1, 4.5D0, RARR2)
      CALL AVNE(IARR1, 7, IARR2)
      CALL AVGE(RARR1, 5.3D0, RARR2)
      CALL AVLE(IARR1, IVALUE, IARR2)
```

Restrictions:

ANAME1 and ANAME2 must be real/integer arrays of the same dimensions.

6.13 Scaling Functions

SCALE1 Scale first independent variable in table array
SCALE2 Scale second independent variable in table array
SCALE3 Scale third independent variable in table array
SCALEV Scale a dependent variable

All arrays in this section are either real or table arrays. ARRNAM is used to denote an array name, and X is a real scaling value.

Note: the scaling routines change the values in the array.

6.13.1 Scale Independent Variable

Name:

SUBROUTINE **SCALE1** (ARRNAM, X)
SUBROUTINE **SCALE2** (ARRNAM, X)
SUBROUTINE **SCALE3** (ARRNAM, X)

Purpose:

Scale first, second, or third independent variable of a table array by X respectively. SCALE1 also permits input from a (2, N) real array, in which case the independent variable is assumed to be stored in the first row.

Example:

```
      DOUBLE PRECISION RVALUE
      RVALUE=5.7
C
      CALL SCALE1(RARR1, RVALUE)
      CALL SACLE2(RARR2, 3.2D0)
      CALL SCALE3(RARR3, 4.5D0)
```

6.13.2 Scale Dependent Variable

Name:

SUBROUTINE **SCALEV** (ARRNAM, X)

Purpose:

Scale dependent variable by X. ARRNAM may be either a table array or a (2, N) real array, in the latter case the dependent variable is assumed to be stored in the second row.

Example:

```
CALL SCALEV(RARR1, 4.5D0)
```

6.14 Solution Routines

SLCRNC Crank-Nicolson forward-backward transient solver
SLFRTF Frequency response transfer function solver
SLFRWD Forward differencing transient solver
SLFWBK Crank-Nicolson forward-backward transient solver
SLGEAR Gear-formulation transient solver (matrix)
SLGRDJ Gear-formulation transient solver (iteration)
SLMODE All modes modal analysis solution routine
SOLVFM Full matrix steady state solver
SOLVIT Iterative steady state solver
SOLVCG Conjugate gradient steady state solver

FGENFI General thermohydraulic implicit transient routine
FGENSS General thermohydraulic steady-state routine
FLTMTS Single-phase thermohydraulic implicit multi-time stepping routine
FLTNSS Single-phase thermohydraulic steady-state solver
FLTNTF Single-phase thermohydraulic explicit transient solver
FLTNTS Single-phase thermohydraulic implicit transient solver

SOLCYC Cyclic meta-solver

6.14.1 Introduction

Solution Program Control Sequence

The solution program is generated from the user-given operations blocks. In general, each of the blocks $INITIAL, $VARIABLES1, and $VARIABLES2 generates one Fortran subroutine per submodel, while every subroutine or function defined in a $SUBROUTINES block generates a corresponding Fortran subroutine or function. The $EXECUTION and $OUTPUTS blocks, valid only in the top-level model, each generate a Fortran subroutine.

A main program is generated by the preprocessor, its function being to initiate and control the solution sequence, which is as follows.

The first event is the initialisation of various items such as logical unit numbers and list lengths. This is followed by loading of the solution common blocks from the binary model data base file, the single-precision data being converted to double precision for better accuracy of the solution. The procedure is then for a main run to be executed followed by any parameter cases which may have been defined.

A test is made for the existence of the parameter case file, *model*.PAR. If this is not present then the solution is instigated by calling first the subroutine resulting from the main-model $INITIAL block, then that from the main-model $EXECUTION block. On exit from the latter, the run terminates.

If the parameter case file *is* present, then the $PARAMETERS line in the file is scanned for the allowed options. If PARMONLY has not been specified then the nominal case is executed just as described above. Each parameter case is now processed.

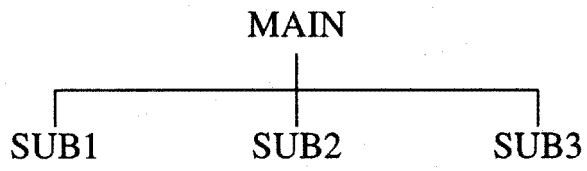
For a final parameter case, the given commands are carried out on the common blocks in their current state. If an initial case is requested then the common blocks are reloaded from the model data base file and the main-model $INITIAL subroutine is called prior to the commands being acted on. The main-model $EXECUTION subroutine is then invoked.

The subroutines generated from the operations blocks $VARIABLES1, $VARIABLES2, and $OUTPUTS are not called directly in the process described above. Instead, they are called directly from the library solution routines, which are in turn called (by the user) from $EXECUTION. The calling sequence for each solution routine is described for each routine in the User Manual, but in general the process is:-

Steady state	-	$VARIABLES1 each iteration $VARIABLES2 and $OUTPUTS at solution completion
Transient	-	$VARIABLES1 each iteration $VARIABLES2 end of each timestep $OUTPUTS at the start, the end, and each requested output interval in between

The Effect of Submodel Hierarchy

As described above, subroutines for $INITIAL, $VARIABLES1, and $VARIABLES2 are generated for each submodel. Consider the submodel structure:-



with the corresponding input deck:-

```
$MODEL MAIN
   $MODEL SUB1
   $ENDMODEL SUB1
   $MODEL SUB2
   $ENDMODEL SUB2
   $MODEL SUB3
   $ENDMODEL SUB3
$ENDMODEL MAIN
```

The generated Fortran subroutines for each model are:-

MAIN	-	IN0001, V10001, V20001
SUB1	-	IN0002, V10002, V20002
SUB2	-	IN0003, V10003, V20003
SUB3	-	IN0004, V10004, V20004

In addition, MAIN generates subroutines EXECTN and OUTPUT for $EXECUTION and $OUTPUTS respectively.

The subroutines for the main model commence with CALL statements to the lower level model subroutines, e.g.:-

```
      SUBROUTINE IN0001

(COMMON BLOCKS)

      CALL IN0002
      CALL IN0003
      CALL IN0004

(USER-SUPPLIED STATEMENTS)
```

```
RETURN
END
```

Therefore, to execute the logic of the $INITIAL, $VARIABLES1, or $VARIABLES2 blocks, the library routines call only the top-level subroutines IN0001, V10001, and V20001.

FHTS Solution Routines

Several solution routines are provided for the thermohydraulic solution of piped, one-dimensional fluid networks. These may be closed networks (loops) or open networks (lines), and a model may contain several such loops and/or lines. The solution routines fall into two very distinct categories. The first set perform steady-state and transient solutions for single-phase fluids only, both vapours and liquids. These do not take into account compressible effects other than that the density is related to the pressure and temperature of the system by an equation of state. They are therefore suitable for pumped liquid loops and low-pressure fan-assisted systems such as air-conditioning ducting, where density changes can be neglected.

The second set of solution routines are for both single- and two-phase fluids, and are termed the 'general solution' routines. Two-phase fluids are treated by the homogeneous, no-slip approximation which assumes that the fluid can be treated as a single, perfectly mixed mass, with mean properties evaluated in proportion to the vapour quality. Both liquid and vapour phases are assumed to move at equal velocity. Compressibility effects *are* modelled in these routines, in that density variations with time and distance are significant, but only at subsonic speeds. An implicit transient and a pseudo-transient solution routine are provided along with a true steady-state finder.

For transient solutions the user may select a timestep in the implicit routines, or rely upon an internally calculated one in the explicit solution. Reference is made there to the Courant number, defined as:-

Co = (Velocity x Timestep) / (Nodal length)

This gives a stability criterion for the solution timestep. Since for a Courant number of unity the timestep is exactly that for a fluid particle to travel the nodal length, timesteps should be less than this in order to avoid numerical instabilities. The Courant limiting timestep (i.e. the timestep required for Co = 1) is contained in control constant DTCOUR. It must also be pointed out that, when the Courant limiting time step is automatically used, a damping factor is applied. Thus, the time step is set to be RLCOUR * DTCOUR where RLCOUR is a control constant which defaults to 0.5; this can be overwritten by the user.

Three other control constants of interest are PABS, MINDP and MINFLO. PABS is a user-given reference pressure; output from FHTS will give absolute pressures. By default PABS is set to zero.

MINDP signifies the minimum pressure difference across a mass-flow link that the solution will recognise. If a ΔP is less than MINDP in magnitude, then the upstream and

downstream nodes will be considered to be at the same pressure; the flow rate in the link must then be determined by applying mass conservation at the two nodes. The default value for MINDP is 1.0×10^{-3} Pa; the user should beware that mathematical inconsistencies may arise if this value is increased too much.

MINFLO is the minimum absolute flow rate for a mass-flow link to be included in the convergence check, and defaults to 1.0×10^{-8} kg/s. Very small flow rates tend to be affected by numerical 'noise', and can therefore prevent solution convergence.

Two-phase Models

The general solution routines are capable of predicting single-phase conditions, but with higher computational effort. It is therefore advised that the user employs single-phase solution routines whenever purely single-phase events are being modelled. Furthermore, the prediction of two-phase conditions is inherently more complex and prone to problems with stability and convergence. The following basic guidelines are therefore presented for optimising convergence of the general solvers and ensuring accuracy of the results.

FGENSS (steady state):-

1) Set convergence criteria RELXCA = FRLXCA = RELXMA = 0.001

2) Set outer iteration limit NLOOP =100

3) Run model with default damping factors (DAMPM, DAMPT) and inner iteration limits (NLOOPH, NLOOPT) – i.e. leave unspecified

4) If convergence not attained but RELXCC, FRLXCC and RELXMC are close to the required values then increase NLOOP and rerun; otherwise, do the following:

    i. Set NLOOPH = NLOOPT = 10 (5, 50, 100, …)

    ii. Set DAMPT = 0.5 (0.2, 0.1, …)

    iii. Set DAMPM = 0.2 (0.1, 0.05, …)

FGENFI (transient):-

1) Choose appropriate timestep: set DTIMEI = 1.0, say

2) Set convergence criteria RELXCA = FRLXCA = RELXMA = 0.001

3) Set outer iteration limit NLOOP = 100

4) Run model with default thermal damping factor (DAMPT) [damping factors for enthalpy and flow rates are calculated automatically]

5) If convergence difficult to attain (e.g. timestep DTIMEU automatically reduced too often) then do the following:

   i. Set NLOOP = 200 (500, …)

   ii. Set DAMPT = 0.5 (0.2, 0.1, …)

   iii. Reduce timestep: DTIMEI = 0.5 (0.1, …)

Units

The ability to work in arbitrary units is catered for by the control constants TABS and STEFAN. The latter is the Stefan-Boltzmann constant, and TABS is the displacement of the zero temperature of the thermal solution from that of absolute zero. For fluid solutions there are four extra control constants introduced, which are PABS, GRAVX, GRAVY and GRAVZ. PABS, as previously described, is the reference pressure and GRAVX, GRAVY and GRAVZ are the acceleration due to gravity in the x-, y- and z-coordinate directions.

For example, the default values

TABS = 273.15
STEFAN = 5.6686E-8
PABS = 0.0
GRAVX = 0.0
GRAVY = 0.0
GRAVZ = 0.0

are consistent with a model specified in watts, metres and degrees Celsius. Changing **TABS** to 0.0, for example, would result in calculations in Kelvin.

SOLVER and MODULE

Two control constants which reflect the chosen solution routine are available for use in operations block logic. SOLVER is set by each solution routine to indicate the type of analysis being performed, and takes the following values:

’SSTH ’	-	Steady state thermal
’SSFL1P’	-	Steady state fluid, single-phase
’SSFL2P’	-	Steady state fluid, two-phase
’TRTH ’	-	Transient thermal
’TRFL1P’	-	Transient fluid, single-phase
’TRFL2P’	-	Transient fluid, two-phase
’CYTH ’	-	Cyclic thermal
’CYFL1P’	-	Cyclic fluid, single-phase
’CYFL2P’	-	Cyclic fluid, two-phase

SOLVER is reset to ’ ’ on exit from the solution routine.

MODULE is set to the name of the solution routine, e.g. ’SOLVIT’, or ’FGENFI’; this value is retained on exit from the routine. In the case of the cyclic solver, SOLCYC, MODULE gives the name of the underlying (transient) solution routine.

6.14.2 SLFRWD

Name:

SUBROUTINE **SLFRWD**

Purpose:

This subroutine performs transient thermal analysis by the explicit forward differencing method. The stability criterion of each diffusion node is calculated and the minimum value if placed in control constant CSGMIN. The time step used (control constant DTIMEU) is set as 0.95*CSGMIN. The subroutine predicts the temperatures of diffusion nodes with non-zero capacitances by the forward differencing method before a successive point iterative solution of the zero capacitance diffusion nodes is performed.

The user is required to specify the problem start time (control constant TIMEO) if this is not 0.0, the end time (TIMEND) and output interval (OUTINT). The user may optionally specify a maximum iteration count (NLOOP) and relaxation criterion (RELXCA) for solution of the zero capacitance nodes. If NLOOP is omitted then a single solution is performed. If convergence to within RELXCA is not achieved inside NLOOP iterations for a time step then the solution continues but an error code is set.

The control constants DTMIN, DTMAX and DTPMAX may be set in order to regulate the time step used by the program. DTMIN specifies the minimum time step length allowed, DTMAX the maximum time step length allowed and DTPMAX the maximum temperature change allowed over a time step (non-arithmetic diffusion nodes only). Where these criteria clash, e.g. a time step smaller than DTMIN gives a temperature change greater than DTPMAX or DTMAX set less than DTMIN, DTMIN prevails and the solution continues. If DTPMAX is exceeded because of this an error code is set.

The user can specify the type of units used by setting the control constants STEFAN and TABS.

$VARIABLES1 is executed at the start of each time step and $VARIABLES2 at the end. $OUTPUTS is executed at the beginning of the transient, every OUTINT units of time during the transient and on completion of the transient.

Control Constants:

Name	Status	Default	Description
CSGMIN	Output		Minimum ratio of capacitance to sum of conductances (excluding arithmetic nodes and temperature boundaries)
DTIMEI	Output		Initial time step

Name	Status	Default	Description
DTIMEU	Output		Time step used
DTMAX	Optional	1.0E10	Maximum time step
DTMIN	Optional	0.0	Minimum time step
DTPMAX	Optional	1.0E10	Maximum allowable temperature change over a time step (non-arithmetic diffusion nodes only)
LOOPCT	Output		Number of solution iterations
MODULE	Output	'SLFWRD'	Current solution module
NCSGMN	Output		Node of CSGMIN
NLOOP	Optional	1	Maximum allowable number of outer iterations
NRLXCC	Output		Node of RELXCC
OUTINT	Optional	TIMEND - TIMEO	Output interval
RELXCA	Optional	1.0E10	Temperature convergence criterion
RELXCC	Output		Calculated temperature convergence
SOLVER	Output	'TRTH'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
TABS	Optional	273.15	Temperature reference
TIMEM	Output		Mean time over step
TIMEN	Output		Time at end of step
TIMEND	Mandatory		Time at end of solution
TIMEO	Optional	0.0	Time at start of solution

Example:

```
$EXECUTION
      RELXCA=0.001
      NLOOP=100
      TIMEND=100.0
      OUTINT=10.0
      CALL SLFRWD
```

Restrictions:

This routine can only be used with pure thermal models. Control constant TIMEND must be specified. It requires (4 * NNT + 2) locations of dynamic core (NNT = total number of nodes in system). SLFRWD may be called from the $EXECUTION block only.

6.14.3 SLCRNC

Name:

SUBROUTINE **SLCRNC**

Purpose:

This routine performs transient thermal analysis by implicit forward-backward differencing—the Crank-Nicolson method.

The user is required to specify the following control constants: the time step length (DTIMEI); the problem start time (TIMEO) if this is not 0.0; the problem end time (TIMEND); the output interval (OUTINT); the maximum iteration count (NLOOP) for each time step; the relaxation criterion (RELXCA) for convergence on a time step.

The time step length is taken as the user supplied initial time step length, DTIMEI, unless one or both of the control constants DTPMAX and DTROCA have been set.

DTPMAX specifies the maximum temperature change allowable over a time step (non-arithmetic diffusion nodes only).

SLCRNC performs the transient by calculating the rate of change of temperature of each node at the beginning and end of the time step, and using the average to predict the end of time step temperatures (the change in temperature of a node being the average rate of change multiplied by the time step length). The variation in the rate of change of temperature over time is an important indicator of acceptable time step length, and hence DTROCA specifies the limit on the maximum change in the rate of change of temperature over a time step multiplied by the time step length. In most circumstances a value of between 1.0 and 10.0 for DTROCA is appropriate.

If DTPMAX and/or DTROCA have been specified, the program attempts to maximise the time step subject to the constraints implied by these constants. If DTPMAX is exceeded then the time step is reduced and the temperatures recalculated. If DTPMAX or DTROCA have been set to regulate the time step length then it is advisable that DTMAX and DTMIN be set. DTMAX and DTMIN specify the maximum and minimum time step lengths allowable. Where DTPMAX or DTROCA would lead to a smaller time step, DTMIN is used and the solution continues but an error code is set.

For a given node its capacitance divided by the sum of conductances to the node (CSG) is a measure of the rate at which the node temperature responds to a given heat impulse. Nodes with a zero capacitance are referred to as "arithmetic" and are assumed to respond immediately to any impulse. This is modelled by calculating the steady state temperature for these nodes at each time step. The control constant ARITH enables the user to specify a minimum value for CSG below which a node will be considered as arithmetic by SLCRNC. The value given to ARITH is the

fraction of the current time step length. That is, setting ARITH to 0.01 implies that any node whose CSG is less or equals to 1% of the current time step length will be considered as an arithmetic node for this time step.

SLCRNC is a forward/backward differencing routine. For the forward step $VARIABLES1 is executed for the time step start temperatures with TIMEM set to the time step start time. For the subsequent backward step $VARIABLES1 is executed for the time step end temperatures with TIMEM set to the time step end time. By default $VARIABLES1 is called only for the first two iterations of the backward step. This behaviour can be modified by setting the control value METHOD=2 which then forces $VARIABLES1 to be invoked at every iteration. This will usually increase the stability and accuracy of the solution but may also increase calculation time. $VARIABLES2 is executed at the end of each time step, and $OUTPUTS is executed at the beginning of the transient, every OUTINT units of time during the transient and on completion of the transient.

The change in temperature at each node may be damped via the control constant DAMPT. However, this is not recommended: If a model fails to converge, then consider first reducing the time step length.

The user can specify the type of units used by setting the control constants STEFAN and TABS.

Control Constants:

	Status	Default	Description
ARITH	Optional	0.0	Fraction of DTIMEU for arithmetic node cut-off
CSGMIN	Output		Minimum ratio of capacitance to sum of conductances (excluding arithmetic nodes and temperature boundaries)
DAMPT	Optional	1.0	Temperature damping
DTIMEI	Mandatory		Input time step
DTIMEU	Output		Time step used
DTMAX	Optional	1.0E10	Maximum time step
DTMIN	Optional	0.0	Minimum time step
DTPMAX	Optional	1.0E10	Maximum allowable temperature change over a time step (non-arithmetic diffusion nodes only)
DTROCA	Optional	1.0E10	Maximum allowable change in temperature rate of change over time step
DTROCC	Output		Calculated change in temperature rate of change over time step
LOOPCT	Output		Number of solution iterations
METHOD	Optional	0	Selective convergence scheme flag
MODULE	Output	'SLCRNC'	Current solution module
NCSGMN	Output		Node of CSGMIN
NDTROC	Output		Node of DTROCC

	Status	Default	Description
NLOOP	Mandatory		Maximum allowable number of outer iterations
NRLXCC	Output		Node of RELXCC
OUTINT	Optional	TIMEND - TIMEO	Output interval
RELXCA	Mandatory		Temperature convergence criterion
RELXCC	Output		Calculated temperature convergence
SOLVER	Output	'TRTH'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
TABS	Optional	273.15	Temperature reference
TIMEM	Output		Integration time constant
TIMEN	Output		Time at end of step
TIMEND	Mandatory		Time at end of solution
TIMEO	Optional	0.0	Time at start of step

Example:

```
$EXECUTION
      RELXCA=0.001
      NLOOP=100
      DTIMEI=1.0
      DTROCA=1.0
      DTMAX=10.0
      TIMEND=100.0
      OUTINT=10.0
      CALL SLCRNC
```

Restrictions:

This routine can only be used with pure thermal models. Control constants DTIMEI, TIMEND, NLOOP and RELXCA must be specified. It requires 9*NNT locations of dynamic core (NNT = total number of thermal nodes in system). SLCRNC may be called from the $EXECUTION block only.

SLCRNC has been developed from SLFWBK. In addition to the availability of DTROCA and ARITH as time step control mechanisms, the following important difference exists between the two routines: SLFWBK calls $VARIABLES1 once per time step, at the beginning of the time step, using the time step start temperatures and with TIMEM set to the start, mid and end time step times respectively. SLCRNC calls $VARIABLES1 before both the forward and backward differencing stages of the time step. Before the forward differencing stage using the time step start temperatures and with TIMEM set to the time step start time, and before the backward differencing stage using the calculated time step end temperatures, and with TIMEM set to the time step end time.

6.14.4 SLFWBK

Name:

SUBROUTINE **SLFWBK**

Purpose:

This routine performs transient thermal analysis by implicit forward-backward differencing—the Crank-Nicolson method.

The user is required to specify the following control constants: the time step length (DTIMEI); the problem start time (TIMEO) if this is not 0.0; the problem end time (TIMEND); the output interval (OUTINT); the maximum iteration count (NLOOP) for each time step; the relaxation criterion (RELXCA) for convergence on a time step.

The time step length is taken as the user supplied initial time step length, DTIMEI, unless the control constant DTPMAX has been set. DTPMAX specifies the maximum temperature change allowable over a time step (non- arithmetic diffusion nodes only).

The program increases the time step to bring the maximum temperature change in line with the user supplied maximum. If DTPMAX is exceeded then the time step is reduced and the temperatures recalculated. If DTPMAX has been set to regulate the time step length then it is advisable that DTMAX and DTMIN are set. DTMAX and DTMIN specify the maximum and minimum time step lengths allowable. Where DTPMAX would lead to a smaller time step, DTMIN is used and the solution continues but an error code is set.

The change in temperature at each node may be damped via the control constant DAMPT.

The user can specify the type of units used by setting the control constants STEFAN and TABS.

$VARIABLES1 is executed at the start of each time step and $VARIABLES2 at the end. $OUTPUTS is executed at the beginning of the transient, every OUTINT units of time during the transient and on completion of the transient.

Control Constants:

	Status	Default	Description
CSGMIN	Output		Minimum ratio of capacitance to sum of conductances (excluding arithmetic nodes and temperature boundaries)
DAMPT	Optional	1.0	Temperature damping
DTIMEI	Mandatory		Input time step

	Status	Default	Description
DTIMEU	Output		Time step used
DTMAX	Optional	1.0E10	Maximum time step
DTMIN	Optional	0.0	Minimum time step
DTPMAX	Optional	1.0E10	Maximum allowable temperature change over a time step (non-arithmetic diffusion nodes only)
LOOPCT	Output		Number of solution iterations
MODULE	Output	'SLFWBK'	Current solution module
NCSGMN	Output		Node of CSGMIN
NLOOP	Mandatory		Maximum allowable number of outer iterations
NRLXCC	Output		Node of RELXCC
OUTINT	Optional	TIMEND - TIMEO	Output interval
RELXCA	Mandatory		Temperature convergence criterion
RELXCC	Output		Calculated temperature convergence
SOLVER	Output	'TRTH'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
TABS	Optional	273.15	Temperature reference
TIMEM	Output		Mean time over step
TIMEN	Output		Time at end of step
TIMEND	Mandatory		Time at end of solution
TIMEO	Optional	0.0	Time at start of solution

Example:

```
$EXECUTION
      RELXCA=0.001
      NLOOP=100
      DTIMEI=1.0
      TIMEND=100.0
      OUTINT=10.0
      CALL SLFWBK
```

Restrictions:

This routine can only be used with pure thermal models. Control constants DTIMEI, TIMEND, NLOOP and RELXCA must be specified. It requires 3*NNT locations of dynamic core (NNT = total number of nodes in system). SLFWBK may be called from the $EXECUTION block only.

6.14.5 SLGEAR/SLGRDJ

Name:

SUBROUTINE **SLGEAR**
SUBROUTINE **SLGRDJ**

Purpose:

These routines perform transient thermal analysis by backwards differencing using the Gear formulation. The time step length is dynamically optimised by the solver after using an initial value of CSGMIN.

The user is required to specify the following control constants; the problem start time (TIMEO) if this is not equal to 0.0; the problem end time (TIMEND); the output interval (OUTINT); the maximum iteration count (NLOOP) for each time step and the relaxation criterion (RELXCA) used to check convergence and to choose time step length and solution order. The control constants DTMIN and DTMAX may also be set to specify the minimum and maximum time steps allowed.

The user can specify the type of units used by setting the control constants STEFAN and TABS.

The Gear formulation consists of a prediction followed by an iterated corrector stage for each time step. This corrector stage involves solving a matrix equation to yield a vector of temperature increments to be added to the prediction. SLGRDJ differs from SLGEAR in that the off-diagonal terms of the matrix in this equation are ignored. In effect, then, SLGEAR solves the temperature increment equation by matrix inversion, and SLGRDJ by successive point iteration.

$VARIABLES1 is executed at the start of each iteration within a time step and $VARIABLES2 on completion of each time step. $OUTPUTS is called at the start of the transient, every OUTINT units of time during the transient and at the end of the transient.

Control Constants:

	Status	Default	Description
CSGMIN	Output		Minimum ratio of capacitance to sum of conductances (excluding arithmetic nodes and temperature boundaries)
DTIMEU	Output		Time step used
DTMAX	Optional	1.0E10	Maximum time step
DTMIN	Optional	0.0	Minimum time step
LOOPCT	Output		Number of solution iterations

	Status	Default	Description
MODULE	Output	'SLGEAR' or 'SLGRDJ'	Current solution module
NCSGMN	Output		Node of CSGMIN
NLOOP	Mandatory		Maximum allowable number of outer iterations
NRLXCC	Output		Node of RELXCC
OUTINT	Optional	TIMEND - TIMEO	Output interval
RELXCA	Mandatory		Temperature convergence criterion
RELXCC	Output		Calculated temperature convergence
SOLVER	Output	'TRTH'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
TABS	Optional	273.15	Temperature reference
TIMEM	Output		Mean time over step
TIMEN	Output		Time at end of step
TIMEND	Mandatory		Time at end of solution
TIMEO	Optional	0.0	Time at start of solution

Example:

```
$EXECUTION
      RELXCA=0.001
      NLOOP=100
      TIMEND=100.0
      OUTINT=10.0
      CALL SLGEAR
```

Restrictions:

This routine can only be used with pure thermal models. Control constants TIMEND, NLOOP and RELXCA must be specified. SLGEAR requires (12*NNT + NNT**2) locations of dynamic core and SLGRDJ 13*NNT locations (NNT = total number of nodes in system). These routines may be called from the $EXECUTION block only.

6.14.6 SLMODE

Name:

SUBROUTINE **SLMODE**

Purpose:

This routine performs transient analysis using the modal analysis method.

Givens' method followed by the QR algorithm is used to evaluate the Eigen values and the Eigen vectors. The user is required to specify the following control constants: the time step length (DTIMEI); the problem start time (TIMEO) if this is not 0.0; the problem time end (TIMEND) and the output interval (OUTINT). The user must specify a maximum iteration count (NLOOP) and relaxation criterion (RELXCA) for solution of zero capacitance nodes.

If convergence to within RELXCA is not achieved inside NLOOP iterations for a time step then the solution continues but an error code is set. The control constants DTMIN, DTMAX and DTPMAX may be set in order to regulate the time step used by the program. DTMIN specifies the minimum time step length allowed, DTMAX the maximum time step length allowed and DTPMAX the maximum temperature change allowed over a time step (non- arithmetic diffusion nodes only). Where these criteria clash, e.g. a time step smaller than DTMIN gives a temperature change greater than DTPMAX or DTMAX set less than DTMIN, DTMIN prevails and the solution continues. If DTPMAX is exceeded because of this an error code is set.

The user can specify the type of units used by setting the control constants STEFAN and TABS.

Control Constants:

	Status	Default	Description
CSGMIN	Output		Minimum ratio of capacitance to sum of conductances (excluding arithmetic nodes and temperature boundaries)
DTIMEI	Mandatory		Input time step
DTIMEU	Output		Time step used
DTMAX	Optional	1.0E10	Maximum time step
DTMIN	Optional	0.0	Minimum time step
DTPMAX	Optional	1.0E10	Maximum allowable temperature change over a time step (non-arithmetic diffusion nodes only)
LOOPCT	Output		Number of solution iterations
MODULE	Output	'SLMODE'	Current solution module
NCSGMN	Output		Node of CSGMIN
NLOOP	Mandatory		Maximum allowable number of outer iterations

	Status	Default	Description
NRLXCC	Output		Node of RELXCC
OUTINT	Optional	TIMEND - TIMEO	Output interval
RELXCA	Mandatory		Temperature convergence criterion
RELXCC	Output		Calculated temperature convergence
SOLVER	Output	'TRTH'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
TABS	Optional	273.15	Temperature reference
TIMEM	Output		Mean time over step
TIMEN	Output		Time at end of step
TIMEND	Mandatory		Time at end of solution
TIMEO	Optional	0.0	Time at start of solution

Example:

```
$EXECUTION
      RELXCA=0.001
      NLOOP=100
      DTIMEI=1.0
      TIMEND=100.0
      OUTINT=10.0
      CALL SLMODE
```

Restrictions:

This solution routine can only be used with purely thermal models. Boundary nodes and arithmetic nodes (zero capacitance) are removed from the Eigen solution at the start of the transient. This means that the status of a node cannot be changed during the solution. If the user wishes to change the status of node during the transient then successive calls should be made to SLMODE, changing the node status between the two calls.

It requires (3*NNT**2 + 3*NNT + 1) locations of dynamic core (NNT = total number of nodes in system). All active diffusion nodes must have at least one active, non zero, link connecting them. The model must contain at least two non-arithmetic diffusion nodes. SLMODE may be called from the $EXECUTION block.

Numerical Background:

The heat conduction equation

$$[C]\dot{\underline{T}} + [K]\underline{T} = \underline{F}$$

is transformed into the new coordinates $\underline{S}$ such that

$$\dot{\underline{S}} + [\Lambda]\underline{S} = [W]^T \underline{F}$$

where

$$\underline{S} = [W]^T \underline{T}$$

[C] is the diagonal matrix of capacitance values

[K] is the conductivity matrix.

[W] and Λ are the matrix of Eigen vectors and Eigen values of the equation:-

$$(\lambda[C] - [K])\underline{x} = 0$$

To account for variation of capacitance and conductivity during the transient the matrices [C] and [K] and split into two parts.

$$[C] = [C]_L + [C]_N$$

$$[K] = [K]_L + [K]_N$$

The first term on the RHS represents the values at the time of the Eigen solution whilst the second represents the difference at the time $\underline{T}$ during the transient.

The equation now becomes

$$\dot{\underline{S}} + [\Lambda]\underline{S} = [W]^T(\underline{F} - [C]\dot{\underline{T}} - [K]\underline{T})$$

The LHS of this equation can be evaluated analytically whilst the RHS side term is evaluated at the start of the time step and is assumed to remain constant over the time step. The Eigen solution is performed at the start of the transient only.

6.14.7 SOLVIT

Name:

SUBROUTINE **SOLVIT**

Purpose:

This subroutine calculates the model steady state solution by successive point iteration. The control constant DAMPT is addressed by this routine, with the following effect. If DAMPT is set to 1.0 (or allowed to default to this), then a relaxation or damping factor is calculated dynamically as the solution proceeds until an optimised value is attained. The remainder of the solution proceeds with this value. DAMPT may also be set to any real value between 0.0 and 2.0. If this is not 1.0 then the solution proceeds with the specified value of DAMPT as the relaxation factor.

The user is required to specify the maximum number of iterations (NLOOP) to be performed in attempting to reach the steady state solution relaxation criterion (RELXCA) which determines when solution has been obtained. During the solution control constants LOOPCT and RELXCC contain the number of iterations undertaken so far and the maximum temperature change from the latest iteration respectively. In addition to NLOOP and RELXCC the user may also specify one or both of INBALA - the maximum allowed energy imbalance, and INBALR - the maximum allowed relative energy imbalance. The solution converges according to RELXCA, and then the energy balance is checked against INBALA and/or INBALR. If either of these are not satisfied, then further iterations are performed and the check repeated. This continues until INBALA and INBALR are satisfied, or NLOOP is exceeded. The absolute energy imbalance is defined to be the difference between the energy into the system (by heat sources and boundary nodes) and the energy out of the system (by boundary nodes). The relative energy imbalance is the absolute imbalance as a fraction of the maximum of the energy into the system or the energy out of the system. The user can specify the type of units used by setting the control constants STEFAN and TABS.

$VARIABLES1 is executed at the beginning of each iteration, $VARIABLES2 and $OUTPUTS on completion of the solution procedure.

Control Constants:

	Status	Default	Description
DAMPT	Optional	1.0	Temperature damping
ENBALA	Output		Absolute energy balance
ENBALR	Output		Relative energy balance
INBALA	Optional	1.0E10	Required absolute energy balance
INBALR	Optional	1.0	Required relative energy balance

	Status	Default	Description
LOOPCT	Output		Number of solution iterations
MODULE	Output	'SOLVIT'	Current solution module
NLOOP	Mandatory		Maximum allowable number of outer iterations
NRLXCC	Output		Node of RELXCC
RELXCA	Mandatory		Temperature convergence criterion
RELXCC	Output		Calculated temperature convergence
SOLVER	Output	'SSTH'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
TABS	Optional	273.15	Temperature reference

Example:

```
$EXECUTION
      RELXCA=0.001
      NLOOP=100
      CALL SOLVIT
```

Restrictions:

This routine can only be used with pure thermal models. Control constants NLOOP and RELXCA must be specified. There is no dynamic core requirement for this routine. SOLVIT may be called from the $EXECUTION block only.

6.14.8 SOLVFM

Name:

SUBROUTINE **SOLVFM**

Purpose:

This subroutine calculates the model steady state solution by full matrix inversion. The technique used is factorisation of the linearised conduction matrix into lower and upper triangular matrices followed by forward and back substitution. The non-linear nature of the radiation terms (and possibly the conductance terms) means that the matrix inversion needs to be applied iteratively with the conduction matrix being reformulated between each inversion/iteration.

The user is required to specify the maximum number of iterations (NLOOP) to be performed in attempting to reach the steady state solution relaxation criterion (RELXCA) which determines when solution has been obtained. During the solution control constants LOOPCT and RELXCC contain the number of iterations undertaken so far and the maximum temperature change from the latest iteration respectively. In addition to NLOOP and RELXCC the user may also specify one or both of INBALA - the maximum allowed energy imbalance, and INBALR - the maximum allowed relative energy imbalance. The solution converges according to RELXCA, and then the energy balance is checked against INBALA and/or INBALR. If either of these are not satisfied, then further iterations are performed and the check repeated. This continues until INBALA and INBALR are satisfied, or NLOOP is exceeded. The absolute energy imbalance is defined to be the difference between the energy into the system (by heat sources and boundary nodes) and the energy out of the system (by boundary nodes). The relative energy imbalance is the absolute imbalance as a fraction of the maximum of the energy into the system or the energy out of the system.

The user can specify the type of units used by setting the control constants STEFAN and TABS.

Control Constants:

	Status	Default	Description
DAMPT	Output	1.0	Temperature damping
ENBALA	Output		Absolute energy balance
ENBALR	Output		Relative energy balance
INBALA	Optional	1.0E10	Required absolute energy balance
INBALR	Optional	1.0	Required relative energy balance
LOOPCT	Output		Number of solution iterations
MODULE	Output	'SOLVFM'	Current solution module
NLOOP	Mandatory		Maximum allowable number of outer iterations

	Status	Default	Description
NRLXCC	Output		Node of RELXCC
RELXCA	Mandatory		Temperature convergence criterion
RELXCC	Output		Calculated temperature convergence
SOLVER	Output	'SSTH'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
TABS	Optional	273.15	Temperature reference
TIMEM	Optional		Mean time over step

Example:

```
$EXECUTION
      RELXCA=0.001
      NLOOP=100
      CALL SOLVFM
```

Restrictions:

This routine can only be used with pure thermal models. Control constants NLOOP and RELXCA must be specified. In addition to those mentioned above the following control constants are addressed by the subroutine; NRLXCC, STEFAN, TIMEM, TIMEN, TIMEO, MODULE, SOLVER.

SOLVFM may be called from the $EXECUTION block.

Numerical Background:

Calculations are carried out in double precision. The linearisation technique used to deal with the fourth power radiation terms is:

$$T_k^p = T_k^{p-1} + \Delta T_k^p$$

where superscript p represents the value at the end of iteration number p.
ΔT_k^p being small compared to T_k , it follows that:

$$(T_k^p)^4 = 4T_k^p(T_k^{p-1})^3 - 3(T_k^{p-1})^4 + O((\Delta T_k^p)^2)$$

The resulting system of equations produced is [A]T = B where the coefficients of matrix [A] and vector B are known.

The contribution of the radiation terms to the matrix [A] and the vector B are:

$$A_{ii} = 4(T_i^{p-1})^3 \sigma \Sigma_j GR_{ij}$$

$$B_i = \sigma \Sigma_j GR_{ij}(T_j^{p-1})^4 + 3\sigma \Sigma_j (T_i^{p-1})^4$$

6.14.9 SOLVCG

Name:

SUBROUTINE **SOLVCG**

Purpose:

This subroutine calculates the steady-state solution for a thermal model using the conjugate gradient (CG) iterative method. It is memory-efficient and so particulary suitable for large models.

Convergence of the solution is determined by the root-sum-square (RSS) of the nodal energy imbalances (in mathematical terms, this is the norm of the residual of the energy equation); this would be equal to zero for the exact solution. At the end of each CG iteration the calculated value (stored in control constant ENBNDM) is compared with the user-specified criterion (INBNDM)

The user is required to specify the maximum number of CG iterations (NLOOP) to be performed in attempting to obtain the solution. The number of iterations undertaken so far is given by LOOPCT. Further convergence control, relating to the global energy imbalance, is provided by the optional control constants INBALA (absolute) and INBALR (relative). If, at the end of the normal solution, these criteria are not met and the maximum number of CG iterations has not been exceeded, then the value of INBNDM is reduced locally by 25% and the CG iterations are repeated. This process continues until all convergence criteria are met or the maximum loop count has been reached.

The *absolute* global energy imbalance is defined to be the difference between the energy entering the system (from heat sources and boundary nodes) and the energy leaving the system (to boundary nodes). The *relative* energy imbalance is this value as a fraction of the larger of the two contributors (entering or leaving).

Simple damping of the change in temperature at each node over an iteration is afforded by DAMPT. Setting this to a value between 0.0 and 1.0 may improve convergence.

Note that, unlike all the other thermal solvers, SOLVCG does not address control constants RELXCA and RELXCC.

$VARIABLES1 is executed at the beginning of each iterarion, $VARIABLES2 and $OUTPUTS on completion of the solution procedure.

Control Constants:

	Status	Default	Description
DAMPT	Optional	1.0	Temperature damping
ENBALA	Output		Computed absolute global energy imbalance
ENBALR	Output		Computed relative global energy imbalance

	Status	Default	Description
ENBNDM	Output		Computed root-sum-square nodal energy imbalance
INBALA	Optional	1.0E10	Required absolute global energy imbalance
INBALR	Optional	1.0	Required relative global energy imbalance
INBNDM	Mandatory		Required root-sum-square nodal energy imbalance
LOOPCT	Output		Number of solution iterations
MODULE	Output	'SOLVCG'	Current solution module
NLOOP	Mandatory		Maximum solution iterations
SOLVER	Output	'SSTH'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
TABS	Optional	273.15	Temperature offset

Example:

```
$EXECUTION
      INBNDM=0.001
      NLOOP=1000
      CALL SOLVCG
```

Restrictions:

The solver requires a certain amount of dynamic storage to run, the amount needed being the larger of $7N_T$ and $N_T + N_G$, where N_T is the total number of thermal nodes in the model and N_G the total number of conductors (linear, radiative and fluidic). If the default amount of memory is not sufficient, more will need to be reserved by setting DYSTOR on the $EXECUTION line.

SOLVCG should be called from the $EXECUTION block and for a thermal model only.

6.14.10SLFRTF

Name:

SUBROUTINE **SLFRTF**

Purpose:

Whereas the other solvers operate in the time domain, SLFRTF performs a solution in the frequency domain. It computes the frequency response transfer function, which enables the prediction of how temperature will behave as a result of variations in boundary conditions.

A variation could be applied to either a heat load or a boundary temperature. The transfer function relates this to the resulting fluctuation in temperature at any internal (diffusion) node, assuming the input to be sinusoidal; it is therefore frequency dependent. An arbitrary input can be decomposed into sinusoidal components of different frequencies, according to Fourier analysis. The frequency response is usually considered in terms of the gain and phase of the output compared to the input.

Since the solution is not time dependent, and doesn't involve an iterative approach, SLFRTF does not execute the usual operations blocks ($VARIABLES1, $VARIABLES2, $OUTPUTS) and references very few control constants.

The computation assumes that it is starting from equilibrium conditions; hence, it is usual for a steady-state solver to be called before SLFRTF. The routine itself produces no output: in order to access the results of the transfer function calculation, a call must be made to DMPFR - see Subsection 6.2.6: “Dump Frequency Response”. EVALFR - Subsection 6.4.46: “Evaluate Thermal Frequency Response”, could also be called to access the transfer function.

Example:

Perform steady-state solution, compute the frequency response transfer function and report some results:

```
$EXECUTION
      CALL SOLVFM
      CALL SLFRTF
      CALL DMPFR('#1, 2', CURRENT,
     &           '#300', CURRENT,
     &           1.0D-6, 1.0D-1, 100,
     &           ' ')
```

Restrictions:

SLFRTF should be called only from the $EXECUTION block.

The transfer function is calculated for thermal nodes only, and fluidic conductors (GFs) are excluded.

The calculation method involves linearising the system about the equilibrium state, and so will be inaccurate if the applied or resultant temperature variations are too large. In particular, where radiative conductors are involved, the temperature variation at a node must be much less than the (absolute) temperature itself.

The calculation of transfer function is computationally intensive; for larger models a large amount of memory and CPU time will be needed.

Control Constants:

	Status	Default	Description
TABS	Optional	273.15	Temperature offset from absolute zero
STEFAN	Optional	5.6686×10^{-8}	Stefan-Boltzmann constant

6.14.11 FGENFI

Name:

SUBROUTINE **FGENFI**

Purpose:

This routine performs transient analysis on single- or two-phase thermohydraulic networks.

A fully implicit scheme, based on the scheme defined in reference[10], is employed to solve the mass, momentum and energy equations, minimising the risk of numerical instability occurring. The homogeneous equilibrium assumption is adopted for two-phase simulation. The temperatures of the solid nodes are solved using a Crank-Nicolson method, as described for SLFWBK.

The solution solves for pressure, enthalpy and mass flow rate for the fluid network and temperature for the solid network. Fluid-node temperatures and other properties are computed from pressure and enthalpy. Initial conditions defined in the model represent true transient starting conditions, and thus can lead to convergence problems if too severe. It is thus recommended that a steady-state solution (FGENSS) be run before FGENFI.

By default FGENFI performs a 'quasi-transient' solution in which the hydraulic part (i.e. pressure and mass flow) is regarded as being in steady state. If no pressure boundary is provided in a fluid loop then the mass in that loop is used as a boundary condition. However, if accurate modelling of the hydraulic response of the fluid network is considered important then a full transient, including compressible effects, can be invoked by setting control constant QTRSOL to 'NO'.

At each timestep matrix inversion is employed to solve for enthalpy, followed by a second matrix inversion solving for pressures. Flow rates are then evaluated by back-substitution, and fluid temperatures and other properties, such as density, updated. Solid-node temperatures are finally evaluated iteratively. This whole procedure is repeated until the convergence of fluid-node temperatures to FRLXCA, mass flow-rates to RELXMA, and solid-node temperatures to RELXCA has been achieved, or NLOOP iterations have been performed. (In fact, FRLXCA is applied to the change in quasi-temperature, enthalpy divided by specific heat, since temperature itself can remain constant in a two-phase fluid.) Typical values for these convergence parameters are: FRLXCA = 0.001, RELXMA = 0.001, RELXCA = 0.001, NLOOP = 100.

The goal of this routine is to achieve a solution without reducing the timestep, specified by the user with the control constant DTIMEI. (If DTIMEI is not given then the Courant limit—RLCOUR * DTCOUR, to be precise—will be used.) The timestep will, however, be reduced if convergence has not been achieved within NLOOP iterations; that is unless the timestep has reached the minimum value, DTMIN, in which case an error is recorded and solution allowed to continue.

The $VARIABLES1 block is executed: a) at the start of each outer iteration, with SOLTYP = ' '; b) before the enthalpy solution, with SOLTYP = 'THERMAL'; c) before the hydraulic solution, with SOLTYP = 'FLUID'; and d) during each iteration of the solid-node solution, with SOLTYP = 'THERMAL' again.

$VARIABLES2 is called upon completion of each timestep. The $OUTPUTS block is executed at the start of the transient, at each output interval, and at TIMEND.

Control Constants:

	Status	Default	Description
CSGMIN	Output		Minimum ratio of capacitance to sum of conductances (excluding arithmetic nodes and temperature boundaries)
DAMPM	Optional	0.5	Mass flow damping
DAMPT	Optional	1.0	Temperature damping
DTCOUR	Output		Courant timestep limit
DTIMEI	Optional	RLCOUR * DTCOUR	Input time step
DTIMEU	Output		Time step used
DTMAX	Optional	1.0E10	Maximum time step
DTMIN	Optional	0.0	Minimum time step
FRLXCA	Mandatory		Temperature convergence criterion for fluid nodes
FRLXCC	Output		Calculated temperature convergence for fluid nodes
GRAVZ	Optional	0.0	Acceleration in the z-direction
LOOPCT	Output		Number of outer solution iterations
MINDP	Optional	1.0E-3	Minimum pressure difference recognised
MINFLO	Optional	1.0E-8	Minimum flow rate for convergence check
MODULE	Output	'FGENFI'	Current solution module
NCSGMN	Output		Node of CSGMIN
NFRLXC	Output		Node of FRLXCC
NLOOP	Mandatory		Maximum allowable number of outer iterations
NLOOPT	Optional	NLOOP	No of thermal iterations allowed
NRLXCC	Output		Node of RELXCC
NRLXMC	Output		Conductor of RELXMC
OUTINT	Optional	TIMEND - TIMEO	Output interval
PABS	Optional	0.0	Reference pressure
QTRSOL	Optional	'YES'	Quasi transient solution method
RELXCA	Mandatory		Temperature convergence criterion for solid nodes
RELXCC	Output		Calculated temperature convergence for solid nodes
RELXMA	Mandatory		Convergence criterion for mass flow
RELXMC	Output		Calculated mass flow convergence

	Status	Default	Description
RLCOUR	Optional	0.5	Fraction of Courant limit for timestep
SOLTYP	Output		Inner solution type for $VARIABLES1
SOLVER	Output	'TRFL2P'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
SUMFLL	Output		Number of fluid loops taken for convergence
SUMTHL	Output		Number of thermal loops taken for convergence
TABS	Optional	273.15	Temperature reference
TIMEM	Output		Integration scheme time constant
TIMEN	Output		Time at end of step
TIMEND	Mandatory		TIme at end of solution
TIMEO	Optional	0.0	Time at start of step

Example:

```
$EXECUTION
      FRLXCA=0.001
      RELXCA=0.001
      RELXMA=0.001
      NLOOP=100
      TIMEND=100.0
      OUTINT=10.0
      CALL FGENFI
```

Restrictions:

A sparse matrix solver is employed to solve for pressure and enthalpy in order to minimise dynamic core requirements. Due to the structure of the solver it is difficult to define an explicit equation to obtain the space required; this depends upon the loop geometry and the starting node number. For most models the following expression can be used: 13*NNF + 5*NFL + 2*NNT + VSIZE, where VSIZE = (5 + maximum number of branches in any fluid loop) * NNF; one branch is considered to be a link between one part of a fluid loop and another.

If QTRSOL = 'YES', FGENFI works out the number of disjoint fluid sub-loops contained in the model. This is done at the beginning of the solution and the status of each mass flow link is assumed to be active. Thus the topology of the model should not be altered by deactivating mass flow links (see STATST, Section 6.4.30) during solution..

Convergence:

Convergence of the timestep is assumed when

$$\left|T^{k+1} - T^{k}\right| < \text{RELXCA}$$

for solid nodes, and

$$\left| \frac{h^{k+1} - h^k}{C_p^{k+1}} \right| < \text{FRLXCA}$$

$$\left| \frac{W^{k+1} - W^k}{W^{k+1}} \right| < \text{RELXMA}$$

for fluid nodes, where

k = kth estimate of quantity at TIMEN
T = temperature
h = enthalpy
C_p = specific heat
W = mass flow rate

An automatic damping procedure is carried out upon pressures, enthalpies and mass flow rates. The calculated damping factor is stored in DAMPM. Within the calculation of the temperatures of the solid nodes damping via the control constant DAMPT is used, as described for module SLFWBK.

6.14.12 FGENSS

Name:

SUBROUTINE **FGENSS**

Purpose:

Steady-state thermohydraulic analysis for single- or two-phase loops.

Enthalpies, pressures and mass flow rates are predicted for the fluid network and temperatures for the solid network.

The hydraulic solution is obtained by solving the combined momentum and mass conservation equation for pressure by matrix inversion. Mass flow rate, in each mass flow link, is obtained by back substitution. This procedure is iterated to convergence, specified by the control constant RELXMA, or until NLOOPH iterations have been performed.

A damping factor (control constant DAMPM) is used in the calculation of new mass flow rates:

Mnew = Mold + DAMPM * (Mcalc - Mold)

where:

Mnew	=	new mass flow rate
Mold	=	old mass flow rate
DAMPM	=	damping factor (default 0.5)
Mcalc	=	mass flow rate obtained by back substitution

Enthalpies of the fluid network are then evaluated via successive-point iteration. Temperature, vapour quality and density are updated at each iteration. This iterative scheme is automatically damped to help convergence, which is determined by the maximum change in temperature (or, more precisely, change in enthalpy divided by specific heat) as defined by FRLXCA. If necessary, a reduced value of FRLXCA is used internally to achieve an energy balance specified optionally by the user (INBALA/INBALR).

The successive-point solution of enthalpy is augmented by a bisection method. This is operative by default, but can be deactivated by setting the control constant METHOD to 4.

Once convergence of the enthalpy solution is reached, or NLOOPT iterations have been computed, then the temperatures of the solid network are obtained, also by successive point iteration, as described for the steady state routine SOLVIT. The number of iterations performed here is also controlled by NLOOPT; convergence is determined by RELXCA. A damping factor DAMPT is employed for the thermal solution of the solid nodes; unlike in SOLVIT, beta-optimisation is not used.

The preceding hydraulic/thermal solution procedure defines one outer iteration towards steady state and is repeated until either an overall converged solution is achieved, or the outer iteration count reaches NLOOP. To aid overall convergence, after each outer iteration enthalpies and temperatures are again automatically damped.

The $VARIABLES1 block is executed: a) at the start of each outer iteration, with SOLTYP = ' '; b) during each iteration of the hydraulic solution, with SOLTYP = 'FLUID'; c) during each iteration of the enthalpy solution, with SOLTYP = 'THERMAL'; and d) during each iteration of the solid-node solution, with SOLTYP = 'THERMAL' again.

$VARIABLES2 and $OUTPUTS are called upon completion of the solution routine.

To enable a steady-state solution to be found the required number of boundary conditions needs to be given. It is necessary to define at least one pressure boundary (R- or J-type node) for the hydraulic solution. A thermal boundary is also required; this could be either a fixed enthalpy (an R- or K-node), or a solid B-node.

The user is required to input values of the convergence parameters RELXCA, FRLXCA, and RELXMA, plus the iteration-count limit NLOOP. Typical values would be: 0.001, 0.001, 0.001 and 100, respectively. We advocate that NLOOPT and NLOOPH be set to values much lower than NLOOP in order to improve convergence between all the equations; a value of 10 is recommended for both. It is also a good idea to specify a relative energy balance (INBALR) as well as the standard convergence criteria. However, it must be realised that only NLOOP iterations will be performed and users should always examine the energy balance given at the end of the solution.

Control Constants:

	Status	Default	Description
DAMPM	Optional	0.5	Mass flow damping
DAMPT	Optional	1.0	Temperature damping
ENBALA	Output		Absolute energy balance
ENBALR	Output		Relative energy balance
FRLXCA	Mandatory		Temperature convergence criterion for fluid nodes
FRLXCC	Output		Calculated temperature convergence for fluid nodes
GRAVZ	Optional	0.0	Acceleration in the z-direction
INBALA	Optional	1.0E10	Required absolute energy balance
INBALR	Optional	1.0	Required relative energy balance
LOOPCT	Output		Number of outer solution iterations
MINDP	Optional	1.0E-3	Minimum pressure difference recognised
MINFLO	Optional	1.0E-8	Minimum flow rate for convergence check

	Status	Default	Description
MODULE	Output	'FGENSS'	Current solution module
NFRLXC	Output		Node of FRLXCC
NLOOP	Mandatory		Maximum allowable number of outer iterations
NLOOPH	Optional	NLOOP	No of hydraulic iterations allowed
NLOOPT	Optional	NLOOP	No of thermal iterations allowed
NRLXCC	Output		Node of RELXCC
NRLXMC	Output		Conductor of RELXMC
PABS	Optional	0.0	Reference pressure
RELXCA	Mandatory		Temperature convergence criterion for solid nodes
RELXCC	Output		Calculated temperature convergence for solid nodes
RELXMA	Mandatory		Convergence criterion for mass flow
RELXMC	Output		Calculated mass flow convergence
SOLTYP	Output		Inner solution type for $VARIABLES1
SOLVER	Output	'SSFL2P'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
SUMFLL	Output		Number of fluid loops taken for convergence
SUMTHL	Output		Number of thermal loops taken for convergence
TABS	Optional	273.15	Temperature reference

Example:

```
$EXECUTION
      FRLXCA=0.001
      RELXCA=0.001
      RELXMA=0.001
      NLOOP=100
      CALL FGENSS
```

Restrictions:

A sparse matrix solver is employed to solve for pressure in order to minimise dynamic core requirements. Due to the structure of the solver it is difficult to define an explicit equation to obtain the space required; this depends upon the loop geometry and the starting node number. For most models the following equation can be used: 4*NNF + 3*NFL * VSIZE, where VSIZE = (5 + maximum number of branches in any fluid loop) * NNF; one branch is considered to be a link between one part of a fluid loop and another.

6.14.13 FLTMTS

Name:

SUBROUTINE **FLTMTS**

Purpose:

Multi-timestepping transient analysis of single-phase thermohydraulic loops.

Temperature, pressure and mass flow rate are solved for at fluid nodes, temperature at solid nodes. Also, humidity and condensation rates are predicted in air/water-vapour loops.

The thermal solution technique is that of the implicit forward-backward differencing scheme, the Crank-Nicolson method. The hydraulic solution is obtained by matrix inversion and iteration at each timestep as described for the routine FLTNSS.

This routine is essentially identical in operation to FLTNTS, except that several fluid timesteps are performed to each solid timestep. The user is required to specify the solid timestep via the control constant DTIMEI, and the (integer) number of fluid timesteps per solid via FSTEPS. The fluid timestep is then given by DTIMEI / FSTEPS. It is possible to use a fluid timestep greater than that imposed by the Courant number restriction; however the user should consider this restriction when selecting FSTEPS. The solid timesteps are made to land exactly on an output interval, by adjusting the last two step lengths where necessary. Similarly, the fluid timesteps will land exactly on each solid one. All other details, regarding start and end times, and the use of DTMIN, DTMAX and DTPMAX to control the timestep, are as described for the solution routine SLFWBK; note however that DTPMAX refers to the change in fluid node temperatures only.

For fluid type AIRW, humidity at each node is found by performing a mass-balance on the water vapour. Condensation rates are determined at each fluid node linked by a GL conductor to a thermal node; this calculation also uses DAMPM. See Section 4.4 for more details

The $VARIABLES1 block is executed: a) at the start of each outer iteration, with SOLTYP = ’ ’; b) during each iteration of the hydraulic solution, with SOLTYP = ’FLUID’; c) during each iteration of the thermal solution, with SOLTYP = ’THERMAL’; and d) during each iteration of the humidity solution, with SOLTYP = ’HUMID’.

$VARIABLES2 is called upon completion of each timestep. The $OUTPUTS block is executed at the start of the transient, at each output interval, and at TIMEND.

When setting the control constant TIMELA to 'YES' the solution routine is able to simulate the timelag effect, thus avoiding false diffusion as described in the

Engineering Manual. During timelag simulation the implemented algorithm tries to minimize false diffusion by referencing fluid node temperatures stored back in time.

Control Constants:

	Status	Default	Description
CSGMIN	Output		Minimum ratio of capacitance to sum of conductances (excluding arithmetic nodes and temperature boundaries)
DAMPM	Optional	0.5	Mass flow damping
DAMPT	Optional	1.0	Temperature damping
DTCOUR	Output		Courant timestep limit
DTIMEI	Mandatory		Input time step
DTIMEU	Output		Time step used
DTMAX	Optional	1.0E10	Maximum time step
DTMIN	Optional	0.0	Minimum time step
DTPMAX	Optional	1.0E10	Maximum allowable temperature change over a time step (non-arithmetic diffusion nodes only)
FCSGMN	Output		Minimum ratio of capacitance to sum of conductances for fluid nodes (excluding arithmetic nodes and temperature boundaries)
FNCSGM	Output		Fluid node of FCSGMN
FRLXCA	Mandatory		Temperature convergence criterion for fluid nodes
FRLXCC	Output		Calculated temperature convergence for fluid nodes
FSTEPS	Mandatory		No of fluid steps per solid step
GRAVX	Optional	0.0	Acceleration in the x-direction
GRAVY	Optional	0.0	Acceleration in the y-direction
GRAVZ	Optional	0.0	Acceleration in the z-direction
LOOPCT	Output		Number of outer solution iterations
MINDP	Optional	1.0E-3	Minimum pressure difference recognised
MINFLO	Optional	1.0E-8	Minimum flow rate for convergence check
MODULE	Output	’FLTMTS’	Current solution module
NCSGMN	Output		Node of CSGMIN
NFRLXC	Output		Node of FRLXCC
NLOOP	Mandatory		Maximum allowable number of outer iterations
NLOOPH	Optional	NLOOP	No of hydraulic iterations allowed
NLOOPT	Optional	NLOOP	No of thermal iterations allowed
NRLXCC	Output		Node of RELXCC
NRLXMC	Output		Conductor of RELXMC
OUTINT	Optional	TIMEND - TIMEO	Output interval
PABS	Optional	0.0	Reference pressure
RELXCA	Mandatory		Temperature convergence criterion for solid nodes

	Status	Default	Description
RELXCC	Output		Calculated temperature convergence for solid nodes
RELXMA	Mandatory		Convergence criterion for mass flow
RELXMC	Output		Calculated mass flow convergence
SOLTYP	Output		Inner solution type for $VARIABLES1
SOLVER	Output	'TRFL1P'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
SUMFLL	Output		Number of fluid loops taken for convergence
SUMTHL	Output		Number of thermal loops taken for convergence
TABS	Optional	273.15	Temperature reference
TIMELA	Optional	'NO'	Indicator for time lag simulation
TIMEM	Output		Mean time over step
TIMEN	Output		Time at end of step
TIMEND	Mandatory		TIme at end of solution
TIMEO	Optional	0.0	Time at start of step

Example:

```
$EXECUTION
      FRLXCA=0.001
      RELXCA=0.001
      RELXMA=0.001
      NLOOP=100
      FSTEPS=5
      DTIMEI=1.0
      TIMEND=100.0
      OUTINT=10.0
      CALL FLTMTS
```

Restrictions:

Note that DTIMEU is the **fluid** timestep used.

A sparse matrix solver is employed to solve for pressure in order to minimise dynamic core requirements. Due to the structure of the solver it is difficult to define an explicit equation to obtain the space required; this depends upon the loop geometry and the starting node number. For most models the following equation can be used, 12*NNF + 2*NFL + 4*NNT + VSIZE + 5, where VSIZE = (5 + maximum number of branches in any fluid loop) * NNF; one branch is considered to be a link between one part of a fluid loop and another. When the timelag facility is to be used a further (NNF + 1)*100 locations must be reserved. If DYSTOR is defined too low an error is recorded, reporting the additional space required, and solution halted.

The timelag simulation may be activated only at the start of a transient; setting TIMELA = 'YES" during solution will cause an error. The maximum number of previous timesteps which can be referenced is 100.

6.14.14 FLTNSS

Name:

SUBROUTINE **FLTNSS**

Purpose:

Steady state analysis of single-phase thermohydraulic loops.

Temperature, pressure and mass flow rate are solved for at fluid nodes, temperature at solid nodes. Also, humidity and condensation rates are predicted in air/water-vapour loops.

The hydraulic solution is obtained by solving the combined momentum and mass conservation equation for pressure by matrix inversion. Mass flow rate, in each mass flow link, is obtained by back substitution. This procedure is iterated to convergence or until NLOOPH is exceeded.

A damping factor (control constant DAMPM) is in the calculation of new mass flow rates:

$$M_{new} = M_{old} + DAMPM \times (M_{calc} - M_{old})$$

where:

M_{new}	=	new mass flow rate
M_{old}	=	old mass flow rate
DAMPM	=	damping factor (default 0.5)
M_{calc}	=	mass flow rate obtained by back substitution

Temperature for the entire network (thermal and fluid nodes) is obtained by successive-point iteration. Similarly to the hydraulic solution, a damping factor DAMPT is employed; however, if this is not reset from the default value of 1.0 then 'beta optimisation' is used.

For fluid type AIRW, humidity at each node is found by performing a mass-balance on the water vapour. Condensation rates are determined at each fluid node linked by a GL conductor to a thermal node; this calculation also uses DAMPM. See Section 4.4 for more details.

The preceding hydraulic/thermal/humidity solution procedure defines one outer iteration towards steady state and is repeated until a converged solution is achieved or NLOOP is exceeded.

The $VARIABLES1 block is executed: a) at the start of each outer iteration, with SOLTYP = ' '; b) during each iteration of the hydraulic solution, with SOLTYP = 'FLUID'; c) during each iteration of the thermal solution, with SOLTYP = 'THERMAL'; and d) during each iteration of the humidity solution, with SOLTYP = 'HUMID'.

$VARIABLES2 and $OUTPUTS are called upon completion of the solution routine.

Further control over the thermal solution is afforded by the control constant METHOD, which takes the value 0 or 1. The default is 0; setting METHOD=1 signals that, for each pass of the thermal solution, only those nodes (fluid or solid) at which temperature has not converged are to be iterated on. In many circumstances this has the effect of speeding up the solution. For example, models containing independent fluid loops or models where a section of the loop converges slowly are likely to benefit from the selective convergence scheme.

Convergence of the hydraulic solution is achieved when RELXMC is less than RELXMA. RELXMC is calculated as the maximum for all links of mass flow rate change during an iteration divided by latest mass flow rate. RELXMA is a user-given convergence criterion. NRLXMC is the mass flow link of RELXMC. The thermal convergence is assumed when the maximum solid node temperature change between iterations (RELXCC) is less than RELXCA, and the maximum fluid node temperature change between iterations (FRLXCC) is less than FRLXCA. RELXCA and FRLXCA are user-given convergence criteria. Control constants NRLXCC and NFRLXC are the nodes of RELXCC and FRLXCC respectively.

The user is required to input values of the control constants RELXCA, FRLXCA, and RELXMA, plus the maximum allowable number of iterations NLOOP.

Control Constants:

	Status	Default	Description
DAMPM	Optional	0.5	Mass flow damping
DAMPT	Optional	1.0	Temperature damping
ENBALA	Output		Absolute energy balance
ENBALR	Output		Relative energy balance
FRLXCA	Mandatory		Temperature convergence criterion for fluid nodes
FRLXCC	Output		Calculated temperature convergence for fluid nodes
GRAVX	Optional	0.0	Acceleration in the x-direction
GRAVY	Optional	0.0	Acceleration in the y-direction
GRAVZ	Optional	0.0	Acceleration in the z-direction
INBALA	Optional	1.0E10	Required absolute energy balance
INBALR	Optional	1.0	Required relative energy balance
LOOPCT	Output		Number of outer solution iterations
METHOD	Optional	0	Selective convergence scheme flag
MINDP	Optional	1.0E-3	Minimum pressure difference recognised
MINFLO	Optional	1.0E-8	Minimum flow rate for convergence check
MODULE	Output	'FLTNSS'	Current solution module

	Status	Default	Description
NFRLXC	Output		Node of FRLXCC
NLOOP	Mandatory		Maximum allowable number of outer iterations
NLOOPH	Optional	NLOOP	No of hydraulic iterations allowed
NLOOPT	Optional	NLOOP	No of thermal iterations allowed
NRLXCC	Output		Node of RELXCC
NRLXMC	Output		Conductor of RELXMC
PABS	Optional	0.0	Reference pressure
RELXCA	Mandatory		Temperature convergence criterion for solid nodes
RELXCC	Output		Calculated temperature convergence for solid nodes
RELXMA	Mandatory		Convergence criterion for mass flow
RELXMC	Output		Calculated mass flow convergence
SOLTYP	Output		Inner solution type for $VARIABLES1
SOLVER	Output	'SSFL1P'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
SUMFLL	Output		Number of fluid loops taken for convergence
SUMTHL	Output		Number of thermal loops taken for convergence
TABS	Optional	273.15	Temperature reference

Example:

```
$EXECUTION
      FRLXCA=0.001
      RELXCA=0.001
      RELXMA=0.001
      NLOOP=100
      CALL FLTNSS
```

Restrictions:

A sparse matrix solver is employed to solve for pressure in order to minimise dynamic core requirements. Due to the structure of the solver it is difficult to define an explicit equation to obtain the space required; this depends upon the loop geometry and the starting node number. For most models the following equation can be used, 6*NNF + 2*NFL + NNT + VSIZE. where VSIZE = (5 + maximum number of branches in any fluid loop) * NNF. One branch considered to be a link between one part of a fluid loop to another. If DYSTOR is defined too low an error is recorded, reporting the additional space required, and solution halted.

6.14.15 FLTNTF

Name:

SUBROUTINE **FLTNTF**

Purpose:

Explicit transient analysis of single-phase thermohydraulic loops.

Temperature, pressure and mass flow rate are solved for at fluid nodes, temperature at solid nodes. Also, humidity and condensation rates are predicted in air/water-vapour loops.

The thermal solution technique is that of explicit forward differencing (as SLFRWD), the hydraulic solution is obtained by matrix inversion and iteration at each timestep as described for the routine FLTNSS.

The timestep used in this solution is the minimum of (0.95*CSGMIN) and (RLCOUR*DTCOUR). DTCOUR is the maximum time step related to the Courant number restriction, for all links. RLCOUR is a damping factor for the Courant limited timestep. All other details, as regards start and end times, and the use of DTMIN, DTMAX and DTPMAX to control the timestep, are as described for the solution routine SLFRWD.

For fluid type AIRW, humidity at each node is found by performing a mass-balance on the water vapour. Condensation rates are determined at each fluid node linked by a GL conductor to a thermal node; this calculation also uses DAMPM. See Section 4.4 for more details

The $VARIABLES1 block is executed: a) at the start of each outer iteration, with SOLTYP = ’ ’; b) during each iteration of the hydraulic solution, with SOLTYP = ’FLUID’; and c) during the thermal solution, with SOLTYP = ’THERMAL’; and d) during the humidity solution, with SOLTYP = ’HUMID’.

$VARIABLES2 is called upon completion of each timestep. The $OUTPUTS block is executed at the start of the transient, at each output interval, and at TIMEND.

When setting the control constant TIMELA to ’YES’ the solution routine is able to simulate the timelag effect, thus avoiding false diffusion as described in the Engineering Manual. During timelag simulation the implemented algorithm tries to minimize false diffusion by referencing fluid node temperatures stored back in time.

Control Constants:

	Status	Default	Description
CSGMIN	Output		Minimum ratio of capacitance to sum of conductances (excluding arithmetic nodes and temperature boundaries)
DAMPM	Optional	0.5	Mass flow damping
DTCOUR	Output		Courant timestep limit
DTIMEI	Output		Initial time step calculated
DTIMEU	Output		Time step used
DTMAX	Optional	1.0E10	Maximum time step
DTMIN	Optional	0.0	Minimum time step
DTPMAX	Optional	1.0E10	Maximum allowable temperature change over a time step (non-arithmetic diffusion nodes only)
FRLXCA	Mandatory		Temperature convergence criterion for fluid nodes
FRLXCC	Output		Calculated temperature convergence for fluid nodes
GRAVX	Optional	0.0	Acceleration in the x-direction
GRAVY	Optional	0.0	Acceleration in the y-direction
GRAVZ	Optional	0.0	Acceleration in the z-direction
LOOPCT	Output		Number of outer solution iterations
MINDP	Optional	1.0E-3	Minimum pressure difference recognised
MINFLO	Optional	1.0E-8	Minimum flow rate for convergence check
MODULE	Output	'FLTNTF'	Current solution module
NCSGMN	Output		Node of CSGMIN
NFRLXC	Output		Node of FRLXCC
NLOOP	Mandatory		Maximum allowable number of outer iterations
NLOOPH	Optional	NLOOP	No. of hydraulic iterations allowed
NLOOPT	Optional	NLOOP	No. of thermal iterations allowed
NRLXCC	Output		Node of RELXCC
NRLXMC	Output		Conductor of RELXMC
OUTINT	Optional	TIMEND - TIMEO	Output interval
PABS	Optional	0.0	Reference pressure
RELXCA	Mandatory		Temperature convergence criterion for solid nodes
RELXCC	Output		Calculated temperature convergence for solid nodes
RELXMA	Mandatory		Convergence criterion for mass flow
RELXMC	Output		Calculated mass flow convergence
RLCOUR	Optional	0.5	Fraction of Courant limit for time step
SOLTYP	Output		Inner solution type for $VARIABLES1
SOLVER	Output	'TRFL1P'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
SUMFLL	Output		Number of fluid loops taken for convergence

	Status	Default	Description
SUMTHL	Output		Number of thermal loops taken for convergence
TABS	Optional	273.15	Temperature reference
TIMELA	Optional	'NO'	Indicator for time lag simulation
TIMEM	Output		Mean time over step
TIMEN	Output		Time at end of step
TIMEND	Mandatory		TIme at end of solution
TIMEO	Optional	0.0	Time at start of step

Example:

```
$EXECUTION
      FRLXCA=0.001
      RELXCA=0.001
      RELXMA=0.001
      NLOOP=100
      TIMEND=100.0
      OUTINT=10.0
      CALL FLTNTF
```

Restrictions:

A sparse matrix solver is employed to solve for pressure in order to minimise dynamic core requirements. Due to the structure of the solver it is difficult to define an explicit equation to obtain the space required; this depends upon the loop geometry and the starting node number. For most models the following equation can be used: the dynamic core required is the maximum of (5*NNF + 2*NFL + VSIZE) and (6*NNF + 2*NNT + 3 + VSIZE), where VSIZE = (5 + maximum number of branches in any fluid loop) * NNF; one branch is considered to be a link between one part of a fluid loop and another. When the timelag facility is to be used a further (NNF + 1)*100 locations must be reserved. If DYSTOR is defined too low an error is recorded, reporting the additional space required, and solution halted.

The timelag simulation may be activated only at the start of a transient; setting TIMELA = 'YES" during solution will cause an error. The maximum number of previous timesteps which can be referenced is 100.

6.14.16 FLTNTS

Name:

SUBROUTINE **FLTNTS**

Purpose:

Implicit transient analysis of single-phase thermohydraulic loops.

Temperature, pressure and mass flow rate are solved for at fluid nodes, temperature at solid nodes. Also, humidity and condensation rates are predicted in air/water-vapour loops.

The thermal solution technique is an iterative formulation of the implicit forward-backward differencing scheme, the Crank-Nicolson method. The hydraulic solution is obtained by matrix inversion and iteration at each timestep as described for the routine FLTNSS.

The timestep used in this solution is that given by the user on control constant DTIMEI. It is possible to use timesteps greater than that imposed by the Courant number restriction; however the user should consider this restriction when selecting a suitable timestep value. All other details, regarding start and end times, and the use of DTMIN, DTMAX and DTPMAX to control the timestep, are as described for the solution routine SLFWBK.

For fluid type AIRW, humidity at each node is found by performing a mass-balance on the water vapour. Condensation rates are determined at each fluid node linked by a GL conductor to a thermal node; this calculation also uses DAMPM. See Section 4.4 for more details

The $VARIABLES1 block is executed: a) at the start of each outer iteration, with SOLTYP = ’ ’; b) during each iteration of the hydraulic solution, with SOLTYP = ’FLUID’; and c) during each iteration of the thermal solution, with SOLTYP = ’THERMAL’; and d) during each iteration of the humidity solution, with SOLTYP = ’HUMID’.

$VARIABLES2 is called upon completion of each timestep. The $OUTPUTS block is executed at the start of the transient, at each output interval, and at TIMEND.

A measure of control over the thermal solution is afforded by the control constant METHOD, which takes the value 0 or 1. The default is 0; setting METHOD=1 signals that, for each pass of the thermal solution, only those nodes (fluid or solid) at which temperature has not converged are to be iterated on. In many circumstances this has the effect of speeding up the solution. For example, models containing independent fluid loops or models where a section of the loop converges slowly are likely to benefit from the selective convergence scheme.

When setting the control constant TIMELA to 'YES' the solution routine is able to simulate the timelag effect, thus avoiding false diffusion as described in the Engineering Manual. During timelag simulation the implemented algorithm tries to minimize false diffusion by referencing fluid node temperatures stored back in time.

Control Constants:

	Status	Default	Description
CSGMIN	Output		Minimum ratio of capacitance to sum of conductances (excluding arithmetic nodes and temperature boundaries)
DAMPM	Optional	0.5	Mass flow damping
DAMPT	Optional	1.0	Temperature damping
DTCOUR	Output		Courant timestep limit
DTIMEI	Mandatory		Input time step
DTIMEU	Output		Time step used
DTMAX	Optional	1.0E10	Maximum time step
DTMIN	Optional	0.0	Minimum time step
DTPMAX	Optional	1.0E10	Maximum allowable temperature change over a time step (non-arithmetic diffusion nodes only)
FRLXCA	Mandatory		Temperature convergence criterion for fluid nodes
FRLXCC	Output		Calculated temperature convergence for fluid nodes
GRAVX	Optional	0.0	Acceleration in the x-direction
GRAVY	Optional	0.0	Acceleration in the y-direction
GRAVZ	Optional	0.0	Acceleration in the z-direction
LOOPCT	Output		Number of outer solution iterations
METHOD	Optional	0	Selective convergence scheme flag
MINDP	Optional	1.0E-3	Minimum pressure difference recognised
MINFLO	Optional	1.0E-8	Minimum flow rate for convergence check
MODULE	Output	'FLTNTS'	Current solution module
NCSGMN	Output		Node of CSGMIN
NFRLXC	Output		Node of FRLXCC
NLOOP	Mandatory		Maximum allowable number of outer iterations
NLOOPH	Optional	NLOOP	No. of hydraulic iterations allowed
NLOOPT	Optional	NLOOP	No. of thermal iterations allowed
NRLXCC	Output		Node of RELXCC
NRLXMC	Output		Conductor of RELXMC
OUTINT	Optional	TIMEND - TIMEO	Output interval
PABS	Optional	0.0	Reference pressure
RELXCA	Mandatory		Temperature convergence criterion for solid nodes
RELXCC	Output		Calculated temperature convergence for solid nodes
RELXMA	Mandatory		Convergence criterion for mass flow

	Status	Default	Description
RELXMC	Output		Calculated mass flow convergence
SOLTYP	Output		Inner solution type for $VARIABLES1
SOLVER	Output	'TRFL1P'	Solver type
STEFAN	Optional	5.6686E-8	Stefan-Boltzman constant
STEPCT	Output		Time step count
SUMFLL	Output		Number of fluid loops taken for convergence
SUMTHL	Output		Number of thermal loops taken for convergence
TABS	Optional	273.15	Temperature reference
TIMELA	Optional	'NO'	Indicator for time lag simulation
TIMEM	Output		Mean time over step
TIMEN	Output		Time at end of step
TIMEND	Mandatory		TIme at end of solution
TIMEO	Optional	0.0	Time at start of step

Example:

```
$EXECUTION
      FRLXCA=0.001
      RELXCA=0.001
      RELXMA=0.001
      NLOOP=100
      DTIMEI=1.0
      TIMEND=100.0
      OUTINT=10.0
      CALL FLTNTS
```

Restrictions:

A sparse matrix solver is employed to solve for pressure in order to minimise dynamic core requirements. Due to the structure of the solver it is difficult to define an explicit equation to obtain the space required; this depends upon the loop geometry and the starting node number. For most models the following equation can be used: The dynamic core required is the maximum of (7*NNF + 3*NNT + VSIZE + 4) and (5*NNF + 2*NFL + VSIZE). where VSIZE = (5 + maximum number of branches in any fluid loop) * NNF; one branch is considered to be a link between one part of a fluid loop and another. When the timelag facility is to be used a further (NNF + 1)*100 locations must be reserved. If DYSTOR is defined too low an error is recorded, reporting the additional space required, and solution halted.

The timelag simulation may be activated only at the start of a transient; setting TIMELA = 'YES" during solution will cause an error. The maximum number of previous timesteps which can be referenced is 100.

6.14.17 SOLCYC

Name:

SUBROUTINE **SOLCYC**(SOLNAM, CVTCA, CVDTCA, PERIOD, MAXCYC, ZLABEL, OUTIML)

SOLNAM	-	Character: underlying transient solver to be used
CVTCA	-	Double precision: cycle convergence criterion for temperature
CVDTCA	-	Double precision: cycle convergence criterion for rate of change of temperature
PERIOD	-	Double precision: period of cycle (seconds)
MAXCYC	-	Integer: maximum number of cycles to perform
ZLABEL	-	Character: specifies nodes for which criteria shall be met
OUTIML	-	Character: defines whether normal output is required ('NONE' or 'ALL')

Purpose:

SOLCYC is a 'meta-solver', the purpose of which is to attain a steady cyclic solution; i.e. successive cycles of a transient analysis giving the same thermal results to within user-specified criteria.

This is achieved by consecutive runs of a standard transient solver, given by SOLNAM, each run lasting for one period and starting from the end conditions of the previous cycle. On completing a cycle the temperature at each node is compared with its value at the end of the previous cycle. Similarly, the rate of change (i.e. time derivative) of temperature at each node is calculated and the change over the cycle determined. Mathematically it can be shown that, given certain reasonable assumptions, for the temperature of a node to be a periodic function of time both the temperature itself and its first derivative must repeat at successive times separated by one period. Hence, if at all nodes the magnitudes of both ΔT and $\Delta(dT/dt)$ are less than the user-specified criteria (CVTCA and CVDTCA, respectively) then cyclic convergence is said to have been achieved.

A report is made at the end of each cycle giving a) the number of the cycle; b) the maximum ΔT and the node at which this was found; and c) the maximum $\Delta(dT/dt)$ and the associated node. This report will appear in either the output file or the trace file, depending on the value of the control constant TRACE (Section 6.3.3).

If convergence has not been attained within MAXCYC cycles, the solution will halt with an appropriate message. The set of nodes tested can be restricted by specifying ZLABEL appropriately (see Section 6.1.3). The user must specify whether normal output is required while SOLCYC is running with the argument OUTIML, which will locally override the value of the control constant OUTIME. That is, if OUTIML is set to 'NONE', the contents of the $OUTPUTS block will not be

executed during SOLCYC; if set to ’ALL’, the $OUTPUTS block will be executed at every output interval.

The user must define all the control constants normally required by the underlying solver, such as RELXCA and NLOOP. However, the solution end time, TIMEND, is set by SOLCYC at the start of each cycle to TIMEO + PERIOD, where TIMEO is the timestep start time and is reset to its original value at the end of each cycle. In other words, for each cycle the solution time, TIMEN, starts at TIMEO and runs for PERIOD seconds to TIMEO + PERIOD. Both TIMEO and TIMEND are restored to their original values when SOLCYC exits.

The rate of change of temperature is computed by evaluating the heat fluxes on each node; it is assumed to be zero at all boundary, arithmetic and inactive nodes. In practice, especially for larger models, it may not be necessary to perform the test on dT/dt; this will therefore be omitted if CVDTCA is 1.0D10 or more.

SOLCYC can be used with either thermal or thermo-hydraulic models. Note, however, that for FHTS models it is temperature only that is monitored for steady cyclic behaviour: pressure and flow rate are not checked. While the thermal calculation is still valid for single-phase solution modules, as the two-phase modules solve for enthalpy not temperature the time-derivative calculation is inapplicable and so should be disabled by setting CVDTCA ≥ 1.0D10.

Example:

```
$EXECUTION
#
# Attain steady cycles
#
      CALL SOLCYC('SLFWBK', 0.01D0, 0.05D0, 3600.0D0,
     &            10, ' ', 'NONE')
#
# Run for one more cycle to get output
#
      CALL SLFWBK
#
$OUTPUTS
#
# Output will not appear during SOLCYC
#
      CALL PRNDTB(’ ’, ’T’, CURRENT)
#
```

Restrictions:

This routine may be called only from the $EXECUTION block. The user is required to specify all the usual control constants for the solver selected by SOLNAM. For FHTS models the hydraulic solution is not checked for steady cycles; for two-phase solutions convergence on the temperature time-derivative is inapplicable.

SOLCYC itself requires 2*NNT + 2*NNF + 2 locations of dynamic core requirements, where NNT is the number of thermal nodes and NNF the number of fluid nodes in the model. This is in addition to the space needed by the underlying solver.

Control Constants:

	Status	Default	Description
MODULE	Output	'SOLCYC'	Current solution module
...	...	...	[See particular solver]

6.15 Index of Library Routines

7. EXAMPLES

7.1 Thermal model

7.1.1 Introduction

The example is intended to illustrate some of the features of ESATAN. The model described consists of a main model containing two submodels, and is not intended to have any significant physical meaning.

The following listings are provided:-

1) Input file
2) Preprocessor log file
3) Solution output
4) Generated Fortran program

7.1.2 Description of Listings

1) **Model input data:** This defines the model and specifies the solution control.

2) **Preprocessor log file:** This file is created during a preprocessing run. It is a direct reflection of the user's input, with preprocessor warning and error messages given immediately following the erroneous definition or statement. Additional summary information is given at the end of each submodel.

   (Statements giving rise to warning messages have been deliberately included in the input file for the current example in order to illustrate this feature of the log file.)

3) **Solution output:** The given example shows usage of table format entity output. The user is referred to the $EXECUTION and $OUTPUTS blocks of THRUSTER for explanation of the output control.

4) **Generated Fortran program:** The Fortran source code derived from the Model Database (MDB) by the ESAFOR Fortran generation process is reproduced here to satisfy the user's curiosity as to the results of a preprocessed ESATAN model. It is not intended that the user should gain a detailed understanding of this program, or, indeed, concern himself with its existence as a general principle. However, it is felt that it may be useful in this instance to include the Fortran, for any user who wishes to study the execution program organisation.

   The common blocks have been included in the main program, but edited out of all functions and subroutines for the sake of clarity and to avoid unnecessary repetition.

7.1.2.1 THRUSTER Model Input Data

```
$MODEL Thruster
#
# RCSID: $Id: listings/thruster.d 1.6 2006/07/21 17:58:02BST jstrutt Released  $
#
# This model contains two, nearly identical submodels.
# It has only little physical meaning, but is primarily
# designed to show the basic features of ESATAN.
#
$MODEL Thruster1
#
# This model is derived from the hydrazine thruster model.
#
$NODES
D1 = 'nozzle', 20., 4.6;
D3 = 'thruster ins.', 20., 8.0;
D4 = 'head plate', 20., 3.5;
D5 = 'heat barrier outer', 20., 1.9;
D6 = 'heat barrier middle', 20., 1.9;
D7 = 'heat barrier inner', 20., 1.9;
D8 = 'injection tube outer', 20., 0.176;
D9 = 'injection tube middle', 20., 0.68;
D10 = 'injection tube inner', 20., 0.065;
D11 = 'cartridge heat.ntwrk out', 20., 7.28;
D12 = 'cartridge heat.ntwrk inn', 20., 12.72;
D13 = 'heat barrier interface', 20., 14.6;
D14 = 'valve flange', 20., 10.35;
D15 = 'valve body', 20., 54.83;
D2 = 'decomposition chamber', 200., 25.;
B105 = 'S/C node 105', 20.;
B999 = 'space', -270.16;
#
$CONDUCTORS
GL(  1,  2) = 0.0015    *   INTRP1(TAV(T1, T2), Conductivity, 1);
GL(  2,  3) = 0.0085;
GL(  2,  4) = 0.0053    *   INTRP1(TAV(T2, T4), Conductivity, 1);
GL(  4,  5) = 4.32E-3   *   INTRP1(TAV(T4, T5), Conductivity, 1);
GL(  5,  6) = 8.23E-4   *   INTRP1(TAV(T5, T6), Conductivity, 1);
GL(  6,  7) = 8.23E-4   *   INTRP1(TAV(T6, T7), Conductivity, 1);
GL(  7, 13) = 4.32E-3   *   INTRP1(TAV(T7, T13), Conductivity, 1);
GL( 13, 14) = 7.00E-3   *   INTRP1(TAV(T13, T14), Conductivity, 1);
GL( 14, 15) = 0.15;
GL(  2,  8) = 2.361E-4  *   INTRP1(TAV(T2, T8), Conductivity, 1);
GL(  8,  9) = 5.059E-5  *   INTRP1(TAV(T8, T9), Conductivity, 1);
GL(  9, 10) = 4.086E-5  *   INTRP1(TAV(T9, T10), Conductivity, 1);
GL( 10, 13) = 1.518E-4  *   INTRP1(TAV(T10, T13), Conductivity, 1);
GL(  2, 11) = 0.0041;
GL( 11, 12) = 0.0041;
GL( 12, 15) = 0.0024;
GL( 14,105) = 0.069;
GR(  1, 999) = 2.1324E-4;
GR(  3, 999) = 6.1080E-4;
GR(  4, 999) = 7.2000E-5;
GR(  5, 999) = 3.9000E-5;
GR(  6, 999) = 3.9000E-5;
GR(  8, 999) = 6.0000E-6;
GR(  9, 999) = 2.3200E-5;
GR( 11, 999) = 2.0600E-4;
$CONSTANTS
$CONTROL
RELXCA = 0.001;
NLOOP  = 1000;
$ARRAYS
REAL*Conductivity (2,2) =
```

```
   0.0, 9.5 ,
   400., 16.6;
#
$SUBROUTINES
      DOUBLE PRECISION FUNCTION TAV(XX, YY)
      DOUBLE PRECISION XX, YY
      TAV = (XX + YY) / 2.0D0
      RETURN
      END
$INITIAL
#
# Simulate catalyst bed heating
#
      CALL STATST('N2', 'B')
#
$ENDMODEL Thruster1
#
$MODEL Thruster2
$REPEAT Thruster1
#
# The Thruster1 model is copied.
# Small changes are effectively made in the main model $INITIAL block,
# and additional control logic included in the main model $VARIABLES1.
#
$ENDMODEL Thruster2
#
# Now starts the overall model, which is trivial.  Only node
# 999 (space) is made identical and 1 coupling added.
# In fact, the two thruster models run nearly independently.
#
$NODES
B999 = 'space' = Thruster1:999 + Thruster2:999,T = -270.16;
#
$CONDUCTORS
GR(Thruster1:13, Thruster2:13) = 0.000005;   # A VERY SMALL VALUE
#
$CONSTANTS
$CONTROL
RELXCA = 0.001;
NLOOP  = 1000;
TIMEND = 1000.;
OUTINT = 500.;
DTIMEI = 10.;
$INTEGER
Switch = 1;
#
$INITIAL
      T:Thruster2:105 = 0.0
#
$EXECUTION
      INTEGER I
#
# Do 3 steady states in a WHILE loop, if switch=1.
# The temperature of node 999 is varied.
#
      IF (Switch .EQ. 1) THEN
#
         I = 1
         WHILE (I .LE. 3)
            SELECT CASE I
                CASE 1
                   HEADER = 'STEADY STATE - CASE 1'
#                  no action taken
                CASE 2
                   T999 = 0.0D0
```

```
                  QI:Thruster1:5 = 2.50D0
                  QI:Thruster1:9 = 2.50D0
                  HEADER = 'STEADY STATE - CASE 2'
               CASE 3
                  T999 = 100.0D0
                  QI:Thruster1:5 = 1.50D0
                  QI:Thruster1:9 = 1.50D0
                  HEADER = 'STEADY STATE - CASE 3'
               CASE ELSE
# Do nothing
            END SELECT
            CALL SOLVIT
#
            I = I + 1
         ENDWHILE
#
      ENDIF
#
# Start firing.  Decomposition chamber raised up to 1000 C.
# We assume instantaneous temperature raising for node 2 only.
# The rest is at steady state from previous case.
#
      HEADER = 'TRANSIENT'
      Switch = 2                # Switch for SS/TR
      T:Thruster1:2 = 1000.0D0
      T:Thruster2:2 = 800.0D0
      CALL STATST('N:Thruster1:2', 'D')   # Put back to diffusion
      CALL STATST('N:Thruster2:2', 'D')   #
      CALL SLFWBK
#
$VARIABLES1
      IF (TIMEM .GT. 100.) THEN
         QI:Thruster2:14 = 5.0D0
         QI:Thruster2:15 = 1.50D0
      ENDIF
#
$OUTPUTS
#
# Selection for output routines
#
      SELECT CASE Switch
      CASE 1
        TFORM = '(1X,I6,1X,A24,1X,F8.2,1X,F8.1)'
        THEAD = ' NODE * NODE LABEL              * TEMP. (C) * POWER *'
        CALL PRNDTB (' ', 'L,T,QI' ,Thruster1)
      CASE 2
        TFORM = '(1X,I6,1X,A24,1X,F8.2,1X,F8.1)'
        THEAD = ' NODE * NODE LABEL              * TEMP. (C) * POWER *'
        CALL PRNDTB (' ', 'L,T,QI' ,CURRENT)
      CASE ELSE
        WRITE (6,100)
        STOP
      END SELECT
#
  100 FORMAT ('@@@@ FATAL ERROR in OUTPUTS CALL')
#
$ENDMODEL Thruster
```

7.1.2.2 THRUSTER Preprocessor Log File

```
$MODEL Thruster
#
# RCSID: $Id: listings/THRUSTER.log 1.4 2009/01/13 11:22:58GMT hsoft Released  $
#
# This model contains two, nearly identical submodels.
# It has only little physical meaning, but is primarily
# designed to show the basic features of ESATAN.
#
$MODEL Thruster1
#
# This model is derived from the hydrazine thruster model.
#
$NODES
D1 = 'nozzle', 20., 4.6;
D3 = 'thruster ins.', 20., 8.0;
D4 = 'head plate', 20., 3.5;
D5 = 'heat barrier outer', 20., 1.9;
D6 = 'heat barrier middle', 20., 1.9;
D7 = 'heat barrier inner', 20., 1.9;
D8 = 'injection tube outer', 20., 0.176;
D9 = 'injection tube middle', 20., 0.68;
D10 = 'injection tube inner', 20., 0.065;
D11 = 'cartridge heat.ntwrk out', 20., 7.28;
D12 = 'cartridge heat.ntwrk inn', 20., 12.72;
D13 = 'heat barrier interface', 20., 14.6;
D14 = 'valve flange', 20., 10.35;
D15 = 'valve body', 20., 54.83;
D2 = 'decomposition chamber', 200., 25.;
B105 = 'S/C node 105', 20.;
B999 = 'space', -270.16;
#
$CONDUCTORS
GL(  1,  2) = 0.0015    *   INTRP1(TAV(T1, T2), Conductivity, 1);
GL(  2,  3) = 0.0085;
GL(  2,  4) = 0.0053    *   INTRP1(TAV(T2, T4), Conductivity, 1);
GL(  4,  5) = 4.32E-3   *   INTRP1(TAV(T4, T5), Conductivity, 1);
GL(  5,  6) = 8.23E-4   *   INTRP1(TAV(T5, T6), Conductivity, 1);
GL(  6,  7) = 8.23E-4   *   INTRP1(TAV(T6, T7), Conductivity, 1);
GL(  7, 13) = 4.32E-3   *   INTRP1(TAV(T7, T13), Conductivity, 1);
GL( 13, 14) = 7.00E-3   *   INTRP1(TAV(T13, T14), Conductivity, 1);
GL( 14, 15) = 0.15;
GL(  2,  8) = 2.361E-4  *   INTRP1(TAV(T2, T8), Conductivity, 1);
GL(  8,  9) = 5.059E-5  *   INTRP1(TAV(T8, T9), Conductivity, 1);
GL(  9, 10) = 4.086E-5  *   INTRP1(TAV(T9, T10), Conductivity, 1);
GL( 10, 13) = 1.518E-4  *   INTRP1(TAV(T10, T13), Conductivity, 1);
GL(  2, 11) = 0.0041;
GL( 11, 12) = 0.0041;
GL( 12, 15) = 0.0024;
GL( 14,105) = 0.069;
GR(  1, 999) = 2.1324E-4;
GR(  3, 999) = 6.1080E-4;
GR(  4, 999) = 7.2000E-5;
GR(  5, 999) = 3.9000E-5;
GR(  6, 999) = 3.9000E-5;
GR(  8, 999) = 6.0000E-6;
GR(  9, 999) = 2.3200E-5;
GR( 11, 999) = 2.0600E-4;
$CONSTANTS
$CONTROL
RELXCA = 0.001;
#@@@
```

```
#@@@  WARNING:     Global control constant RELXCA             discarded
#@@@  Trace:(SUSCN ,  16)
#@@@
NLOOP  = 1000;
#@@@
#@@@  WARNING:     Global control constant NLOOP              discarded
#@@@  Trace:(SUSCN ,  16)
#@@@
$ARRAYS
REAL*Conductivity (2,2) =
   0.0, 9.5 ,
   400., 16.6;
#
$SUBROUTINES
      DOUBLE PRECISION FUNCTION TAV(XX, YY)
      DOUBLE PRECISION XX, YY
      TAV = (XX + YY) / 2.0D0
      RETURN
      END
$INITIAL
#
# Simulate catalyst bed heating
#
      CALL STATST('N2', 'B')
#
$ENDMODEL Thruster1

#@@> Submodel: Thruster1                This  Included
#@@>                                  Submodel Submodels
#@@> Thermal Nodes                         17        17
#@@> GL Conductors                         17        17
#@@> GR Conductors                          8         8
#@@> Real Arrays & Events                   1         1

#
$MODEL Thruster2
$REPEAT Thruster1
#
# The Thruster1 model is copied.
# Small changes are effectively made in the main model $INITIAL block,
# and additional control logic included in the main model $VARIABLES1.
#
#
# Now starts the overall model, which is trivial.  Only node
# 999 (space) is made identical and 1 coupling added.
# In fact, the two thruster models run nearly independently.
#
$NODES
B999 = 'space' = Thruster1:999 + Thruster2:999,T = -270.16;
#
$CONDUCTORS
GR(Thruster1:13, Thruster2:13) = 0.000005;   # A VERY SMALL VALUE
#
$CONSTANTS
$CONTROL
RELXCA = 0.001;
NLOOP  = 1000;
TIMEND = 1000.;
OUTINT = 500.;
DTIMEI = 10.;
$INTEGER
Switch = 1;
#
$INITIAL
      T:Thruster2:105 = 0.0
```

```
#
$EXECUTION
      INTEGER I
#
# Do 3 steady states in a WHILE loop, if switch=1.
# The temperature of node 999 is varied.
#
      IF (Switch .EQ. 1) THEN
#
         I = 1
         WHILE (I .LE. 3)
            SELECT CASE I
                CASE 1
                   HEADER = 'STEADY STATE - CASE 1'
#                  no action taken
                CASE 2
                   T999 = 0.0D0
                   QI:Thruster1:5 = 2.50D0
                   QI:Thruster1:9 = 2.50D0
                   HEADER = 'STEADY STATE - CASE 2'
                CASE 3
                   T999 = 100.0D0
                   QI:Thruster1:5 = 1.50D0
                   QI:Thruster1:9 = 1.50D0
                   HEADER = 'STEADY STATE - CASE 3'
                CASE ELSE
# Do nothing
            END SELECT
            CALL SOLVIT
#
            I = I + 1
         ENDWHILE
#
      ENDIF
#
# Start firing.  Decomposition chamber raised up to 1000 C.
# We assume instantaneous temperature raising for node 2 only.
# The rest is at steady state from previous case.
#
      HEADER = 'TRANSIENT'
      Switch = 2                # Switch for SS/TR
      T:Thruster1:2 = 1000.0D0
      T:Thruster2:2 = 800.0D0
      CALL STATST('N:Thruster1:2', 'D')   # Put back to diffusion
      CALL STATST('N:Thruster2:2', 'D')   #
      CALL SLFWBK
#
$VARIABLES1
      IF (TIMEM .GT. 100.) THEN
         QI:Thruster2:14 = 5.0D0
         QI:Thruster2:15 = 1.50D0
      ENDIF
#
$OUTPUTS
#
# Selection for output routines
#
      SELECT CASE Switch
      CASE 1
        TFORM = '(1X,I6,1X,A24,1X,F8.2,1X,F8.1)'
        THEAD = ' NODE * NODE LABEL              * TEMP. (C) * POWER *'
        CALL PRNDTB (' ', 'L,T,QI' ,Thruster1)
      CASE 2
        TFORM = '(1X,I6,1X,A24,1X,F8.2,1X,F8.1)'
        THEAD = ' NODE * NODE LABEL              * TEMP. (C) * POWER *'
```

```
        CALL PRNDTB (' ', 'L,T,QI' ,CURRENT)
      CASE ELSE
        WRITE (6,100)
        STOP
      END SELECT
#
  100 FORMAT ('@@@@ FATAL ERROR in OUTPUTS CALL')
#
$ENDMODEL Thruster

#@@> Submodel: THRUSTER                 This  Included
#@@>                                Submodel Submodels
#@@> Thermal Nodes                         1        35
#@@> Thermal Supernodes                    1         1
#@@> GL Conductors                         0        34
#@@> GR Conductors                         0        16
#@@> Inter-Model GR Conductors             1         1
#@@> Integer User Constants                1         1
#@@> Real Arrays & Events                  0         2

#@@> Hierarchy of Models Including Virtuals(V).
#@@>
#@@>      Level    Model Name
#@@>          0    THRUSTER
#@@>          1       THRUSTER1
#@@>          1       THRUSTER2
```

7.1.2.3 THRUSTER Output Listing

```
EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.7.1)  PAGE     1
21 JULY 2006                    17:51:51                                 THRUSTER

STEADY STATE - CASE 1




 TIMEN =      0.00      MODULE SOLVIT      LOOPCT =    19
ENBALA = 1.036E-05      ENBALR = 2.E-06


TABLE OUTPUT WITH ZENTS = 'L,T,QI'
FOR NODES OF ZLABEL = ' '

================================================================================

THRUSTER:THRUSTER1


  NODE * NODE LABEL              * TEMP. (C) * POWER *

         1 nozzle                    174.72      0.0
         2 decomposition chamber     200.00      0.0
         3 thruster ins.             111.15      0.0
         4 head plate                187.15      0.0
         5 heat barrier outer        174.44      0.0
         6 heat barrier middle       113.04      0.0
         7 heat barrier inner         50.95      0.0
         8 injection tube outer      179.64      0.0
         9 injection tube middle     101.23      0.0
        10 injection tube inner       52.26      0.0
        11 cartridge heat.ntwrk out  109.72      0.0
        12 cartridge heat.ntwrk inn   80.63      0.0
        13 heat barrier interface     38.36      0.0
        14 valve flange               30.15      0.0
        15 valve body                 30.94      0.0
       105 S/C node 105               20.00      0.0
       999 space                    -270.16      0.0



EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.7.1)  PAGE     2
21 JULY 2006                    17:51:51                                 THRUSTER

STEADY STATE - CASE 2




 TIMEN =      0.00      MODULE SOLVIT      LOOPCT =    20
ENBALA = 5.601E-06      ENBALR = 6.E-07


TABLE OUTPUT WITH ZENTS = 'L,T,QI'
FOR NODES OF ZLABEL = ' '

================================================================================
```

```
THRUSTER:THRUSTER1


  NODE * NODE LABEL              * TEMP. (C) * POWER *

       1 nozzle                      177.60      0.0
       2 decomposition chamber       200.00      0.0
       3 thruster ins.               122.68      0.0
       4 head plate                  217.86      0.0
       5 heat barrier outer          242.82      2.5
       6 heat barrier middle         160.07      0.0
       7 heat barrier inner           73.45      0.0
       8 injection tube outer        311.71      0.0
       9 injection tube middle       797.83      2.5
      10 injection tube inner        253.95      0.0
      11 cartridge heat.ntwrk out    119.13      0.0
      12 cartridge heat.ntwrk inn     89.85      0.0
      13 heat barrier interface       55.51      0.0
      14 valve flange                 39.03      0.0
      15 valve body                   39.83      0.0
     105 S/C node 105                 20.00      0.0
     999 space                         0.00      0.0


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.7.1)  PAGE     3
21 JULY 2006                    17:51:51                                 THRUSTER

STEADY STATE - CASE 3




 TIMEN =      0.00      MODULE SOLVIT      LOOPCT =    16
ENBALA = 3.792E-05      ENBALR = 6.E-06


TABLE OUTPUT WITH ZENTS = 'L,T,QI'
FOR NODES OF ZLABEL = ' '

================================================================================

THRUSTER:THRUSTER1


  NODE * NODE LABEL              * TEMP. (C) * POWER *

       1 nozzle                      184.68      0.0
       2 decomposition chamber       200.00      0.0
       3 thruster ins.               149.27      0.0
       4 head plate                  207.38      0.0
       5 heat barrier outer          218.73      1.5
       6 heat barrier middle         146.04      0.0
       7 heat barrier inner           67.50      0.0
       8 injection tube outer        284.61      0.0
       9 injection tube middle       644.23      1.5
      10 injection tube inner        213.67      0.0
      11 cartridge heat.ntwrk out    137.51      0.0
      12 cartridge heat.ntwrk inn    100.80      0.0
      13 heat barrier interface       51.35      0.0
      14 valve flange                 37.07      0.0
      15 valve body                   38.08      0.0
     105 S/C node 105                 20.00      0.0
     999 space                       100.00      0.0
```

```
EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.7.1)  PAGE     4
21 JULY 2006                    17:51:51                                THRUSTER

TRANSIENT



 TIMEN =      0.00      MODULE SLFWBK      DTIMEU =   10.0000
CSGMIN =   23.8669  AT NODE 4 IN SUB-MODEL  THRUSTER:THRUSTER1


TABLE OUTPUT WITH ZENTS = 'L,T,QI'
FOR NODES OF ZLABEL = ' '

================================================================================

THRUSTER


  NODE * NODE LABEL              * TEMP. (C) * POWER *

       999 space                      100.00      0.0


================================================================================

THRUSTER:THRUSTER1


  NODE * NODE LABEL              * TEMP. (C) * POWER *

         1 nozzle                     184.68      0.0
         2 decomposition chamber     1000.00      0.0
         3 thruster ins.              149.27      0.0
         4 head plate                 207.38      0.0
         5 heat barrier outer         218.73      1.5
         6 heat barrier middle        146.04      0.0
         7 heat barrier inner          67.50      0.0
         8 injection tube outer       284.61      0.0
         9 injection tube middle      644.23      1.5
        10 injection tube inner       213.67      0.0
        11 cartridge heat.ntwrk out   137.51      0.0
        12 cartridge heat.ntwrk inn   100.80      0.0
        13 heat barrier interface      51.35      0.0
        14 valve flange                37.07      0.0
        15 valve body                  38.08      0.0
       105 S/C node 105                20.00      0.0
       999 space                      100.00      0.0


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.7.1)  PAGE     5
21 JULY 2006                    17:51:51                                THRUSTER

TRANSIENT


THRUSTER:THRUSTER2
```

```
  NODE * NODE LABEL              * TEMP. (C) * POWER *

       1 nozzle                      184.68      0.0
       2 decomposition chamber       800.00      0.0
       3 thruster ins.               149.27      0.0
       4 head plate                  188.29      0.0
       5 heat barrier outer          175.61      0.0
       6 heat barrier middle         109.92      0.0
       7 heat barrier inner           37.31      0.0
       8 injection tube outer        184.26      0.0
       9 injection tube middle       119.93      0.0
      10 injection tube inner         44.46      0.0
      11 cartridge heat.ntwrk out    133.31      0.0
      12 cartridge heat.ntwrk inn     89.15      0.0
      13 heat barrier interface       22.40      0.0
      14 valve flange                 12.49      0.0
      15 valve body                   13.70      0.0
     105 S/C node 105                  0.00      0.0
     999 space                       100.00      0.0


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.7.1)  PAGE     6
21 JULY 2006                    17:51:51                                THRUSTER

TRANSIENT




 TIMEN =    500.00      MODULE SLFWBK      DTIMEU =   10.0000
CSGMIN =   21.3781  AT NODE 4 IN SUB-MODEL  THRUSTER:THRUSTER1


TABLE OUTPUT WITH ZENTS = 'L,T,QI'
FOR NODES OF ZLABEL = ' '

================================================================================

THRUSTER


  NODE * NODE LABEL              * TEMP. (C) * POWER *

     999 space                       100.00      0.0


================================================================================

THRUSTER:THRUSTER1


  NODE * NODE LABEL              * TEMP. (C) * POWER *

       1 nozzle                      471.55      0.0
       2 decomposition chamber       574.85      0.0
       3 thruster ins.               277.80      0.0
       4 head plate                  550.35      0.0
       5 heat barrier outer          530.68      1.5
       6 heat barrier middle         345.47      0.0
       7 heat barrier inner          126.00      0.0
       8 injection tube outer        570.23      0.0
       9 injection tube middle       691.26      1.5
```

```
       10 injection tube inner       239.30      0.0
       11 cartridge heat.ntwrk out   229.98      0.0
       12 cartridge heat.ntwrk inn   109.73      0.0
       13 heat barrier interface      75.47      0.0
       14 valve flange                44.06      0.0
       15 valve body                  40.72      0.0
      105 S/C node 105                20.00      0.0
      999 space                      100.00      0.0


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.7.1)  PAGE     7
21 JULY 2006                    17:51:51                                THRUSTER

TRANSIENT


THRUSTER:THRUSTER2


  NODE * NODE LABEL              * TEMP. (C) * POWER *

       1 nozzle                     406.67      0.0
       2 decomposition chamber      469.98      0.0
       3 thruster ins.              247.06      0.0
       4 head plate                 446.34      0.0
       5 heat barrier outer         419.62      0.0
       6 heat barrier middle        275.96      0.0
       7 heat barrier inner          99.08      0.0
       8 injection tube outer       421.23      0.0
       9 injection tube middle      220.06      0.0
      10 injection tube inner        96.02      0.0
      11 cartridge heat.ntwrk out   202.07      0.0
      12 cartridge heat.ntwrk inn    96.61      0.0
      13 heat barrier interface      61.29      0.0
      14 valve flange                50.49      5.0
      15 valve body                  38.70      1.5
     105 S/C node 105                 0.00      0.0
     999 space                      100.00      0.0


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.7.1)  PAGE     8
21 JULY 2006                    17:51:51                                THRUSTER

TRANSIENT




 TIMEN =   1000.00      MODULE SLFWBK      DTIMEU =   10.0000
CSGMIN =   21.5319  AT NODE 4 IN SUB-MODEL  THRUSTER:THRUSTER1


TABLE OUTPUT WITH ZENTS = 'L,T,QI'
FOR NODES OF ZLABEL = ' '

================================================================================

THRUSTER


  NODE * NODE LABEL              * TEMP. (C) * POWER *
```

999	space	100.00	0.0

==========

THRUSTER:THRUSTER1

NODE	* NODE LABEL	* TEMP. (C)	* POWER *
1	nozzle	388.15	0.0
2	decomposition chamber	446.73	0.0
3	thruster ins.	256.88	0.0
4	head plate	436.60	0.0
5	heat barrier outer	429.15	1.5
6	heat barrier middle	288.87	0.0
7	heat barrier inner	117.80	0.0
8	injection tube outer	472.18	0.0
9	injection tube middle	677.05	1.5
10	injection tube inner	238.32	0.0
11	cartridge heat.ntwrk out	234.31	0.0
12	cartridge heat.ntwrk inn	122.08	0.0
13	heat barrier interface	78.83	0.0
14	valve flange	48.27	0.0
15	valve body	46.35	0.0
105	S/C node 105	20.00	0.0
999	space	100.00	0.0

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.7.1) PAGE 9
21 JULY 2006 17:51:51 THRUSTER

TRANSIENT

THRUSTER:THRUSTER2

NODE	* NODE LABEL	* TEMP. (C)	* POWER *
1	nozzle	331.95	0.0
2	decomposition chamber	366.23	0.0
3	thruster ins.	230.06	0.0
4	head plate	351.53	0.0
5	heat barrier outer	333.49	0.0
6	heat barrier middle	231.41	0.0
7	heat barrier inner	108.62	0.0
8	injection tube outer	338.45	0.0
9	injection tube middle	220.84	0.0
10	injection tube inner	114.11	0.0
11	cartridge heat.ntwrk out	204.85	0.0
12	cartridge heat.ntwrk inn	108.53	0.0
13	heat barrier interface	82.72	0.0
14	valve flange	70.38	5.0
15	valve body	65.23	1.5
105	S/C node 105	0.00	0.0
999	space	100.00	0.0

7.1.2.4 THRUSTER Generated Fortran Listing

```
 PROGRAM THRUST
C
      INCLUDE 'THRUSTER.h'
C
      CHARACTER MNAME * 24
C
      MNAME = 'THRUSTER                '
C
      FLG(1) = 35
      FLG(2) = 34
      FLG(3) = 0
      FLG(4) = 17
      FLG(5) = 0
      FLG(6) = 0
      FLG(7) = 1
      FLG(8) = 0
      FLG(9) = 0
      FLG(10) = 0
      FLG(11) = 2
      FLG(12) = 0
      FLG(13) = 49
      FLG(14) = 25
      FLG(15) = 16
      FLG(16) = 1
      FLG(17) = 92
      FLG(18) = 22
      FLG(19) = 2
      FLG(20) = 8
      FLG(21) = 0
      FLG(22) = 0
      FLG(23) = 0
      FLG(24) = 444
      FLG(25) = 6
      FLG(26) = 3
      FLG(27) = 133
      FLG(28) = 10540
      FLG(29) = 1
      FLG(30) = 0
      FLG(31) = 0
      FLG(32) = 0
      FLG(33) = 0
      FLG(34) = 0
      FLG(35) = 30
      FLG(36) = 1
      FLG(37) = 0
      FLG(38) = 0
      FLG(39) = 68
      FLG(40) = 0
C
      SPRNDM = 1
      SPRNDN = 3
      USRNDC = 0
      USRNDI = 0
      USRNDR = 0
C
      CALL MAINA(MNAME)
C
      CALL SUCCES
C
      STOP
C
      END
      DOUBLE PRECISION FUNCTION DF0201(XX,YY)
```

```
 INCLUDE 'THRUSTER.h'
 DOUBLEPRECISION XX,YY
 DF0201=(XX+YY)/2.0D0
 RETURN
 END
 SUBROUTINE IN0002
 INCLUDE 'THRUSTER.h'
 LOGICAL HTFLAG
 HTFLAG = .TRUE.
 IG(2) = 2
 CALL STATST('N2','B')
 CALL FINITS(' ' , -    2)
 RETURN
 END
 SUBROUTINE V10002
 INCLUDE 'THRUSTER.h'
 LOGICAL HTFLAG
 HTFLAG = (SOLTYP .EQ. 'THERMAL' .OR. SOLTYP .EQ. 'HUMID')
 IG(2) = 2
 CALL GM0002(HTFLAG)
 RETURN
 END
 SUBROUTINE GM0002_00001(HTFLAG)
 INCLUDE 'THRUSTER.h'
 LOGICAL HTFLAG
 GL(MD(4,2)+1)=0.0015*INTRP1(DF0201(T(MD(3,2)+1),T(MD(3,2)+2)),MD(1
$4,2)+1,1)
 GL(MD(4,2)+3)=0.0053*INTRP1(DF0201(T(MD(3,2)+2),T(MD(3,2)+4)),MD(1
$4,2)+1,1)
 GL(MD(4,2)+6)=4.32E-3*INTRP1(DF0201(T(MD(3,2)+4),T(MD(3,2)+5)),MD(
$14,2)+1,1)
 GL(MD(4,2)+7)=8.23E-4*INTRP1(DF0201(T(MD(3,2)+5),T(MD(3,2)+6)),MD(
$14,2)+1,1)
 GL(MD(4,2)+8)=8.23E-4*INTRP1(DF0201(T(MD(3,2)+6),T(MD(3,2)+7)),MD(
$14,2)+1,1)
 GL(MD(4,2)+9)=4.32E-3*INTRP1(DF0201(T(MD(3,2)+7),T(MD(3,2)+13)),MD
$(14,2)+1,1)
 GL(MD(4,2)+15)=7.00E-3*INTRP1(DF0201(T(MD(3,2)+13),T(MD(3,2)+14)),
$MD(14,2)+1,1)
 GL(MD(4,2)+4)=2.361E-4*INTRP1(DF0201(T(MD(3,2)+2),T(MD(3,2)+8)),MD
$(14,2)+1,1)
 GL(MD(4,2)+10)=5.059E-5*INTRP1(DF0201(T(MD(3,2)+8),T(MD(3,2)+9)),M
$D(14,2)+1,1)
 GL(MD(4,2)+11)=4.086E-5*INTRP1(DF0201(T(MD(3,2)+9),T(MD(3,2)+10)),
$MD(14,2)+1,1)
 GL(MD(4,2)+12)=1.518E-4*INTRP1(DF0201(T(MD(3,2)+10),T(MD(3,2)+13))
$,MD(14,2)+1,1)
 RETURN
 END
 SUBROUTINE GM0002(HTFLAG)
 LOGICAL HTFLAG
 CALL GM0002_00001(HTFLAG)
 RETURN
 END
 SUBROUTINE V20002
 INCLUDE 'THRUSTER.h'
 IG(2) = 2
 RETURN
 END
 DOUBLE PRECISION FUNCTION DF0301(XX,YY)
 INCLUDE 'THRUSTER.h'
 DOUBLEPRECISION XX,YY
 DF0301=(XX+YY)/2.0D0
 RETURN
 END
```

```
 SUBROUTINE IN0003
 INCLUDE 'THRUSTER.h'
 LOGICAL HTFLAG
 HTFLAG = .TRUE.
 IG(2) = 3
 CALL STATST('N2','B')
 CALL FINITS(' ' , -    3)
 RETURN
 END
 SUBROUTINE V10003
 INCLUDE 'THRUSTER.h'
 LOGICAL HTFLAG
 HTFLAG = (SOLTYP .EQ. 'THERMAL' .OR. SOLTYP .EQ. 'HUMID')
 IG(2) = 3
 CALL GM0003(HTFLAG)
 RETURN
 END
 SUBROUTINE GM0003_00001(HTFLAG)
 INCLUDE 'THRUSTER.h'
 LOGICAL HTFLAG
 GL(MD(4,3)+1)=0.0015*INTRP1(DF0301(T(MD(3,3)+1),T(MD(3,3)+2)),MD(1
$4,3)+1,1)
 GL(MD(4,3)+3)=0.0053*INTRP1(DF0301(T(MD(3,3)+2),T(MD(3,3)+4)),MD(1
$4,3)+1,1)
 GL(MD(4,3)+6)=4.32E-3*INTRP1(DF0301(T(MD(3,3)+4),T(MD(3,3)+5)),MD(
$14,3)+1,1)
 GL(MD(4,3)+7)=8.23E-4*INTRP1(DF0301(T(MD(3,3)+5),T(MD(3,3)+6)),MD(
$14,3)+1,1)
 GL(MD(4,3)+8)=8.23E-4*INTRP1(DF0301(T(MD(3,3)+6),T(MD(3,3)+7)),MD(
$14,3)+1,1)
 GL(MD(4,3)+9)=4.32E-3*INTRP1(DF0301(T(MD(3,3)+7),T(MD(3,3)+13)),MD
$(14,3)+1,1)
 GL(MD(4,3)+15)=7.00E-3*INTRP1(DF0301(T(MD(3,3)+13),T(MD(3,3)+14)),
$MD(14,3)+1,1)
 GL(MD(4,3)+4)=2.361E-4*INTRP1(DF0301(T(MD(3,3)+2),T(MD(3,3)+8)),MD
$(14,3)+1,1)
 GL(MD(4,3)+10)=5.059E-5*INTRP1(DF0301(T(MD(3,3)+8),T(MD(3,3)+9)),M
$D(14,3)+1,1)
 GL(MD(4,3)+11)=4.086E-5*INTRP1(DF0301(T(MD(3,3)+9),T(MD(3,3)+10)),
$MD(14,3)+1,1)
 GL(MD(4,3)+12)=1.518E-4*INTRP1(DF0301(T(MD(3,3)+10),T(MD(3,3)+13))
$,MD(14,3)+1,1)
 RETURN
 END
 SUBROUTINE GM0003(HTFLAG)
 LOGICAL HTFLAG
 CALL GM0003_00001(HTFLAG)
 RETURN
 END
 SUBROUTINE V20003
 INCLUDE 'THRUSTER.h'
 IG(2) = 3
 RETURN
 END
 SUBROUTINE IN0001
 INCLUDE 'THRUSTER.h'
 LOGICAL HTFLAG
 HTFLAG = .TRUE.
 OPBLOK = 'INITIAL   '
 CALL IN0002
 CALL IN0003
 IG(2) = 1
 T(MD(3,3)+16)=0.0
 CALL FINITS(' ' , -    1)
 OPBLOK = ' '
```

```
      RETURN
      END
      SUBROUTINE V10001
      INCLUDE 'THRUSTER.h'
      LOGICAL HTFLAG
      HTFLAG = (SOLTYP .EQ. 'THERMAL' .OR. SOLTYP .EQ. 'HUMID')
      OPBLOK = 'VARIABLES1'
      IF(SOLVER(:2) .EQ. 'SS' .AND. IG(24) .GT. 0)THEN
         IF(MOD(IG(1),IG(24)) .EQ. 0 .AND. IG(1) .GT. 0)CALL PROGRS(0)
         IF(MOD(IG(9),IG(24)) .EQ. 0 .AND. IG(9) .GT. 0)CALL PROGRS(0)
         IF(MOD(IG(10),IG(24)) .EQ. 0 .AND. IG(10) .GT. 0)CALL PROGRS(0)
      END IF
      CALL V10002
      CALL V10003
      IG(2) = 1
      IF(RG(16).GT.100.)THEN
      QI(MD(3,3)+14)=5.0D0
      QI(MD(3,3)+15)=1.50D0
      ENDIF
      CALL GM0001(HTFLAG)
      OPBLOK = ' '
      RETURN
      END
      SUBROUTINE GM0001(HTFLAG)
      LOGICAL HTFLAG
      RETURN
      END
      SUBROUTINE V20001
      INCLUDE 'THRUSTER.h'
      OPBLOK = 'VARIABLES2'
      CALL SSNCNT(FLG(24),FLG(25),MAX0(FLG(1),1),PCS,T)
      IG(25) = IG(25) + 1
      IF(SOLVER(:2) .EQ. 'TR' .AND. IG(24) .GT. 0)THEN
         IF(MOD(IG(25),IG(24)) .EQ. 0 .AND. IG(25) .GT. 0)CALL PROGRS(0)
      END IF
      CALL V20002
      CALL V20003
      IG(2) = 1
      OPBLOK = ' '
      CALL PARWRT('VARIABLES2')
      RETURN
      END
      SUBROUTINE EXECTN
      INCLUDE 'THRUSTER.h'
      INTEGER I
      IF(IU(MD(9,1)+1).EQ.1)THEN
      I=1
      LW0001=0
    1 CONTINUE
      LW0001=LW0001+1
      IF(LW0001.GT.IQQMAX)THEN
      CALL SERROR('EXECUTION','THRUSTER                ','LW0001')
      ELSE IF(I.LE.3)THEN
      QQ0001=I
      IF(QQ0001.EQ.1)THEN
      HEADER='STEADY STATE - CASE 1'
      ELSE IF(QQ0001.EQ.2)THEN
      T(MD(3,1)+1)=0.0D0
      QI(MD(3,2)+5)=2.50D0
      QI(MD(3,2)+9)=2.50D0
      HEADER='STEADY STATE - CASE 2'
      ELSE IF(QQ0001.EQ.3)THEN
      T(MD(3,1)+1)=100.0D0
      QI(MD(3,2)+5)=1.50D0
      QI(MD(3,2)+9)=1.50D0
```

```
      HEADER='STEADY STATE - CASE 3'
      ELSE
      END IF
      CALL SOLVIT
      I=I+1
      ELSE
      GO TO    2
      END IF
      GO TO    1
    2 CONTINUE
      ENDIF
      HEADER='TRANSIENT'
      IU(MD(9,1)+1)=2
      T(MD(3,2)+2)=1000.0D0
      T(MD(3,3)+2)=800.0D0
      CALL STATST('N:Thruster1:2','D')
      CALL STATST('N:Thruster2:2','D')
      CALL SLFWBK
      RETURN
      END
      SUBROUTINE OUTPUT
      INCLUDE 'THRUSTER.h'
      IF (OUTIME .NE. 'ALL') RETURN
      OPBLOK = 'OUTPUTS'
      QQ0001=IU(MD(9,1)+1)
      IF(QQ0001.EQ.1)THEN
      TFORM='(1X,I6,1X,A24,1X,F8.2,1X,F8.1)'
      THEAD=' NODE * NODE LABEL              * TEMP. (C) * POWER *'
      CALL PRNDTB(' ','L,T,QI',2)
      ELSE IF(QQ0001.EQ.2)THEN
      TFORM='(1X,I6,1X,A24,1X,F8.2,1X,F8.1)'
      THEAD=' NODE * NODE LABEL              * TEMP. (C) * POWER *'
      CALL PRNDTB(' ','L,T,QI',1)
      ELSE
      WRITE(6,100)
      STOP
      END IF
  100 FORMAT('@@@@ FATAL ERROR in OUTPUTS CALL')
      OPBLOK = ' '
      CALL PARWRT('OUTPUTS')
      RETURN
      END
```

7.2 Fluid Loop Example

7.2.1 Discussion

The fluid loop topology is described by the $NODES and $CONDUCTORS blocks, but the linking of the conventional pipe network to any standard elements has to be defined. Most of these items, once defined, generate a direct submodel of the current model. Joining to the pipework is therefore a matter of defining an inter-model link between the pipework and the element.

Most elements have one input and one output, these are then connected into the main loop by intermodel conductances. The following example illustrates this point.

The model described below has the following features. A coldplate (COLDPLATE1) is connected to a panel radiator (DSPRADIATOR) via an accumulator and pump. The coldplate is initially set to receive heat sources of 15W per node, on each of its 32 nodes, by the statements shown in the $INITIAL block of the main model. All of the heat so gained is rejected to deep space by the radiator panel. A bypass line around the coldplate contains a flow control valve(modelled by a GP conductance between nodes 5 and 6). The tee-piece models TEE1 and TEE2 feature tee-piece loss correlations formulated by $VARIABLES1 statements and also functions provided in the $SUBROUTINES block.

A steady state solution is performed by a call to subroutine FLTNSS in the $EXECUTION block. There then follows an immediate rise in the coldplate heat sources, and a request for a transient solution by subroutine FLTNTS. The heat sources on the coldplate are reset back to their original values after 5 seconds of the transient, by suitable Mortran statements in $VARIABLES2.

During the transient, an active control strategy is followed by adjusting the flow control valve—GP(5, 6)—in response to the coldplate fluid exit temperature (node 8). This is achieved by the function shown in $VARIABLES1, which adjusts the valve GP value as a function of the error between T8 and the setpoint temperature. The latter is chosen to be 30°C. Both the valve gain and the setpoint are assigned values through user constants.

The following listings are provided:-

1) Input data

2) Solution output

An apparent inconsistency in the results may be noted, whereby a pressure *rise* is observed in the direction of flow between nodes 2 and 3. This is because part of the flow is diverted into the side branch, resulting in a drop in mass flow rate, and hence velocity, in the main branch. This results in a pressure recovery: the values output are *static* pressure, as opposed to total or stagnation pressure.

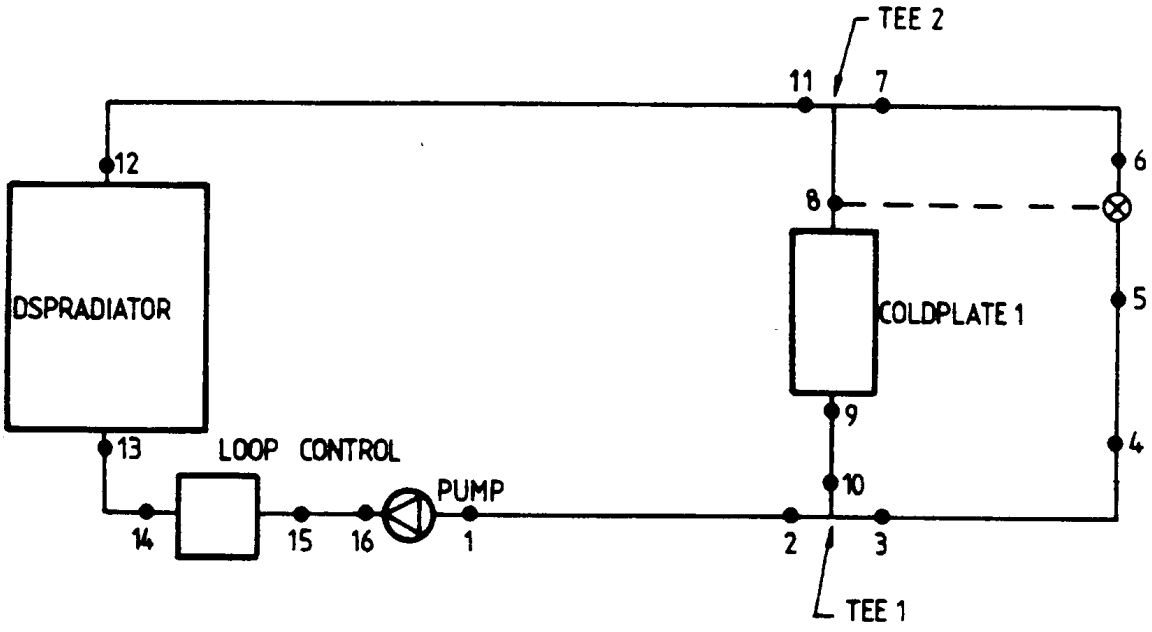


Figure 7-1 Example Fluid Loop

7.2.1.1 Fluid Loop Input File

```
$MODEL UMLOOP, FLUID = WATER

$Id: listings/UMLOOP.d 1.2 2005/11/30 16:09:06GMT jstrutt Released  $

      EXAMPLE OF FLUID LOOP INPUT
      MODEL FEATURES ELEMENTS:
                              COLDPLATE
                              PANEL RADIATOR
                              TEES
                              PUMP
                              ACCUMULATOR

      OTHER FEATURES ARE TEE PIECE LOSS CORRELATIONS,
      AND ACTIVE CONTROL FOR A VALVE.

      JOHN TURNER     NOVEMBER 1988

$MODEL PUMP, FLUID = WATER
      $ELEMENT PUMP_CF
      $SUBSTITUTIONS
      CHAR_TYPE = 'POLY';
      DIAM = 0.010;
      TEMP = 20.0;
      PRESS = 0.0;
      VOL = 1.571E-5;
      SPEED = 4000.0;   # Fixed-speed pump only
      DP_ARRAY = 5.0E4, -1.0E7, -1.0E6;
      EFF_ARRAY = 0.9, 0.0001;
      MFLOW = 0.0374;
#
#
$ENDMODEL PUMP
#
$MODEL COLDPLATE1, FLUID = WATER
      $ELEMENT COLDPLATE
      $SUBSTITUTIONS
#
      LENX = 0.5; LENY = 0.3; DZ = 0.010;
      NX = 4; NY = 4; EK = 360.; ERHO = 2.71E3;
      ECP = .896E3; ETP = 20.0;
      EFD = 0.005; EP = 1.010;
      EFE = 2.1E3; ET = 20.0; EFF = 1.E-7;
      NU = 2; MFLOW = .0183;
$ENDMODEL COLDPLATE1
#
$MODEL DSPRADIATOR, FLUID = WATER
#
      $ELEMENT PANEL
      $SUBSTITUTIONS
      LENX = 0.5; LENY = 1.0; DZ = 0.010;
      NX = 4; NY = 4; RTP = 20.0; RK = 220.0;
      RRHO = 5.6E03; RCP = 1.9E03; RFD = 0.005;
      RP = 0.0; RFE = 2100.0; RT = 20.0; RFF = 1.E-7;
      NU = 2; MFLOW = .0374; REMM1 = 0.8; REMM2 = %REMM1%;
      RVFAC1 = 1.0; RVFAC2 = 0.45; RT1 = -273.0;
      RT2 = %RT1%;
#
$ENDMODEL DSPRADIATOR
#
$MODEL LOOPCONTROL, FLUID = WATER
#
      $ELEMENT ACCP
      $SUBSTITUTIONS
```

```
      AA = 0.005; AFD = 0.005; AFL = 0.1;
      AFF = 1.E-7; AP = 1.E5; AT=20.0;
      AFE = 2.0E3;
$ENDMODEL LOOPCONTROL
#
$MODEL TEE1, FLUID = WATER
      $ELEMENT TEE
      $SUBSTITUTIONS
      TA1 = 0.09; TFD1 = 0.010; TFL1 = 0.10; TP1 = 0.0;
      TFE1 = 2150.0; TT1 = 20.0; TFF1 = 1.E-7;
      TA2 = %TA1%; TFD2 = %TFD1%;
      TFL2 = %TFL1%;
      TP2 = %TP1%;
      TFE2 = %TFE1%; TT2 = %TT1%; TFF2 = %TFF1%;
      TGP = 1.E10 ;
      MFLOW = 0.0183;
$ENDMODEL TEE1
#
$MODEL TEE2, FLUID = WATER
      $REPEAT TEE1
$ENDMODEL TEE2
#
#     ELEMENT DESCRIPTIONS FINISHED, NOW THE MAIN
#     MODEL, WHICH DESCRIBES THE LOOP PIPEWORK
#
$NODES
      F1 = 'PUMP EXIT', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F2 = 'BEFORE BRANCH', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F3 = 'AFTER BRANCH', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F4 = 'BYPASS', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F5 = 'VALVE INLET', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F6 = 'VALVE EXIT', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F7 = 'BEFORE MERGE', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F8 = 'COLDPLATE1 EXIT', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F9 = 'COLDPLATE1 INLET', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F10= 'TEE SIDE EXIT', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F11= 'AFTER MERGE', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F12= 'RADIATOR INLET', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F13= 'RADIATOR EXIT', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F14= 'BEFORE ACCUMULATOR', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F15= 'AFTER ACCUMULATOR', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
      F16 = 'PUMP INLET', A = 0.0015, FD = 0.010, FL = 0.10,
      P = 0.0, FE = 2000.0, T = 20.0, FF = 1.E-7;
#
$CONDUCTORS
#
#     MASS FLOW LINKS GIVEN FIRST, TO DEFINE LOOP
#     TOPOLOGY
#
      M (1, 2) = 0.0374;
```

```
      M (2, TEE1 : 1) = 0.0374;
      M (TEE1 : 1, 3) = 0.0191;
      M (3, 4) = 0.0191;
      M (4, 5) = 0.0191;
      M (5, 6) = 0.0191;
      M (6, 7) = 0.0191;
      M (7, TEE2 : 1) = 0.0191;
      M (TEE1 : 2, 10) = 0.0183;
      M (10, 9) = 0.0183;
      M (9, COLDPLATE1 : 1017) = 0.0183;
      M (COLDPLATE 1 : 2024, 8) = 0.0183;
      M (8, TEE2 : 2) = 0.0183;
      M (TEE2 : 1, 11) = 0.0374;
      M (11, 12) = 0.0374;
      M (12, DSPRADIATOR : 1017) = 0.0374;
      M (DSPRADIATOR : 2024, 13) = 0.0374;
      M (13, 14) = 0.0374;
      M (14, LOOPCONTROL : 1) = 0.0374;
      M (LOOPCONTROL : 1, 15) = 0.0374;
      M (15, 16) = 0.0374;
      M (16, PUMP : 1) = 0.0374;
      M (PUMP : 2, 1) = 0.0374;
#
#     GP LINKS ARE ONLY DEFINED WHERE NECESSARY
#
      GP (3, 4) = 2.0;  # BEND LOSS
      GP (6, 7) = 2.0;  #ANOTHER BEND LOSS
#
      GP (11, 12) = 2.2;
      GP (13, 14) = 2.1;
#
      GP (5, 6) = 0.0225;  # VALVE AT NOMINAL SETTING
#
$CONSTANTS
$REAL
      temp_set_point = 30.0 ;   #  TEMPERATURE SET POINT
      KF = U ;                  #  VALVE COEFFICIENT
      XOPEN = U ;               #  VALVE OPENING FRACTION
$CONTROL
      DAMPT = 0.5;
      NLOOP = 100;
      FRLXCA=0.005;
      RELXCA = 0.005;
      RELXMA = 0.005;
      OUTINT = 5.0;
      TIMEND = 10.;
      DTIMEI = 0.5;
      PABS = 1.01325D5;  # GAUGE PRESSURES
      GRAVZ = 9.81;      # TERRESTRIAL
$ARRAYS
#
#  ARRAYS FOR IDEL'CIK TEE-PIECE LOSS CORRELATIONS
#  TAKEN FROM 'SPACECRAFT THERMAL CONTROL DESIGN DATA',
#  ESA(TST - 02) , VOL. 4.
#
$REAL
      Array_1(2,6) = 0.0, 0.0,  0.2, 0.26,  0.4, 0.44,
                     0.6, 0.54, 0.8, 0.58,  1.0, 0.5;
      Array_2(2,6) = 0.0,-0.6,  0.2,-0.18,  0.4, 0.2,
                     0.6, 0.58, 0.8, 0.92,  1.0,1.2;
      Array_3(2,6) = 0.0, 0.085,  0.2, 0.07,  0.4, 0.055,
                     0.6, 0.05,   0.8, 0.07,  1.0,0.1;
      Array_4(2,6) = 0.0, 0.0,  0.2, 0.085,  0.4, 0.15,
                     0.6,0.25,  0.8,0.33,  1.0,0.4;
$SUBROUTINES
```

```
#
##############################################
#
      DOUBLE PRECISION FUNCTION CONLOS(M1,M2)
C
C IDEL'CIK CORRELATION FOR TEE CONVERGENCE
C
      DOUBLE PRECISION K12,MRATIO,M1,M2
C
C  RATIO FOR LOSS FACTOR
C
      MRATIO = ABS(M2 / M1)
C
C  LOSS FACTOR BY INTERPOLATION
C
      K12=INTRP1(MRATIO,Array_1,1)
C
C  CONLOS
C
      CONLOS=(1/K12)*((1 - M2/M1)**2)
C
      RETURN
C
      END
#
##############################################
#
      DOUBLE PRECISION FUNCTION CONTGP(M1,M2)
C
C TEE CONVERGENCE CORRELATION
C
      DOUBLE PRECISION K23,MRATIO,M1,M2
C
C  RATIO FOR LOSS FACTOR
C
      MRATIO = ABS(M2 / M1)
C
C  LOSS FACTOR BY INTERPOLATION
C
      K23=INTRP1(MRATIO,Array_2,1)
C
C  CONVERGENCE TGP
C
      CONTGP=(1/K23)*((M2/M1)**2)
C
      RETURN
C
      END
#
##############################################
#
      DOUBLE PRECISION FUNCTION DIVTGP(M1,M2)
C
C TEE DIVERGENCE
C
      DOUBLE PRECISION K13,MRATIO,M1,M2
C
C  RATIO FOR LOSS FACTOR
C
      MRATIO = ABS(M2 / M1)
C
C  LOSS FACTOR BY INTERPOLATION
C
      K13=INTRP1(MRATIO,Array_3,1)
C
```

```
C  DIVERGENCE TGP
C
      DIVTGP=(1/K13)*((M2/M1)**2)
C
      RETURN
C
      END
#
##############################################
#
      DOUBLE PRECISION FUNCTION DIVLOS(M1,M2)
C
C  TEE DIVERGENCE
C
      DOUBLE PRECISION K12,MRATIO,M1,M2
C
C  RATIO FOR LOSS FACTOR
C
      MRATIO = ABS(M2 / M1)
C
C  LOSS FACTOR BY INTERPOLATION
C
      K12=INTRP1(MRATIO,Array_4,1)
C
C  DIVERGENCE LOSS
C
      DIVLOS=(1/K12)*((1 - M2/M1)**2)
C
      RETURN
C
      END
#
##############################################
#
$INITIAL
      INTEGER ICOUNT
#
# SET COLDPLATE HEAT SOURCES
#
      ICOUNT = 0
      REPEAT
*
      QR : COLDPLATE1 : 1001(ICOUNT) = 15.0D0
      QR : COLDPLATE1 : 2001(ICOUNT) = 15.0D0
*
      ICOUNT = ICOUNT + 1
*
      UNTIL(ICOUNT .EQ. 16)
#
#
#
 $VARIABLES1
      IF(MODULE .EQ. 'FLTNTS')THEN
C
C     SIMPLE CONTROLLER USING ERROR FUNCTION FOR
C     COLDPLATE EXIT TEMPERATURE.
C     ADJUSTS CONTROL VALVE BETWEEN NODES 5 AND 6
C     TO MODULATE FLOW THROUGH COLDPLATE.
C
#   FIRST CALCULATE VALVE OPENING FRACTION FROM ERROR
#   WITH RESPECT TO SET POINT
#
         XOPEN = (temp_set_point - T8)/temp_set_point + 0.5
C
         IF(XOPEN .GT. 1.0)THEN
```

```
            XOPEN = 1.0
C
         ELSE IF(XOPEN .LT. 0.0)THEN
            XOPEN = 0.0
C
         ELSE
            CONTINUE
C
         END IF
#
#   CALCULATE VALVE COEFFICIENT FROM CHARACTERISTIC
#
         KF = XOPEN * 0.3
#
#   ASSIGN FLUID CONDUCTANCE FOR VALVE
#
          GP(5 , 6) = KF ** 2
#
      ELSE

         CONTINUE
#
      END IF
C
C THE FOLLOWING INVOKES THE IDEL'CIK TEE-PIECE LOSS
C CORRELATIONS GIVEN AS FUNCTIONS IN THE SUBROUTINES
C BLOCK
C
      GP:TEE2:(1,2) = CONTGP(M(TEE2:1,11),M:TEE2:(1,2))
      GP(7,TEE2:1) = CONLOS(M(TEE2:1,11),M:TEE2:(1,2))
      GP:TEE1:(1,2) = DIVTGP(M(2,TEE1:1),M:TEE1:(1,2))
      GP(TEE1:1,3) = DIVLOS(M(2,TEE1:1),M:TEE1:(1,2))
$VARIABLES2
C
C RESET COLDPLATE HEAT SOURCES BACK TO ORIGINAL
C VALUES AFTER 5 SECONDS
C
      INTEGER ICOUNT
      IF(TIMEN .GE. 5.0)THEN
        ICOUNT = 0
        REPEAT
*
        QR : COLDPLATE1 : 1001(ICOUNT) = 15.0
        QR : COLDPLATE1 : 2001(ICOUNT) = 15.0
*
        ICOUNT = ICOUNT + 1
*
        UNTIL(ICOUNT .EQ. 16)
      ELSE
C
        CONTINUE
      END IF
C
$OUTPUTS
      FORMAT='F10.5'
      CALL PRNDTB(' ','L,T,P,QR',CURRENT)
      CALL PRNDBL(' ','M',CURRENT)
$EXECUTION DYSTOR = 12000
      INTEGER ICOUNT
#
      CALL FLTNSS
C
C  INCREASE COLDPLATE HEAT SOURCES
C
      ICOUNT = 0
```

```
#
      REPEAT
#
      QR : COLDPLATE1 : 1001(ICOUNT) = 500.0
      QR : COLDPLATE1 : 2001(ICOUNT) = 500.0
#
      ICOUNT = ICOUNT + 1
#
      UNTIL(ICOUNT .EQ. 16)
C
      CALL FLTNTS
C
$ENDMODEL UMLOOP
```

7.2.1.2 Fluid Loop Output Listing

```
EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE     1
30 NOVEMBER 2005                16:05:37                                  UMLOOP




 TIMEN =      0.00      MODULE FLTNSS      LOOPCT =     3
ENBALA =   14.3110      ENBALR = 0.0112    DTCOUR =     0.0000


TABLE OUTPUT WITH ZENTS = 'L,T,P,QR'
FOR NODES OF ZLABEL = ' '

================================================================================

UMLOOP

         NODE   LABEL                              T             P


           1    PUMP EXIT                        18.29     150950.11
           2    BEFORE BRANCH                    18.29     150894.03
           3    AFTER BRANCH                     18.28     150953.89
           4    BYPASS                           18.28     150892.84
           5    VALVE INLET                      18.28     150866.23
           6    VALVE EXIT                       18.28     147777.75
           7    BEFORE MERGE                     18.28     147716.70
           8    COLDPLATE1 EXIT                  26.50     147661.91
           9    COLDPLATE1 INLET                 18.28     150780.39
          10    TEE SIDE EXIT                    18.28     150786.27
          11    AFTER MERGE                      20.91     147458.79
          12    RADIATOR INLET                   20.91     147335.74
          13    RADIATOR EXIT                    18.30     103138.52
          14    BEFORE ACCUMULATOR               18.29     103011.26
          15    AFTER ACCUMULATOR                18.29     101472.58
          16    PUMP INLET                       18.29     101416.50


         NODE          QR


           1
           2
           3
           4
           5
           6
           7
           8
           9
          10
          11
          12
          13
          14
          15
          16
```

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 2
30 NOVEMBER 2005 16:05:37 UMLOOP

UMLOOP:PUMP

NODE	LABEL	T	P
1		18.29	101388.46
2		18.30	150978.15

NODE	QR
1	
2	

==========

UMLOOP:COLDPLATE1

NODE	LABEL	T	P
1001		25.44	
1002		28.56	
1003		29.23	
1004		27.33	
1005		25.15	
1006		26.95	
1007		27.67	
1008		26.96	
1009		26.69	
1010		25.64	
1011		26.25	
1012		28.23	
1013		28.35	
1014		26.03	
1015		26.55	
1016		29.83	
2001		27.97	
2002		31.26	
2003		32.05	
2004		30.18	
2005		27.69	
2006		29.67	
2007		30.53	
2008		29.78	
2009		29.09	
2010		28.44	
2011		29.14	
2012		30.80	
2013		30.63	
2014		28.81	
2015		29.39	
2016		32.27	

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 3
30 NOVEMBER 2005 16:05:37 UMLOOP

NODE	LABEL	T	P
1017		18.95	150429.02
1018		19.38	150241.06
1019		19.81	150099.43
1020		20.43	149909.46
1021		21.07	149670.71
1022		21.50	149478.82
1023		21.91	149334.24
1024		22.51	149140.35
2017		23.14	148896.71
2018		23.57	148700.90
2019		23.97	148553.37
2020		24.56	148355.54
2021		25.18	148106.99
2022		25.58	147907.27
2023		25.97	147757.14
2024		26.51	147557.45

NODE	QR
1001	15.00
1002	15.00
1003	15.00
1004	15.00
1005	15.00
1006	15.00
1007	15.00
1008	15.00
1009	15.00
1010	15.00
1011	15.00
1012	15.00
1013	15.00
1014	15.00
1015	15.00
1016	15.00
2001	15.00
2002	15.00
2003	15.00
2004	15.00
2005	15.00
2006	15.00
2007	15.00
2008	15.00
2009	15.00
2010	15.00
2011	15.00
2012	15.00
2013	15.00
2014	15.00

NODE	QR
2015	15.00
2016	15.00
1017	
1018	
1019	
1020	
1021	
1022	
1023	
1024	
2017	
2018	
2019	
2020	
2021	
2022	
2023	
2024	

==========

UMLOOP:DSPRADIATOR

NODE	LABEL	T	P
1001		18.14	
1002		15.54	
1003		15.19	
1004		17.15	
1005		19.91	
1006		19.48	
1007		18.88	
1008		18.97	
1009		19.41	
1010		19.61	
1011		19.28	
1012		18.82	
1013		15.46	
1014		17.71	
1015		17.53	
1016		14.94	
1025		-273.00	
1026		-273.00	
2001		16.91	
2002		14.35	
2003		14.03	
2004		15.98	
2005		18.64	
2006		18.18	

NODE	LABEL	T	P

2007	17.61	
2008	17.76	
2009	18.23	
2010	18.33	
2011	18.02	
2012	17.67	
2013	14.59	
2014	16.53	
2015	16.36	
2016	14.08	
2025	-273.00	
2026	-273.00	
1017	20.73	144152.87
1018	20.55	141413.66
1019	20.37	137757.87
1020	20.21	135013.56
1021	20.04	133181.87
1022	19.87	130430.84
1023	19.71	126759.59
1024	19.55	124003.93
2017	19.39	122164.76
2018	19.22	119402.62
2019	19.06	115716.61
2020	18.91	112949.87
2021	18.75	111103.28
2022	18.60	108329.99
2023	18.44	104629.13
2024	18.30	101851.23

NODE	QR
1001	0.00
1002	0.00
1003	0.00
1004	0.00
1005	0.00
1006	0.00
1007	0.00
1008	0.00
1009	0.00
1010	0.00
1011	0.00
1012	0.00
1013	0.00
1014	0.00
1015	0.00
1016	0.00
1025	0.00
1026	0.00

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 6
30 NOVEMBER 2005 16:05:37 UMLOOP

NODE	QR
2001	0.00
2002	0.00
2003	0.00

```
        2004          0.00
        2005          0.00
        2006          0.00
        2007          0.00
        2008          0.00
        2009          0.00
        2010          0.00
        2011          0.00
        2012          0.00
        2013          0.00
        2014          0.00
        2015          0.00
        2016          0.00
        2025          0.00
        2026          0.00

        1017
        1018
        1019
        1020
        1021
        1022
        1023
        1024
        2017
        2018
        2019
        2020
        2021
        2022
        2023
        2024
```

==========

```
UMLOOP:LOOPCONTROL

         NODE   LABEL                                T             P

           1                                     18.29     100000.00
```

```
EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE     7
30 NOVEMBER 2005               16:05:37                                  UMLOOP
```

```
         NODE          QR

           1
```

==========

```
UMLOOP:TEE1

         NODE   LABEL                                T             P
```

```
           1                                    18.29     150837.94
           2                                    18.29     150792.15


         NODE          QR


           1
           2

================================================================================

UMLOOP:TEE2


         NODE   LABEL                              T             P


           1                                    20.91     147675.10
           2                                    26.50     147656.95


         NODE          QR


           1
           2


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE     8
30 NOVEMBER 2005                16:05:37                                  UMLOOP




 TIMEN =      0.00      MODULE FLTNSS      LOOPCT =     3
ENBALA =   14.3110      ENBALR = 0.0112    DTCOUR =     0.0000


BLOCK OUTPUT WITH ZENTS = 'M'
FOR NODES OF ZLABEL = ' '


UMLOOP

VALUES FOR CONDUCTORS M  :
M (1,2)         =    0.04290      M (1,Z2:2)      =   -0.04290
M (2,1)         =   -0.04290      M (2,Z6:1)      =    0.04290
M (3,4)         =    0.02913      M (3,Z6:1)      =   -0.02913
M (4,3)         =   -0.02913      M (4,5)         =    0.02913
M (5,4)         =   -0.02913      M (5,6)         =    0.02913
M (6,5)         =   -0.02913      M (6,7)         =    0.02913
M (7,6)         =   -0.02913      M (7,Z7:1)      =    0.02913
M (8,Z3:2024)   =   -0.01378      M (8,Z7:2)      =    0.01378
M (9,10)        =   -0.01378      M (9,Z3:1017)   =    0.01378
M (10,9)        =    0.01378      M (10,Z6:2)     =   -0.01378
M (11,12)       =    0.04290      M (11,Z7:1)     =   -0.04290
M (12,11)       =   -0.04290      M (12,Z4:1017)  =    0.04290
M (13,14)       =    0.04290      M (13,Z4:2024)  =   -0.04290
M (14,13)       =   -0.04290      M (14,Z5:1)     =    0.04290
M (15,16)       =    0.04290      M (15,Z5:1)     =   -0.04290
M (16,15)       =   -0.04290      M (16,Z2:1)     =    0.04290
```

```
UMLOOP:PUMP

VALUES FOR CONDUCTORS M  :
M (1,2)     =    0.04290      M (1,Z1:16) =   -0.04290
M (2,1)     =   -0.04290      M (2,Z1:1)  =    0.04290


UMLOOP:COLDPLATE1

VALUES FOR CONDUCTORS M  :
M (1017,1018) =    0.01378      M (1017,Z1:9) =   -0.01378
M (1018,1017) =   -0.01378      M (1018,1019) =    0.01378
M (1019,1018) =   -0.01378      M (1019,1020) =    0.01378
M (1020,1019) =   -0.01378      M (1020,1021) =    0.01378
M (1021,1020) =   -0.01378      M (1021,1022) =    0.01378
M (1022,1021) =   -0.01378      M (1022,1023) =    0.01378
M (1023,1022) =   -0.01378      M (1023,1024) =    0.01378
M (1024,1023) =   -0.01378      M (1024,2017) =    0.01378
M (2017,1024) =   -0.01378      M (2017,2018) =    0.01378
M (2018,2017) =   -0.01378      M (2018,2019) =    0.01378
M (2019,2018) =   -0.01378      M (2019,2020) =    0.01378
M (2020,2019) =   -0.01378      M (2020,2021) =    0.01378


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE     9
30 NOVEMBER 2005                16:05:37                                UMLOOP



M (2021,2020) =   -0.01378      M (2021,2022) =    0.01378
M (2022,2021) =   -0.01378      M (2022,2023) =    0.01378
M (2023,2022) =   -0.01378      M (2023,2024) =    0.01378
M (2024,2023) =   -0.01378      M (2024,Z1:8) =    0.01378


UMLOOP:DSPRADIATOR

VALUES FOR CONDUCTORS M  :
M (1017,1018)  =    0.04290      M (1017,Z1:12) =   -0.04290
M (1018,1017)  =   -0.04290      M (1018,1019)  =    0.04290
M (1019,1018)  =   -0.04290      M (1019,1020)  =    0.04290
M (1020,1019)  =   -0.04290      M (1020,1021)  =    0.04290
M (1021,1020)  =   -0.04290      M (1021,1022)  =    0.04290
M (1022,1021)  =   -0.04290      M (1022,1023)  =    0.04290
M (1023,1022)  =   -0.04290      M (1023,1024)  =    0.04290
M (1024,1023)  =   -0.04290      M (1024,2017)  =    0.04290
M (2017,1024)  =   -0.04290      M (2017,2018)  =    0.04290
M (2018,2017)  =   -0.04290      M (2018,2019)  =    0.04290
M (2019,2018)  =   -0.04290      M (2019,2020)  =    0.04290
M (2020,2019)  =   -0.04290      M (2020,2021)  =    0.04290
M (2021,2020)  =   -0.04290      M (2021,2022)  =    0.04290
M (2022,2021)  =   -0.04290      M (2022,2023)  =    0.04290
M (2023,2022)  =   -0.04290      M (2023,2024)  =    0.04290
M (2024,2023)  =   -0.04290      M (2024,Z1:13) =    0.04290


UMLOOP:LOOPCONTROL

VALUES FOR CONDUCTORS M  :
M (1,Z1:14) =   -0.04290      M (1,Z1:15) =    0.04290


UMLOOP:TEE1
```

```
VALUES FOR CONDUCTORS M  :
M (1,2)     =    0.01378      M (1,Z1:2)  =   -0.04290
M (1,Z1:3)  =    0.02913      M (2,1)     =   -0.01378
M (2,Z1:10) =    0.01378


UMLOOP:TEE2

VALUES FOR CONDUCTORS M  :
M (1,2)     =   -0.01377      M (1,Z1:7)  =   -0.02913
M (1,Z1:11) =    0.04290      M (2,1)     =    0.01377
M (2,Z1:8)  =   -0.01378


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    10
30 NOVEMBER 2005                16:05:37                                  UMLOOP



 KEY FOR SUB-MODEL CODE :


Z1 = UMLOOP


Z2 = UMLOOP:PUMP


Z3 = UMLOOP:COLDPLATE1


Z4 = UMLOOP:DSPRADIATOR


Z5 = UMLOOP:LOOPCONTROL


Z6 = UMLOOP:TEE1


Z7 = UMLOOP:TEE2


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    11
30 NOVEMBER 2005                16:05:37                                  UMLOOP




 TIMEN =      0.00      MODULE FLTNTS      RELXMC = 3.434E-03
DTIMEU =    0.5000      DTCOUR = 4.527E-02
CSGMIN = 4.569E-02  AT NODE 1 IN SUB-MODEL  UMLOOP:LOOPCONTROL


TABLE OUTPUT WITH ZENTS = 'L,T,P,QR'
FOR NODES OF ZLABEL = ' '

================================================================================

UMLOOP


         NODE   LABEL                              T             P
```

1	PUMP EXIT	18.29	150973.49
2	BEFORE BRANCH	18.29	150916.52
3	AFTER BRANCH	18.28	150960.09
4	BYPASS	18.28	150890.30
5	VALVE INLET	18.28	150859.47
6	VALVE EXIT	18.28	148551.37
7	BEFORE MERGE	18.28	148481.58
8	COLDPLATE1 EXIT	26.50	148415.89
9	COLDPLATE1 INLET	18.28	150809.48
10	TEE SIDE EXIT	18.28	150814.74
11	AFTER MERGE	20.91	148229.48
12	RADIATOR INLET	20.91	148104.30
13	RADIATOR EXIT	18.30	103194.70
14	BEFORE ACCUMULATOR	18.29	103065.24
15	AFTER ACCUMULATOR	18.29	101501.79
16	PUMP INLET	18.29	101444.82

NODE	QR
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	

UMLOOP:PUMP

NODE	LABEL	T	P
1		18.29	101416.33
2		18.30	151001.98

NODE	QR
1	
2	

==========

UMLOOP:COLDPLATE1

NODE	LABEL	T	P
1001		25.44	
1002		28.56	
1003		29.23	
1004		27.33	
1005		25.15	
1006		26.95	
1007		27.67	
1008		26.96	
1009		26.69	
1010		25.64	
1011		26.25	
1012		28.23	
1013		28.35	
1014		26.03	
1015		26.55	
1016		29.83	
2001		27.97	
2002		31.26	
2003		32.05	
2004		30.18	
2005		27.69	
2006		29.67	
2007		30.53	
2008		29.78	
2009		29.09	
2010		28.44	
2011		29.14	
2012		30.80	
2013		30.63	
2014		28.81	
2015		29.39	
2016		32.27	

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 13
30 NOVEMBER 2005 16:05:37 UMLOOP

NODE	LABEL	T	P
1017		18.95	150531.80
1018		19.38	150387.42
1019		19.81	150278.66
1020		20.43	150132.84
1021		21.07	149949.63
1022		21.50	149802.43
1023		21.91	149691.55
1024		22.51	149542.91
2017		23.14	149356.19
2018		23.57	149206.18
2019		23.97	149093.18
2020		24.56	148941.71
2021		25.18	148751.46
2022		25.58	148598.64
2023		25.97	148483.54
2024		26.51	148329.31

NODE	QR
1001	500.00
1002	500.00
1003	500.00
1004	500.00
1005	500.00
1006	500.00
1007	500.00
1008	500.00
1009	500.00
1010	500.00
1011	500.00
1012	500.00
1013	500.00
1014	500.00
1015	500.00
1016	500.00
2001	500.00
2002	500.00
2003	500.00
2004	500.00
2005	500.00
2006	500.00
2007	500.00
2008	500.00
2009	500.00
2010	500.00
2011	500.00
2012	500.00
2013	500.00
2014	500.00

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 14
30 NOVEMBER 2005 16:05:37 UMLOOP

NODE	QR
2015	500.00
2016	500.00
1017	
1018	
1019	
1020	
1021	
1022	
1023	
1024	
2017	
2018	
2019	
2020	
2021	
2022	
2023	
2024	

==========

UMLOOP:DSPRADIATOR

NODE	LABEL	T	P
1001		18.14	
1002		15.54	
1003		15.19	
1004		17.15	
1005		19.91	
1006		19.48	
1007		18.88	
1008		18.97	
1009		19.41	
1010		19.61	
1011		19.28	
1012		18.82	
1013		15.46	
1014		17.71	
1015		17.53	
1016		14.94	
1025		-273.00	
1026		-273.00	
2001		16.91	
2002		14.35	
2003		14.03	
2004		15.98	
2005		18.64	
2006		18.18	

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 15
30 NOVEMBER 2005 16:05:37 UMLOOP

NODE	LABEL	T	P
2007		17.61	
2008		17.76	
2009		18.23	
2010		18.33	
2011		18.02	
2012		17.67	
2013		14.59	
2014		16.53	
2015		16.36	
2016		14.08	
2025		-273.00	
2026		-273.00	
1017		20.73	144864.67
1018		20.55	142081.31
1019		20.37	138366.58
1020		20.21	135578.03
1021		20.04	133716.81
1022		19.87	130921.43
1023		19.71	127190.99
1024		19.55	124390.89
2017		19.39	122522.08
2018		19.22	119715.41
2019		19.06	115969.98

2020	18.91	113158.65
2021	18.75	111282.30
2022	18.60	108464.33
2023	18.44	104703.84
2024	18.30	101881.19

NODE	QR
1001	0.00
1002	0.00
1003	0.00
1004	0.00
1005	0.00
1006	0.00
1007	0.00
1008	0.00
1009	0.00
1010	0.00
1011	0.00
1012	0.00
1013	0.00
1014	0.00
1015	0.00
1016	0.00
1025	0.00
1026	0.00

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 16
30 NOVEMBER 2005 16:05:37 UMLOOP

NODE	QR
2001	0.00
2002	0.00
2003	0.00
2004	0.00
2005	0.00
2006	0.00
2007	0.00
2008	0.00
2009	0.00
2010	0.00
2011	0.00
2012	0.00
2013	0.00
2014	0.00
2015	0.00
2016	0.00
2025	0.00
2026	0.00

1017
1018
1019
1020
1021
1022
1023
1024

```
        2017
        2018
        2019
        2020
        2021
        2022
        2023
        2024

================================================================================

UMLOOP:LOOPCONTROL

         NODE   LABEL                                T             P

           1                                      18.29     100000.00


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    17
30 NOVEMBER 2005               16:05:37                                  UMLOOP



         NODE          QR

           1

================================================================================

UMLOOP:TEE1

         NODE   LABEL                                T             P

           1                                      18.29     150859.55
           2                                      18.29     150819.99

         NODE          QR

           1
           2

================================================================================

UMLOOP:TEE2

         NODE   LABEL                                T             P

           1                                      20.91     148434.42
           2                                      26.50     148411.56

         NODE          QR
```

```
          1
          2


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    18
30 NOVEMBER 2005               16:05:37                                   UMLOOP




 TIMEN =      0.00      MODULE FLTNTS      RELXMC = 3.434E-03
DTIMEU =    0.5000      DTCOUR = 4.527E-02
CSGMIN = 4.569E-02  AT NODE 1 IN SUB-MODEL  UMLOOP:LOOPCONTROL


BLOCK OUTPUT WITH ZENTS = 'M'
FOR NODES OF ZLABEL = ' '


UMLOOP

VALUES FOR CONDUCTORS M  :
M (1,2)        =    0.04330      M (1,Z2:2)     =   -0.04330
M (2,1)        =   -0.04330      M (2,Z6:1)     =    0.04330
M (3,4)        =    0.03098      M (3,Z6:1)     =   -0.03098
M (4,3)        =   -0.03098      M (4,5)        =    0.03098
M (5,4)        =   -0.03098      M (5,6)        =    0.03098
M (6,5)        =   -0.03098      M (6,7)        =    0.03098
M (7,6)        =   -0.03098      M (7,Z7:1)     =    0.03098
M (8,Z3:2024)  =   -0.01232      M (8,Z7:2)     =    0.01232
M (9,10)       =   -0.01232      M (9,Z3:1017)  =    0.01232
M (10,9)       =    0.01232      M (10,Z6:2)    =   -0.01232
M (11,12)      =    0.04330      M (11,Z7:1)    =   -0.04330
M (12,11)      =   -0.04330      M (12,Z4:1017) =    0.04330
M (13,14)      =    0.04330      M (13,Z4:2024) =   -0.04330
M (14,13)      =   -0.04330      M (14,Z5:1)    =    0.04330
M (15,16)      =    0.04330      M (15,Z5:1)    =   -0.04330
M (16,15)      =   -0.04330      M (16,Z2:1)    =    0.04330


UMLOOP:PUMP

VALUES FOR CONDUCTORS M  :
M (1,2)     =    0.04330      M (1,Z1:16) =   -0.04330
M (2,1)     =   -0.04330      M (2,Z1:1)  =    0.04330


UMLOOP:COLDPLATE1

VALUES FOR CONDUCTORS M  :
M (1017,1018) =    0.01232      M (1017,Z1:9) =   -0.01232
M (1018,1017) =   -0.01232      M (1018,1019) =    0.01232
M (1019,1018) =   -0.01232      M (1019,1020) =    0.01232
M (1020,1019) =   -0.01232      M (1020,1021) =    0.01232
M (1021,1020) =   -0.01232      M (1021,1022) =    0.01232
M (1022,1021) =   -0.01232      M (1022,1023) =    0.01232
M (1023,1022) =   -0.01232      M (1023,1024) =    0.01232
M (1024,1023) =   -0.01232      M (1024,2017) =    0.01232
M (2017,1024) =   -0.01232      M (2017,2018) =    0.01232
M (2018,2017) =   -0.01232      M (2018,2019) =    0.01232


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    19
```

```
30 NOVEMBER 2005                16:05:37                                    UMLOOP



M (2019,2018) =   -0.01232      M (2019,2020) =    0.01232
M (2020,2019) =   -0.01232      M (2020,2021) =    0.01232
M (2021,2020) =   -0.01232      M (2021,2022) =    0.01232
M (2022,2021) =   -0.01232      M (2022,2023) =    0.01232
M (2023,2022) =   -0.01232      M (2023,2024) =    0.01232
M (2024,2023) =   -0.01232      M (2024,Z1:8) =    0.01232


UMLOOP:DSPRADIATOR

VALUES FOR CONDUCTORS M  :
M (1017,1018)  =    0.04330      M (1017,Z1:12) =   -0.04330
M (1018,1017)  =   -0.04330      M (1018,1019)  =    0.04330
M (1019,1018)  =   -0.04330      M (1019,1020)  =    0.04330
M (1020,1019)  =   -0.04330      M (1020,1021)  =    0.04330
M (1021,1020)  =   -0.04330      M (1021,1022)  =    0.04330
M (1022,1021)  =   -0.04330      M (1022,1023)  =    0.04330
M (1023,1022)  =   -0.04330      M (1023,1024)  =    0.04330
M (1024,1023)  =   -0.04330      M (1024,2017)  =    0.04330
M (2017,1024)  =   -0.04330      M (2017,2018)  =    0.04330
M (2018,2017)  =   -0.04330      M (2018,2019)  =    0.04330
M (2019,2018)  =   -0.04330      M (2019,2020)  =    0.04330
M (2020,2019)  =   -0.04330      M (2020,2021)  =    0.04330
M (2021,2020)  =   -0.04330      M (2021,2022)  =    0.04330
M (2022,2021)  =   -0.04330      M (2022,2023)  =    0.04330
M (2023,2022)  =   -0.04330      M (2023,2024)  =    0.04330
M (2024,2023)  =   -0.04330      M (2024,Z1:13) =    0.04330


UMLOOP:LOOPCONTROL

VALUES FOR CONDUCTORS M  :
M (1,Z1:14) =   -0.04330      M (1,Z1:15) =    0.04330


UMLOOP:TEE1

VALUES FOR CONDUCTORS M  :
M (1,2)     =    0.01232      M (1,Z1:2)  =   -0.04330
M (1,Z1:3)  =    0.03098      M (2,1)     =   -0.01232
M (2,Z1:10) =    0.01232


UMLOOP:TEE2

VALUES FOR CONDUCTORS M  :
M (1,2)     =   -0.01232      M (1,Z1:7)  =   -0.03098
M (1,Z1:11) =    0.04330      M (2,1)     =    0.01232
M (2,Z1:8)  =   -0.01232


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    20
30 NOVEMBER 2005                16:05:37                                    UMLOOP



 KEY FOR SUB-MODEL CODE :
```

```
Z1 = UMLOOP

Z2 = UMLOOP:PUMP

Z3 = UMLOOP:COLDPLATE1

Z4 = UMLOOP:DSPRADIATOR

Z5 = UMLOOP:LOOPCONTROL

Z6 = UMLOOP:TEE1

Z7 = UMLOOP:TEE2

EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    21
30 NOVEMBER 2005               16:05:37                                  UMLOOP



 TIMEN =      5.00      MODULE FLTNTS      RELXMC = 2.406E-03
DTIMEU =    0.5000      DTCOUR = 4.591E-02
CSGMIN = 4.591E-02  AT NODE 1 IN SUB-MODEL  UMLOOP:LOOPCONTROL

TABLE OUTPUT WITH ZENTS = 'L,T,P,QR'
FOR NODES OF ZLABEL = ' '

================================================================================

UMLOOP

         NODE   LABEL                          T            P

           1    PUMP EXIT                    18.30     150938.20
           2    BEFORE BRANCH                18.30     150882.59
           3    AFTER BRANCH                 18.29     150952.79
           4    BYPASS                       18.29     150896.97
           5    VALVE INLET                  18.29     150872.85
           6    VALVE EXIT                   18.30     147307.74
           7    BEFORE MERGE                 18.30     147251.91
           8    COLDPLATE1 EXIT              32.75     147204.10
           9    COLDPLATE1 INLET             18.30     150763.99
          10    TEE SIDE EXIT                18.30     150770.29
          11    AFTER MERGE                  22.54     146990.34
          12    RADIATOR INLET               22.34     146868.76
          13    RADIATOR EXIT                18.33     103109.01
          14    BEFORE ACCUMULATOR           18.32     102982.92
          15    AFTER ACCUMULATOR            18.31     101457.43
          16    PUMP INLET                   18.30     101401.82


         NODE          QR
```

```
           1
           2
           3
           4
           5
           6
           7
           8
           9
          10
          11
          12
          13
          14
          15
          16


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    22
30 NOVEMBER 2005                16:05:37                                  UMLOOP


UMLOOP:PUMP

         NODE   LABEL                              T             P

           1                                    18.30     101374.01
           2                                    18.30     150966.00

         NODE          QR

           1
           2

================================================================================

UMLOOP:COLDPLATE1

         NODE   LABEL                              T             P

        1001                                    35.85
        1002                                    39.21
        1003                                    39.88
        1004                                    37.75
        1005                                    35.42
        1006                                    37.46
        1007                                    38.18
        1008                                    37.25
        1009                                    37.19
        1010                                    35.91
        1011                                    36.53
        1012                                    38.74
        1013                                    39.00
        1014                                    36.43
        1015                                    36.96
        1016                                    40.48
        2001                                    38.40
        2002                                    41.91
        2003                                    42.70
```

2004	40.63
2005	38.00
2006	40.19
2007	41.06
2008	40.13
2009	39.61
2010	38.78
2011	39.49
2012	41.34
2013	41.28
2014	39.25
2015	39.84
2016	42.91

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 23
30 NOVEMBER 2005 16:05:37 UMLOOP

NODE	LABEL	T	P
1017		20.14	150356.47
1018		21.26	150132.60
1019		22.33	149962.82
1020		23.88	149734.37
1021		25.32	149449.35
1022		26.19	149222.82
1023		26.99	149053.86
1024		28.13	148830.07
2017		29.20	148551.97
2018		29.87	148330.56
2019		30.49	148165.19
2020		31.36	147945.76
2021		32.21	147672.69
2022		32.74	147455.00
2023		33.23	147292.25
2024		33.92	147076.07

NODE	QR
1001	15.00
1002	15.00
1003	15.00
1004	15.00
1005	15.00
1006	15.00
1007	15.00
1008	15.00
1009	15.00
1010	15.00
1011	15.00
1012	15.00
1013	15.00
1014	15.00
1015	15.00
1016	15.00
2001	15.00
2002	15.00
2003	15.00
2004	15.00

2005	15.00
2006	15.00
2007	15.00
2008	15.00
2009	15.00
2010	15.00
2011	15.00
2012	15.00
2013	15.00
2014	15.00

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 24
30 NOVEMBER 2005 16:05:37 UMLOOP

NODE	QR
2015	15.00
2016	15.00
1017	
1018	
1019	
1020	
1021	
1022	
1023	
1024	
2017	
2018	
2019	
2020	
2021	
2022	
2023	
2024	

==========

UMLOOP:DSPRADIATOR

NODE	LABEL	T	P
1001		18.14	
1002		15.54	
1003		15.19	
1004		17.15	
1005		19.92	
1006		19.48	
1007		18.88	
1008		18.97	
1009		19.41	
1010		19.62	
1011		19.28	
1012		18.82	
1013		15.46	
1014		17.71	
1015		17.53	
1016		14.94	
1025		-273.00	

1026	-273.00
2001	16.91
2002	14.35
2003	14.03
2004	15.98
2005	18.64
2006	18.18

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 25
30 NOVEMBER 2005 16:05:37 UMLOOP

NODE	LABEL	T	P
2007		17.61	
2008		17.76	
2009		18.23	
2010		18.33	
2011		18.02	
2012		17.67	
2013		14.59	
2014		16.53	
2015		16.36	
2016		14.08	
2025		-273.00	
2026		-273.00	
1017		21.97	143719.88
1018		21.50	141015.40
1019		21.10	137402.94
1020		20.84	134689.55
1021		20.58	132877.66
1022		20.28	130154.88
1023		20.02	126519.88
1024		19.82	123790.67
2017		19.62	121968.81
2018		19.39	119232.01
2019		19.19	115579.22
2020		19.02	112837.13
2021		18.84	111006.85
2022		18.66	108257.83
2023		18.49	104589.11
2024		18.34	101835.23

NODE	QR
1001	0.00
1002	0.00
1003	0.00
1004	0.00
1005	0.00
1006	0.00
1007	0.00
1008	0.00
1009	0.00
1010	0.00
1011	0.00
1012	0.00
1013	0.00
1014	0.00

```
        1015          0.00
        1016          0.00
        1025          0.00
        1026          0.00


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    26
30 NOVEMBER 2005               16:05:37                                  UMLOOP



         NODE          QR

        2001          0.00
        2002          0.00
        2003          0.00
        2004          0.00
        2005          0.00
        2006          0.00
        2007          0.00
        2008          0.00
        2009          0.00
        2010          0.00
        2011          0.00
        2012          0.00
        2013          0.00
        2014          0.00
        2015          0.00
        2016          0.00
        2025          0.00
        2026          0.00

        1017
        1018
        1019
        1020
        1021
        1022
        1023
        1024
        2017
        2018
        2019
        2020
        2021
        2022
        2023
        2024

================================================================================

UMLOOP:LOOPCONTROL

         NODE   LABEL                              T             P

           1                                    18.31     100000.00


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    27
30 NOVEMBER 2005               16:05:37                                  UMLOOP
```

```
         NODE          QR

           1

================================================================================

UMLOOP:TEE1

         NODE   LABEL                              T             P

           1                                    18.29     150826.98
           2                                    18.29     150776.59

         NODE          QR

           1
           2

================================================================================

UMLOOP:TEE2

         NODE   LABEL                              T             P

           1                                    22.76     147213.53
           2                                    31.55     147198.14

         NODE          QR

           1
           2


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    28
30 NOVEMBER 2005               16:05:37                                  UMLOOP




 TIMEN =      5.00      MODULE FLTNTS      RELXMC = 2.406E-03
DTIMEU =    0.5000      DTCOUR = 4.591E-02
CSGMIN = 4.591E-02  AT NODE 1 IN SUB-MODEL  UMLOOP:LOOPCONTROL


BLOCK OUTPUT WITH ZENTS = 'M'
FOR NODES OF ZLABEL = ' '


UMLOOP
```

```
VALUES FOR CONDUCTORS M  :
M (1,2)         =    0.04270      M (1,Z2:2)      =   -0.04270
M (2,1)         =   -0.04270      M (2,Z6:1)      =    0.04270
M (3,4)         =    0.02795      M (3,Z6:1)      =   -0.02795
M (4,3)         =   -0.02795      M (4,5)         =    0.02795
M (5,4)         =   -0.02795      M (5,6)         =    0.02795
M (6,5)         =   -0.02795      M (6,7)         =    0.02795
M (7,6)         =   -0.02795      M (7,Z7:1)      =    0.02795
M (8,Z3:2024)   =   -0.01475      M (8,Z7:2)      =    0.01475
M (9,10)        =   -0.01475      M (9,Z3:1017)   =    0.01475
M (10,9)        =    0.01475      M (10,Z6:2)     =   -0.01475
M (11,12)       =    0.04270      M (11,Z7:1)     =   -0.04270
M (12,11)       =   -0.04270      M (12,Z4:1017)  =    0.04270
M (13,14)       =    0.04270      M (13,Z4:2024)  =   -0.04270
M (14,13)       =   -0.04270      M (14,Z5:1)     =    0.04270
M (15,16)       =    0.04270      M (15,Z5:1)     =   -0.04270
M (16,15)       =   -0.04270      M (16,Z2:1)     =    0.04270


UMLOOP:PUMP

VALUES FOR CONDUCTORS M  :
M (1,2)      =    0.04270      M (1,Z1:16) =   -0.04270
M (2,1)      =   -0.04270      M (2,Z1:1)  =    0.04270


UMLOOP:COLDPLATE1

VALUES FOR CONDUCTORS M  :
M (1017,1018) =    0.01475      M (1017,Z1:9) =   -0.01475
M (1018,1017) =   -0.01475      M (1018,1019) =    0.01475
M (1019,1018) =   -0.01475      M (1019,1020) =    0.01475
M (1020,1019) =   -0.01475      M (1020,1021) =    0.01475
M (1021,1020) =   -0.01475      M (1021,1022) =    0.01475
M (1022,1021) =   -0.01475      M (1022,1023) =    0.01475
M (1023,1022) =   -0.01475      M (1023,1024) =    0.01475
M (1024,1023) =   -0.01475      M (1024,2017) =    0.01475
M (2017,1024) =   -0.01475      M (2017,2018) =    0.01475
M (2018,2017) =   -0.01475      M (2018,2019) =    0.01475


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    29
30 NOVEMBER 2005                16:05:37                                  UMLOOP




M (2019,2018) =   -0.01475      M (2019,2020) =    0.01475
M (2020,2019) =   -0.01475      M (2020,2021) =    0.01475
M (2021,2020) =   -0.01475      M (2021,2022) =    0.01475
M (2022,2021) =   -0.01475      M (2022,2023) =    0.01475
M (2023,2022) =   -0.01475      M (2023,2024) =    0.01475
M (2024,2023) =   -0.01475      M (2024,Z1:8) =    0.01475


UMLOOP:DSPRADIATOR

VALUES FOR CONDUCTORS M  :
M (1017,1018)  =    0.04270      M (1017,Z1:12) =   -0.04270
M (1018,1017)  =   -0.04270      M (1018,1019)  =    0.04270
M (1019,1018)  =   -0.04270      M (1019,1020)  =    0.04270
M (1020,1019)  =   -0.04270      M (1020,1021)  =    0.04270
M (1021,1020)  =   -0.04270      M (1021,1022)  =    0.04270
M (1022,1021)  =   -0.04270      M (1022,1023)  =    0.04270
M (1023,1022)  =   -0.04270      M (1023,1024)  =    0.04270
```

```
M (1024,1023)  =   -0.04270      M (1024,2017)  =    0.04270
M (2017,1024)  =   -0.04270      M (2017,2018)  =    0.04270
M (2018,2017)  =   -0.04270      M (2018,2019)  =    0.04270
M (2019,2018)  =   -0.04270      M (2019,2020)  =    0.04270
M (2020,2019)  =   -0.04270      M (2020,2021)  =    0.04270
M (2021,2020)  =   -0.04270      M (2021,2022)  =    0.04270
M (2022,2021)  =   -0.04270      M (2022,2023)  =    0.04270
M (2023,2022)  =   -0.04270      M (2023,2024)  =    0.04270
M (2024,2023)  =   -0.04270      M (2024,Z1:13) =    0.04270


UMLOOP:LOOPCONTROL

VALUES FOR CONDUCTORS M  :
M (1,Z1:14) =   -0.04270      M (1,Z1:15) =    0.04270


UMLOOP:TEE1

VALUES FOR CONDUCTORS M  :
M (1,2)     =    0.01475      M (1,Z1:2)  =   -0.04270
M (1,Z1:3)  =    0.02795      M (2,1)     =   -0.01475
M (2,Z1:10) =    0.01475


UMLOOP:TEE2

VALUES FOR CONDUCTORS M  :
M (1,2)     =   -0.01475      M (1,Z1:7)  =   -0.02795
M (1,Z1:11) =    0.04270      M (2,1)     =    0.01475
M (2,Z1:8)  =   -0.01475


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    30
30 NOVEMBER 2005               16:05:37                                UMLOOP



 KEY FOR SUB-MODEL CODE :


Z1 = UMLOOP


Z2 = UMLOOP:PUMP


Z3 = UMLOOP:COLDPLATE1


Z4 = UMLOOP:DSPRADIATOR


Z5 = UMLOOP:LOOPCONTROL


Z6 = UMLOOP:TEE1


Z7 = UMLOOP:TEE2


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    31
30 NOVEMBER 2005               16:05:37                                UMLOOP
```

```
 TIMEN =     10.00      MODULE FLTNTS      RELXMC = 2.724E-04
DTIMEU =    0.5000      DTCOUR = 4.655E-02
CSGMIN = 4.655E-02  AT NODE 1 IN SUB-MODEL  UMLOOP:LOOPCONTROL


TABLE OUTPUT WITH ZENTS = 'L,T,P,QR'
FOR NODES OF ZLABEL = ' '

================================================================================

UMLOOP

         NODE   LABEL                              T             P


           1    PUMP EXIT                       18.70     150902.54
           2    BEFORE BRANCH                   18.68     150848.37
           3    AFTER BRANCH                    18.63     150946.74
           4    BYPASS                          18.60     150905.94
           5    VALVE INLET                     18.56     150888.84
           6    VALVE EXIT                      18.52     145806.57
           7    BEFORE MERGE                    18.49     145765.79
           8    COLDPLATE1 EXIT                 35.29     145740.17
           9    COLDPLATE1 INLET                18.53     150708.23
          10    TEE SIDE EXIT                   18.57     150716.82
          11    AFTER MERGE                     25.66     145492.12
          12    RADIATOR INLET                  25.66     145374.82
          13    RADIATOR EXIT                   18.75     103025.17
          14    BEFORE ACCUMULATOR              18.75     102902.46
          15    AFTER ACCUMULATOR               18.74     101417.09
          16    PUMP INLET                      18.73     101362.94


         NODE          QR


           1
           2
           3
           4
           5
           6
           7
           8
           9
          10
          11
          12
          13
          14
          15
          16


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    32
30 NOVEMBER 2005               16:05:37                                 UMLOOP
```

UMLOOP:PUMP

NODE	LABEL	T	P
1		18.72	101335.86
2		18.72	150929.63

NODE	QR
1	
2	

==========

UMLOOP:COLDPLATE1

NODE	LABEL	T	P
1001		34.63	
1002		39.11	
1003		39.81	
1004		37.10	
1005		33.79	
1006		36.69	
1007		37.66	
1008		36.35	
1009		36.46	
1010		34.56	
1011		35.42	
1012		38.19	
1013		38.90	
1014		35.48	
1015		36.11	
1016		40.40	
2001		37.82	
2002		41.85	
2003		42.66	
2004		40.32	
2005		37.26	
2006		39.81	
2007		40.80	
2008		39.73	
2009		39.25	
2010		38.17	
2011		39.00	
2012		41.07	
2013		41.23	
2014		38.81	
2015		39.44	
2016		42.87	

NODE	LABEL	T	P
1017		20.44	150107.24
1018		21.59	149781.88
1019		22.68	149539.92
1020		24.25	149220.49
1021		25.73	148824.66
1022		26.65	148510.22
1023		27.51	148275.80
1024		28.73	147965.43
2017		29.92	147579.93
2018		30.66	147273.13
2019		31.36	147044.05
2020		32.34	146740.24
2021		33.30	146362.31
2022		33.89	146061.15
2023		34.46	145836.05
2024		35.25	145537.13

NODE	QR
1001	15.00
1002	15.00
1003	15.00
1004	15.00
1005	15.00
1006	15.00
1007	15.00
1008	15.00
1009	15.00
1010	15.00
1011	15.00
1012	15.00
1013	15.00
1014	15.00
1015	15.00
1016	15.00
2001	15.00
2002	15.00
2003	15.00
2004	15.00
2005	15.00
2006	15.00
2007	15.00
2008	15.00
2009	15.00
2010	15.00
2011	15.00
2012	15.00
2013	15.00
2014	15.00

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 34
30 NOVEMBER 2005 16:05:37 UMLOOP

NODE	QR
2015	15.00
2016	15.00

1017
1018
1019
1020
1021
1022
1023
1024
2017
2018
2019
2020
2021
2022
2023
2024

==========

UMLOOP:DSPRADIATOR

NODE	LABEL	T	P
1001		18.20	
1002		15.54	
1003		15.19	
1004		17.17	
1005		20.05	
1006		19.56	
1007		18.91	
1008		19.02	
1009		19.48	
1010		19.71	
1011		19.35	
1012		18.86	
1013		15.46	
1014		17.74	
1015		17.56	
1016		14.94	
1025		-273.00	
1026		-273.00	
2001		16.93	
2002		14.35	
2003		14.03	
2004		15.99	
2005		18.67	
2006		18.20	

NODE	LABEL	T	P
2007		17.62	
2008		17.77	
2009		18.24	
2010		18.35	
2011		18.04	

2012	17.68	
2013	14.59	
2014	16.54	
2015	16.37	
2016	14.08	
2025	-273.00	
2026	-273.00	
1017	25.01	142325.29
1018	24.07	139733.32
1019	23.27	136262.51
1020	22.82	133650.96
1021	22.40	131904.84
1022	21.82	129276.02
1023	21.32	125761.24
1024	21.01	123119.50
2017	20.70	121354.60
2018	20.31	118700.20
2019	19.96	115153.98
2020	19.72	112490.04
2021	19.49	110711.08
2022	19.21	108037.48
2023	18.95	104467.59
2024	18.76	101786.85

NODE	QR
1001	0.00
1002	0.00
1003	0.00
1004	0.00
1005	0.00
1006	0.00
1007	0.00
1008	0.00
1009	0.00
1010	0.00
1011	0.00
1012	0.00
1013	0.00
1014	0.00
1015	0.00
1016	0.00
1025	0.00
1026	0.00

EUROPEAN SPACE AGENCY THERMAL ANALYSIS NETWORK (VERSION 9.6.0) PAGE 36
30 NOVEMBER 2005 16:05:37 UMLOOP

NODE	QR
2001	0.00
2002	0.00
2003	0.00
2004	0.00
2005	0.00
2006	0.00
2007	0.00
2008	0.00

2009	0.00
2010	0.00
2011	0.00
2012	0.00
2013	0.00
2014	0.00
2015	0.00
2016	0.00
2025	0.00
2026	0.00

1017
1018
1019
1020
1021
1022
1023
1024
2017
2018
2019
2020
2021
2022
2023
2024

==========

UMLOOP:LOOPCONTROL

NODE	LABEL	T	P
1		18.74	100000.00

NODE	QR
1	

==========

UMLOOP:TEE1

NODE	LABEL	T	P
1		18.67	150794.18
2		18.62	150725.42

NODE	QR

```
           1
           2

================================================================================

UMLOOP:TEE2

         NODE   LABEL                                T             P

           1                                      25.66     145737.41
           2                                      35.32     145730.41

         NODE          QR

           1
           2

EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    38
30 NOVEMBER 2005                16:05:37                                  UMLOOP



 TIMEN =     10.00      MODULE FLTNTS      RELXMC = 2.724E-04
DTIMEU =    0.5000      DTCOUR = 4.655E-02
CSGMIN = 4.655E-02  AT NODE 1 IN SUB-MODEL  UMLOOP:LOOPCONTROL


BLOCK OUTPUT WITH ZENTS = 'M'
FOR NODES OF ZLABEL = ' '


UMLOOP

VALUES FOR CONDUCTORS M  :
M (1,2)         =    0.04211      M (1,Z2:2)     =   -0.04211
M (2,1)         =   -0.04211      M (2,Z6:1)     =    0.04211
M (3,4)         =    0.02416      M (3,Z6:1)     =   -0.02416
M (4,3)         =   -0.02416      M (4,5)        =    0.02416
M (5,4)         =   -0.02416      M (5,6)        =    0.02416
M (6,5)         =   -0.02416      M (6,7)        =    0.02416
M (7,6)         =   -0.02416      M (7,Z7:1)     =    0.02416
M (8,Z3:2024)   =   -0.01795      M (8,Z7:2)     =    0.01795
M (9,10)        =   -0.01795      M (9,Z3:1017)  =    0.01795
M (10,9)        =    0.01795      M (10,Z6:2)    =   -0.01795
M (11,12)       =    0.04211      M (11,Z7:1)    =   -0.04211
M (12,11)       =   -0.04211      M (12,Z4:1017) =    0.04211
M (13,14)       =    0.04211      M (13,Z4:2024) =   -0.04211
M (14,13)       =   -0.04211      M (14,Z5:1)    =    0.04211
M (15,16)       =    0.04211      M (15,Z5:1)    =   -0.04211
M (16,15)       =   -0.04211      M (16,Z2:1)    =    0.04211


UMLOOP:PUMP

VALUES FOR CONDUCTORS M  :
M (1,2)      =    0.04211      M (1,Z1:16) =   -0.04211
```

```
M (2,1)     =   -0.04211      M (2,Z1:1)  =    0.04211


UMLOOP:COLDPLATE1

VALUES FOR CONDUCTORS M  :
M (1017,1018) =    0.01795      M (1017,Z1:9) =   -0.01795
M (1018,1017) =   -0.01795      M (1018,1019) =    0.01795
M (1019,1018) =   -0.01795      M (1019,1020) =    0.01795
M (1020,1019) =   -0.01795      M (1020,1021) =    0.01795
M (1021,1020) =   -0.01795      M (1021,1022) =    0.01795
M (1022,1021) =   -0.01795      M (1022,1023) =    0.01795
M (1023,1022) =   -0.01795      M (1023,1024) =    0.01795
M (1024,1023) =   -0.01795      M (1024,2017) =    0.01795
M (2017,1024) =   -0.01795      M (2017,2018) =    0.01795
M (2018,2017) =   -0.01795      M (2018,2019) =    0.01795


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    39
30 NOVEMBER 2005               16:05:37                                UMLOOP



M (2019,2018) =   -0.01795      M (2019,2020) =    0.01795
M (2020,2019) =   -0.01795      M (2020,2021) =    0.01795
M (2021,2020) =   -0.01795      M (2021,2022) =    0.01795
M (2022,2021) =   -0.01795      M (2022,2023) =    0.01795
M (2023,2022) =   -0.01795      M (2023,2024) =    0.01795
M (2024,2023) =   -0.01795      M (2024,Z1:8) =    0.01795


UMLOOP:DSPRADIATOR

VALUES FOR CONDUCTORS M  :
M (1017,1018)  =    0.04211      M (1017,Z1:12) =   -0.04211
M (1018,1017)  =   -0.04211      M (1018,1019)  =    0.04211
M (1019,1018)  =   -0.04211      M (1019,1020)  =    0.04211
M (1020,1019)  =   -0.04211      M (1020,1021)  =    0.04211
M (1021,1020)  =   -0.04211      M (1021,1022)  =    0.04211
M (1022,1021)  =   -0.04211      M (1022,1023)  =    0.04211
M (1023,1022)  =   -0.04211      M (1023,1024)  =    0.04211
M (1024,1023)  =   -0.04211      M (1024,2017)  =    0.04211
M (2017,1024)  =   -0.04211      M (2017,2018)  =    0.04211
M (2018,2017)  =   -0.04211      M (2018,2019)  =    0.04211
M (2019,2018)  =   -0.04211      M (2019,2020)  =    0.04211
M (2020,2019)  =   -0.04211      M (2020,2021)  =    0.04211
M (2021,2020)  =   -0.04211      M (2021,2022)  =    0.04211
M (2022,2021)  =   -0.04211      M (2022,2023)  =    0.04211
M (2023,2022)  =   -0.04211      M (2023,2024)  =    0.04211
M (2024,2023)  =   -0.04211      M (2024,Z1:13) =    0.04211


UMLOOP:LOOPCONTROL

VALUES FOR CONDUCTORS M  :
M (1,Z1:14) =   -0.04211      M (1,Z1:15) =    0.04211


UMLOOP:TEE1

VALUES FOR CONDUCTORS M  :
M (1,2)     =    0.01795      M (1,Z1:2)  =   -0.04211
M (1,Z1:3)  =    0.02416      M (2,1)     =   -0.01795
M (2,Z1:10) =    0.01795
```

```
UMLOOP:TEE2

VALUES FOR CONDUCTORS M  :
M (1,2)     =   -0.01795      M (1,Z1:7)  =   -0.02416
M (1,Z1:11) =    0.04211      M (2,1)     =    0.01795
M (2,Z1:8)  =   -0.01795


EUROPEAN  SPACE  AGENCY  THERMAL  ANALYSIS  NETWORK  (VERSION 9.6.0)  PAGE    40
30 NOVEMBER 2005                16:05:37                                  UMLOOP



 KEY FOR SUB-MODEL CODE :


Z1 = UMLOOP


Z2 = UMLOOP:PUMP


Z3 = UMLOOP:COLDPLATE1


Z4 = UMLOOP:DSPRADIATOR


Z5 = UMLOOP:LOOPCONTROL


Z6 = UMLOOP:TEE1


Z7 = UMLOOP:TEE2
```

Appendix A: ESATAN Manual Syntax

Symbol	Meaning
cent	conductor entity - GL, GR, GF, GV, M, GP
cname	concatenated model name
dim	integer array dimension
exp	real, character, or integer valued expression or literal.
i-val	integer-valued expression of literal and/or local constants
l-exp	logical expression
length	integer length of character variable/function
m	conductor sequence number (integer)
mname	model name, alphanumeric + underscore (up to 24 characters) first character alphabetic, case insensitive
n	user defined entity reference number (integer)
nent	node entity:- L, T, C, QA, QE, QI, QR, QS, A, ALP, EPS, QAI, QEI, QSI, QSI, FD, FL, P, FE, FF, FQ, FM, FH, FR, FX, FY, FZ, VQ, FRG FLA, VOL, PHI, FW, CMP, VDT, FST
r-exp	real valued expression of literal constants, local constants, user constants, and/or function references
r-val	real-valued expression of literal and/or local constants
string	literal character string (e.g. 'ABC')
subs	integer subscript
type	data type:- DOUBLE PRECISION, INTEGER, REAL or CHARACTER
uname	user defined name, alphanumeric + underscore, up to eighteen characters, first character alphabetic; case sensitive
val	expression of literal constants and/or local constants.
z-val	character-valued expression of literal and/or local constants

Table A-1: ESATAN Manual Syntax

The following regulations apply to the definitions shown in this manual.

1. CAPITAL letters must appear exactly as shown.
2. *Italic lower case* must be replaced with suitable values.
3. Square brackets [ ] enclose optional items.
4. The pipe (|) symbol separates alternatives. If not enclosed in square brackets, one alternative must be given.
5. **BOLD CAPITAL** typeface denotes the default of a number of alternatives.
6. Ellipsis (...) indicates that repetition of the preceding item is allowed.
7. Other characters ( ) : ; , = $! ? # ' @ & must appear literally as shown.
8. Blanks terminate keywords, parameters, and numbers, and are significant within literal character strings, but otherwise have no effect.

Appendix B: ESATAN Definitions

[*Note in sections 6-8 and 10, the character * should be used literally as seen*].

1. **Thermal Nodes**

D|B|X*n*[=*string* [=*cname:n* [+ *cname:n*...]]],
[T=] *r-exp* [,[C=] *r-exp*] [,[QA=] *r-exp*] [,[QE=] *r-exp*]
[,[QI=] *r-exp*] [,[QS=] *r-exp*] [,[QR=] *r-exp*]
[,[A=] *r-exp*] [,[ALP=] *r-exp*] [,[EPS=] *r-exp*]
[,[QAI=] *r-exp*] [,QEI=] *r-exp*] [,[QSI=] *r-exp*] [,[FX=] *r-exp*]
[,[FY=] *r-exp*] [,[FZ=] *r-exp*];

2. **Fluid Nodes**

F|J|R|H|K*n*[=*string* [=*cname:n* [+ *cname:n*...]]],
[A=] *r-exp* [,[FD=] *r-exp*] [,[FL=] *r-exp*]
[,[P=] *r-exp*] [,[FE=] *r-exp*] [,[T=] *r-exp*]
[,[FF=] *r-exp*] [,[FQ=] *r-exp*] [,[FM=] *r-exp*]
[,[FH=] *r-exp*] [,[FR=] *r-exp*] [,[FX=] *r-exp*]
[,[FY=] *r-exp*] [,[FZ=] *r-exp*] [,[FT=] *z-val*]
[,[VQ=] *r-exp*] [,[FRG =] *z-val*] [,[FLA =] *r-exp*]
[,[VOL =] *r-exp*] [,[PHI =] *r-exp*] [,[FW =] *r-exp*]
[,[CMP =] *r-exp*] [,[VDT =] *r-exp*][,[FST =] *z-val*];

3. Node Aliases

uname = D|F*n*;

4. **Thermal Conductors**

GL([*cname*:] *n*,[*cname*:] *n*) = *r-exp*;
GF([*cname*:] *n*,[*cname*:] *n*) = *r-exp*;
GR([*cname*:] *n*,[*cname*:] *n*) = *r-exp*;
GV([*cname*:] *n*,[*cname*:] *n*) = *r-exp*;

5. **Fluid Conductors**

M([*cname*:] *n*,[*cname*:] *n*) = *r-exp*;
GP([*cname*:] *n*,[*cname*:] *n*) = *r-exp*;

6. **User Constants**

[*type**]*uname* = *i-exp*|*r-exp*|*string*;

7. **Local Constants**

[*type**]*uname* = *i-val*|*r-val*|*string*;

8. **Arrays**

   [*type**]*uname*[(*dim*[,*dim*...]][/SIZE = *i-val*/] = *val*[,*val*...];

9. **Events**

   uname = *r-val*[,*r-val*];

10. **Functions**

    [*type**] FUNCTION [**length*] **uname*
    [(*uname*[,*uname* ...])] = *statement*;

11. **Subroutines**

    SUBROUTINE *uname* [(*uname* [,*uname* ...])]
    [LANG=**MORTRAN**|FORTRAN]
    statement
    .
    .
    .
    RETURN
    END

12. **$Keywords**

    $MODEL *mname* [, **REAL**|VIRTUAL][, GLOBALFILE = *myglobalfile*]
    [, **STANDARD**|VCHP][, TEMP = *tdesig*]
    $MODEL *mname* [, **REAL**|VIRTUAL][, GLOBALFILE = *myglobalfile*]
    [, TEMP = *tdesig*][, FLUID = *fname*][, STATE = *fstate*]
    $ELEMENT *cname*
    $SUBSTITUTIONS *cname*
    $EXTERNAL *cname*
    $REPEAT *cname*
    $VIRTUAL *cname*
    $LOCALS
    $NODES
    $ALIAS
    $CONDUCTORS
    $CONSTANTS
    $ARRAYS
    $EVENTS
    $TIMESTEP
    $OUTPUTS
    $SUBROUTINES
    $INITIAL
    $VARIABLES1
    $VARIABLES2
    $EXECUTION, DYSTOR = *i-val*

$OUTPUTS
$REAL
$INTEGER
$CHARACTER
$CONTROL
$TABLE
$ENDMODEL [*mname*]
$INCLUDE '*filename*'

13. **Mortran Statements** (additional to Fortran 77)

i) Select Case.

SELECT CASE (exp)
CASE *val* [:*val*] [,*val*[:*val*]]...
statement
.
.
.
[CASE ELSE]
[*statement*]
.
.
.
END SELECT

ii) While.

WHILE (*l-exp*)
statement
.
.
.
ENDWHILE

iii) Repeat.

REPEAT
statement
.
.
.
UNTIL (*l-exp*)

Text continues on next page

Appendix C: ESATAN Reference Forms

Please refer to Appendix A for a description of the terminology used below.

1. **Node**

   nent [: *cname*:] *n* [(*subs*)]

   ex: T:SUBMODEL:10

2. **Node Alias**

   nent [: *cname*:] *uname*[(*subs*)]

   ex: T:SUBMODEL:Thermocouple1

3. **Conductor**

   cent [:*cname*:] ([*cname*:] *n*, [*cname*:] *n*[,*m*]) [(*subs*)]

   ex: GL:SUBMODEL:(1,2)

4. **User Constant**

   [*cname*:] *uname*

   ex: SUBMODEL:USRCST

5. **Array**

   [*cname*:] *uname* [(*subs*, ...)]

   ex: SUBMODEL:SPECHEAT(2,2)

6. **Event**

   can only be referenced by specific event functions (BEFORE, AT, AFTER and BTWEEN);

   [*cname*:] *uname*

   ex: SUBMODEL:eclipse

7. **Local Constant**

   uname

8. **Local index**

   KL*n*

   ex: FOR KL1 =1 TO 9 DO

   KL2 = KL1 + 1

   GL(KL1, KL2) = 1.0;

   ENDDO

9. Subroutine

[*cname*:] *sname*

ex: CALL SUBMODEL:SUBNAM

Appendix D: ESATAN Entities

All ESATAN entities may be defined under the following categories:-

1. $MODEL
2. $BLOCK (Data and Operations), function, subroutine, line.
3. Node, node alias, conductor, user constant, array, event
4. Node quantities:

    - Thermal

        Status
        Label
        Temperature
        Heat Source (albedo)
        Heat Source (earth)
        Heat Source (internal)
        Heat Source (rest)
        Heat Source (solar)
        Area
        Absorptivity
        Emissivity
        Incident heat source (albedo)
        Incident heat source (earth)
        Incident heat source (solar)
        Cartesian coordinates

    - Fluid

        Status
        Label
        Heat transfer area
        Hydraulic diameter
        Length
        Static pressure
        Specific enthalpy
        Temperature
        Wall surface roughness
        Internal heat source
        Mass source rate
        Specific enthalpy of mass source
        Temperature of mass source
        Vapour quality of mass source
        Cartesian coordinates
        Fluid type
        Vapour quality
        Flow regime
        Flow area
        Volume
        Relative humidity
        Specific humidity of mass source
        Compliance
        Change in volume with time
        Fluid state descriptor
        Specific humidity

5. Array parameters:

        Dimensions
        Size

Text continues on next page

Appendix E: ESATAN Control Constants

Name	Type	Global / Local	User definable	Default	Purpose
ARITH	R	G	*	0.0	Arithmetic node cut-off point
CSGMIN	R	G		U	Minimum ratio of capacitance to sum of conductances (excluding arithmetic nodes and temperature boundaries)
DAMPM	R	G	*	0.5	Mass flow damping
DAMPT	R	G	*	1.0	Temperature damping
DTCOUR	R	G		U	Courant timestep limit
DTIMEI	R	G	*	U	Input time step
DTIMEU	R	G		U	Time step used
DTMAX	R	G	*	1.0E+10	Maximum time step
DTMIN	R	G	*	0.0	Minimum time step
DTPMAX	R	G	*	1.0E+10	Maximum allowable temperature change over a time step (non-arithmetic diffusion nodes only)
DTROCA	R	G	*	1.0E+10	Temperature rate of change convergence criterion
DTROCC	R	G		U	Calculated temperature rate of change convergence
ENBALA	R	G		0.0	Absolute energy balance
ENBALR	R	G		0.0	Relative energy balance
ENBDM	R	G		U	Computed root-sum-square nodal energy imbalance

Table E-1:ESATAN Control Constants

Name	Type	Global / Local	User definable	Default	Purpose
FCSGMN	R	G		U	Minimum ratio of capacitance to sum of conductances for fluid nodes (excluding arithmetic nodes and temperature boundaries)
FNCSGM	I	G		U	Fluid node of FCSGMN
FRLXCA	R	G	*	U	Temperature convergence criterion for fluid nodes
FRLXCC	R	G		U	Calculated temperature convergence for fluid nodes
FORMAT	C*8	G	*	'*'	Print format
FSTEPS	I	G	*	U	No. of fluid steps per solid step (FLTMTS)
GRAVX	R	G	*	0.0	Acceleration in the x-direction
GRAVY	R	G	*	0.0	Acceleration in the y-direction
GRAVZ	R	G	*	0.0	Acceleration in the z-direction
HEADER	C*132	G	*	' '	Output page header title
INBALA	R	G	*	1.0E10	Required absolute energy balance
INBALR	R	G	*	1.0	Required relative energy balance
INBDM	R	G	*	U	Required root-sum-square nodal energy imbalance
LENGTH	I	G	*	60	Page length for output (lines per page)
LINTYP	C*6	G	*	'SOLID'	Plot line type

Table E-1:ESATAN Control Constants

Name	Type	Global / Local	User definable	Default	Purpose
LOOPCT	I	G		0	Number of solution iterations
METHOD	I	G	*	0	Solver-specific
MINDP	R	G	*	1.0E-3	Minimum pressure difference recognised
MINFLO	R	G	*	1.0E-8	Minimum flow rate for convergence check
MLEVEL	I	G	*	2	Error message suppression flag
MODEL	I	G		0	Submodel number
MODULE	C*6	G		-	Current solution module
MSHORT	I	G	*	0	Error message length flag
NCSGMN	I	G		U	Node of CSGMIN
NDTROC	I	G		U	Node of DTROCC
NFRLXC	I	G		U	Node of FRLXCC
NLOOP	I	G	*	U	Maximum allowable no. of outer iterations
NLOOPH	I	G	*	NLOOP	No. of hydraulic iterations allowed
NLOOPT	I	G	*	NLOOP	No. of thermal iterations allowed
NODES	I	L		0	Number of nodes in current submodel
NODESM	I	L		0	Number of nodes in current submodel without included submodels
NRLXCC	I	G		U	Node of RELXCC
NRLXMC	I	G		U	Conductor of RELXMC
OPBLOK	C*10	G		' '	Holds name of current operation block

Table E-1:ESATAN Control Constants

Name	Type	Global / Local	User definable	Default	Purpose
OUTIME	C	G	*	'ALL'	Controls whether $OUTPUTS is executed: 'ALL' or 'NONE'
OUTINT	R	G	*	U	Output interval
PABS	R	G	*	0.0	Reference pressure
PARNAM	C*24	G		'ORIGINAL'	Parameter case name
PPOWER	R	G		U	Pump power (obsolete)
PRGFBK	I	G	*	10	Frequency of output to progress feedback file
QTRSOL	C*3	G	*	'YES'	Holds quasi-transient solution method for FGENFI
RELXCA	R	G	*	U	Temperature convergence criterion for solid nodes
RELXCC	R	G		U	Calculated temperature convergence for solid nodes
RELXMA	R	G	*	U	Convergence criterion for mass flow
RELXMB	R	G		U	Mass imbalance ratio on pump (obsolete)
RELXMC	R	G			Calculated flow rate convergence
RLCOUR	R	G	*	0.5	Fraction of Courant limit for timestep
SOLTYP	C*7	G		' '	Inner solution type for $VARIABLES1: 'THERMAL', 'FLUID' or 'HUMID'
SOLVER	C	G		' '	Solver type
STEFAN	R	G	*	5.6686E-8	Stefan-Boltzman constant

Table E-1:ESATAN Control Constants

Name	Type	Global / Local	User definable	Default	Purpose
STEPCT	I	G		0	Time step count - value is incremented every time $VARIABLES2 called
SUMFLL	I	G		0	Number of fluid loops taken for convergence
SUMTHL	I	G		0	Number of thermal loops taken for convergence
TABS	R	G	*	273.15	Temperature reference
TFORM	C*132	G	*	'*'	Table value format for thermal nodes
TFORMF	C*132	G	*	'*'	Table value format for fluidic nodes
THEAD	C*132	G	*	'*'	Table heading format
TIMELA	C*3	G	*	'NO'	Indicator for timelag simulation
TIMEM	R	G		TIMEO	Mean time over step
TIMEN	R	G		TIMEO	Time at end of step
TIMEND	R	G	*	0.0	Time at end of solution
TIMEO	R	G	*	0.0	Time at start of step
TRACE	I	G	*	0	Error message output flag
WIDTH	I	G	*	80	Page width for output (no. of characters per line, 80–132)

Table E-1:ESATAN Control Constants

Appendix F: Reserved Names

The names listed in the following table are regarded as "reserved" by the preprocessor. They comprise three basic types, the first being alphabetic characters which form a reserved name when followed by numeric characters. Such names are reserved mainly for nodal entities; in the table, the numeric characters which may follow are indicated by *n*.

The second type is formed of alphanumeric characters, and contains such names as Fortran keywords, ESATAN control constants and ESATAN library module names.

These two types of reserved name are recognised as such by the preprocessor and editor and are treated accordingly. The user should only employ them for their intended use; for instance, a reserved name may not be used as a user constant or array name - any attempt to do so will be detected at definition stage and rejected.

The third class contains names which are not reserved in the strict sense but which are used internally in the generated Fortran output file, as either declared variables or common-block names. Such names may be used for user-defined constants, arrays and subroutines/functions, but must not be used as local variables in operations blocks. Extreme care should therefore be taken with these names; of course, it is best not to use them at all! They are included in the table below, although the current version of the Preprocessor will not issue any warning if they are used.

A	AAPRG	AAPRGL	ABLCOF	ABLGEO	ABLINT
ABLMLT	ABLOUT	ABLPRO	ABLQRM	ABLREC	ABLSPT
ABLSUB	ABS	ACDDYU	ACGET	ACLOSS	ACONST
ACOS	ACSET	ADIM	ADIMVL	AELADD	AELCPY
AELDIV	AELEQ	AELGE	AELGT	AELINV	AELLE
AELLT	AELMLT	AELNE	AELSUB	AFTER	AFUNCI
AFUNCR	AFUNI2	AFUNR2	AGPRG	AGPRGL	AI
AIGET	AIMAG	AINF	AINT	AISET	ALINAM
ALIVAL	ALNAM	ALOG	ALOG10	ALP	ALP*n*
ALRADI	ALVAL	AMAX0	AMAX1	AMIN0	AMIN1
AMOD	AN	ANINT	AP	APS	AREA

Table F-1 ESATAN E

ARGET	ARITH	ARPT	ARRC	ARRI	ARRR
ARSET	ASIN	ASIZE	ASSIGN	AT	ATAN
ATAN2	AUNDF	AVADD	AVDIV	AVEQ	AVG
AVGE	AVGT	AVLE	AVLT	AVMLT	AVNE
AVSUB	AXISA3	AXISA4	AXISA5	AXISG	AXISXY
AXLABL	A*n*	BEFORE	BTWEEN	B*n*	C
CA	CABS	CALL	CAP	CASE	CCN
CCOS	CEXPv	CHAR	CL	CLOG	CLOSE
CLST69	CLST70	CLST71	CMFLH	CMFLX	CMODND
CMP	CMPLX	CMP*n*	CMTYPE	CNDFN1	CNDFN2
CNDFN3	CNDFN4	CNDFN5	COLCHG	COMMON	COND
CONDU	CONJG	CONTINUE	COORDS	COS	COSH
CP	CPHSON	CPHSX	CPU	CPUINC	CPUTOT
CQEVL	CSEQ	CSGDMP	CSGMIN	CSIN	CSP
CSQRT	CU	CUD	CV	C*n*	DABS
DACOS	DAMPM	DAMPT	DASIN	DATA	DATAN
DATAN2	DAYDAT	DAYTIM	DBCOEF	DBLE	DBSNCF
DCOS	DCOSH	DDIM	DELETE	DEXP	DIM
DINT	DLOG	DLOG10	DMAX1	DMIN1	DMOD
DMPCLR	DMPENT	DMPFR	DMPGFF	DMPGFS	DMPTHM
DMPTMD	DNINT	DO	DO*n*	DPROD	DPSDV
DRAWAR	DRAWL	DRAWNL	DSIGN	DSIN	DSINH
DSQRT	DTAN	DTANH	DTCOUR	DTIMEI	DTIMEU
DTIMIF	DTIMUT	DTINP	DTIST	DTMAX	DTMIN

Table F-1 ESATAN E

DTPINP	DTPIST	DTPMAX	DTROCA	DTROCC	DVDHU
DVDPU	DYCORE	D*n*	ELSE	ELSEIF	EMM
ENBALA	ENBALR	ENBNDM	END	ENDIF	ENTH
ENTHU	ENTRY	EPS	EPS*n*	ERRADI	EVALFR
EVLQST	EXP	FA	FAILUR	FAREA	FCMP
FCSEQ	FCSGMN	FCSP	FCUD	FD	FDIA
FDPSDV	FD*n*	FE	FENTH	FETCHT	FE*n*
FF	FFRIC	FFST	FF*n*	FGENFI	FGENSS
FGM	FGP	FH	FHSOR	FH*n*	FILE
FINITS	FIUD	FL	FLA	FLABEL	FLAF
FLAGS	FLA*n*	FLEN	FLFLG	FLG	FLLPCT
FLOAT	FLRG	FLST	FLSTT	FLTMTS	FLTNSS
FLTNTF	FLTNTS	FLTOPP	FLUXF	FLUXGF	FLUXGL
FLUXGR	FLUXGT	FLUXL	FLUXMF	FLUXML	FLUXMR
FLUXR	FLUXT	FL*n*	FM	FMFLO	FMSOR
FM*n*	FNCSGM	FNNUM	FNP	FNPAIR	FNST
FNUMB	FORM	FORMAT	FPCS	FPCSP	FPHI
FPRESS	FPSRC	FQ	FQSOR	FQUAL	FQ*n*
FR	FRG	FRG*n*	FRLXCA	FRLXCC	FRSOR
FRUD	FR*n*	FS	FSPHUM	FSPRNC	FSPRNI
FSPRNR	FST	FSTEPS	FST*n*	FS*n*	FT
FTEMP	FTYPE	FTYPEC	FTYPEI	FT*n*	FVCF
FVDT	FVOL	FW	FWSOR	FWVSNK	FW*n*

Table F-1 ESATAN E

FX	FXN	FX*n*	FY	FYN	FY*n*
FZ	FZN	FZ*n*	F*n*	GETA	GETALP
GETC	GETCCR	GETCMP	GETEPS	GETFD	GETFE
GETFF	GETFH	GETFL	GETFLA	GETFM	GETFQ
GETFR	GETFRG	GETFST	GETFT	GETFW	GETFX
GETFY	GETFZ	GETGF	GETGF2	GETGL	GETGL2
GETGP	GETGR	GETGR2	GETGV	GETGV2	GETL
GETM	GETP	GETPHI	GETQA	GETQAI	GETQE
GETQEI	GETQI	GETQR	GETQS	GETQSI	GETT
GETVDT	GETVOL	GETVQ	GF	GFFLG	GFLO
GFST	GL	GLFLG	GLIN	GLOC	GLOI
GLOR	GLST	GM	GOTO	GOTO*n*	GP
GP*n*	GR	GRAD	GRAVX	GRAVY	GRAVZ
GRFLG	GRLABL	GRPAVE	GRPLST	GRPMAX	GRPMIN
GRPSUM	GRST	GV	GVEW	GVFLG	GVST
HCAP	HEADER	HSATU	HTBOKR	HTCHAT	HTCHEN
HTCOEF	HTDIBO	HTDOBS	HTDORO	HTFLAG	HTROMY
HTTRAV	HUMDMP	H*n*	IA	IABS	ICHAR
IDIM	IDINT	IDNINT	IF	IFIX	IG
IL	INBALA	INBALR	INBNDM	INCORE	INDEX
INT	INTCY1	INTCY2	INTCY3	INTCYC	INTEGL
INTERP	INTGL1	INTGL2	INTNOD	INTRP1	INTRP2
INTRP3	INTRPA	ISIGN	IU	IUD	J*n*

Table F-1 ESATAN E

K*n*	L	LABEL	LEN	LENGTH	LGE
LGT	LINTYP	LIST69	LIST70	LIST71	LLE
LLT	LOCC	LOCI	LOCR	LOG	LOG10
LOOPCT	LQOPEN	LQPARK	LSTSQ	LU	LU*n*
LW	LW*n*	L*n*	M	MATCML	MATCMN
MATCMX	MATCSM	MATDIA	MATDTI	MATDTR	MATIDN
MATINV	MATMLT	MATRML	MATRSM	MATSMI	MATSMR
MATTRN	MAX	MAX0	MAX1	MCAPRG	MCCNST
MCGPRG	MD	MDIR	MDLOFF	MDLON	METHOD
MFFB	MFLH	MFLX	MIN	MIN0	MIN1
MINDP	MINFLO	MLEVEL	MOD	MODCHK	MODEL
MODNOD	MODULE	MRAPRG	MRCNST	MRGPRG	MSHORT
MST	MTYPE	MTYPST	M*n*	NAMA	NAMC
NAMM	NAMU	NCSEQ	NCSGMN	NCSP	NDMFL
NDTROC	NFRLXC	NINT	NLAB	NLOOP	NLOOPH
NLOOPT	NNUM	NODES	NODESM	NODFN1	NODFN2
NODNUM	NPCS	NPCSP	NRLXCC	NRLXMC	NST
NST2	NUMBR	NUVRE	OPBLOK	OPEN	OUTIME
OUTINT	P	PABS	PARNAM	PAUSE	PCS
PCSP	PHI	PHI*n*	PHSDST	PHSON	PHSX
PHSXST	PLBEPI	PLBUVA	PLDIFF	PLNOZZ	PLORIF
PLOTT	PLSLVA	PMPSPD	POLYNM	PPOWER	PRGFBK
PRHEAD	PRINT	PRNCSV	PRNDBL	PRNDPT	PRNDTB
PROGHD	PROGRS	PROPS1	PROPS2	PROPS3	PRPAGE

Table F-1 ESATAN E

PRPLT	PRQBAL	PRQBOU	PRQG	PRQLIN	PRQNOD
PRTARR	PRTC	PRTCR	PRTCRM	PRTGRP	PRTNAV
PRTNSM	PRTTMD	PRTUC	PSATU	PSRC	PTSINK
PU	PYRO	P*n*	QA	QAI	QAIS
QAI*n*	QALB	QA*n*	QE	QEAR	QEI
QEIS	QEI*n*	QEVL	QE*n*	QI	QINT
QI*n*	QQ	QQ*n*	QR	QRATES	QRES
QR*n*	QS	QSI	QSIS	QSI*n*	QSOL
QS*n*	QTRSOL	QUAL	Q*n*	RA	READ
REAL	REDIM1	REDIM2	REDIM3	RELXCA	RELXCC
RELXMA	RELXMB	RELXMC	REPEAT	RESET	RESULT
RETURN	REWIND	RG	RHO	RHOU	RL
RLCOUR	RLVALV	RSTART	RSTORE	RU	RUD
R*n*	SAVE	SAVET	SCALE1	SCALE2	SCALE3
SCALEV	SELECT	SETA	SETALP	SETC	SETCMP
SETDP	SETDPV	SETEPS	SETERX	SETFD	SETFE
SETFF	SETFH	SETFL	SETFLA	SETFM	SETFQ
SETFR	SETFRG	SETFST	SETFT	SETFW	SETFX
SETFY	SETFZ	SETGF	SETGF2	SETGL	SETGL2
SETGP	SETGR	SETGR2	SETGV	SETGV2	SETL
SETM	SETNDI	SETNDR	SETNDZ	SETP	SETPHI
SETQA	SETQAI	SETQE	SETQEI	SETQI	SETQR
SETQS	SETQSI	SETT	SETVDT	SETVOL	SETVQ

Table F-1 ESATAN E

SHUM	SIG	SIGMAU	SIGN	SIN	SINH
SLCRNC	SLFRTF	SLFRWD	SLFWBK	SLGEAR	SLGRDJ
SLMODE	SLMODR	SLRADI	SN	SNGL	SOLCYC
SOLTYP	SOLVCG	SOLVER	SOLVFM	SOLVIT	SOLVLW
SOLVSM	SORT	SPHUM	SPRNC	SPRNDE	SPRNDM
SPRNDN	SPRNI	SPRNOD	SPRNR	SPRSIZ	SQRT
STAT2	STATM	STATRP	STATST	STATUS	STEFAN
STEPCT	STOP	STORMM	STORXY	STRLNA	STVRE
SUBMDN	SUBMOD	SUMFLL	SUMTHL	SUNDCK	SZEROR
S*n*	T	TABS	TAN	TANH	TEMP
TEMPU	TF	TFORM	TFORMF	THEAD	THEN
THLPCT	THRMST	TIMELA	TIMEM	TIMEN	TIMEND
TIMENT	TIMEO	TIMEOT	TMAIN	TMAIN1	TQDUMP
TRACE	TSATU	TSIDE	TSINK	TSVAL	TVAL
T*n*	UCNC	UCNI	UCNR	UCPT	UNIT
UNTIL	USRNDC	USRNDI	USRNDR	VCF	VCHP
VCHPD	VCN	VDT	VDT*n*	VFAC	VISC
VISCU	VLNK	VLSTAT	VOL	VOLST	VOL*n*
VQ	VQUAL	VQ*n*	VSEQ	VSL	VSLIP
V*n*	WHILE	WIDTH	WORK	WORKI	WRITE
WVSINK	WVSNK	XCOND	XCONDS	XCP	XCPS
XENTH	XENTHS	XJT	XJTS	XKT	XKTS
XPRES	XPRESS	XRHO	XRHOS	XSIG	XSIGS

Table F-1 ESATAN E

XTEMP	XTEMPS	XVISC	XVISCS	X*n*	ZDAYDT
ZDAYTM	ZW	ZW*n*	ZX	ZX*n*	ZY
ZY*n*	ZZB	ZZB*n*	ZZC	ZZC*n*	

Table F-1 ESATAN E

Appendix G: SINDA to ESATAN Translator

G.1 Introduction

The SINDA to ESATAN translator performs a line conversion of a SINDA input deck to an ESATAN source file. Whilst it verifies the syntax of individual SINDA data block lines it makes the assumption that the SINDA deck is consistent. That is, for example, no check is made that the nodes referenced in a conductor definition actually exist. In addition the syntax of M type statements in the operations blocks is not checked, the lines purely being scanned for SINDA entities to be converted according to the rules defined below.

The SINESA translator cannot provide a full conversion and so work has to be done by the user subsequent to the translation. Most of this work involves the operations blocks and the SINDA mixed type arrays.

G.2 User Interface

On beginning the translation, the user is prompted for certain information. Firstly the SINDA input deck name and the required ESATAN source file name. A format for the writing of integers within the ESATAN source file (e.g. I6) and for writing out real number (e.g. 1PE11.4) must be supplied together with a value for the Stefan-Boltzmann constant (e.g. 5.67E-8) and the ESATAN model name. Each of these four values is read as a character string by the program and may not exceed ten characters in length.

G.2.1 Title Block

The title block is fully translated, with the SINDA title becoming the ESATAN title after the $MODEL card. The model name is that given by the user.

G.2.2 Data Blocks

The program provides a reasonably full translation of the data blocks according to the following rules. Throughout the data blocks multiple SINDA definitions on a single line are supported, as are most of the generating mnemonics (e.g. GEN, SIM, DPM). Both of these input forms lead to one entity per line ESATAN definitions; that is a GEN statement to generate 10 nodes is translated to 10 ESATAN node definition lines. In every data block other than the source data block, in line, C and REM type comments are supported. C and REM comments are translated to full ESATAN comment lines, while in line comments may suffer truncation if the ESATAN data line is longer than the equivalent SINDA line. The variable format facility is fully supported.

Node Data:- Node numbers may be of up to six digits and the user-supplied numbers are preserved by the translation.

The following node data mnemonics are fully supported by the translator; 3 blanks, CAL, GEN, SIV, SPV, SIM, SPM, DIV, DPV, DIM, DPM, BIV. SINDA diffusion nodes translate to ESATAN diffusion nodes, arithmetic nodes to zero capacitance diffusion nodes and boundary nodes to boundary nodes.

Source Data:- The following source data mnemonics are fully supported by the translator; 3 blanks, GEN, SIV, SIT, DIT, DTV, CYC. All SINDA sources are translated to ESATAN QR sources with the SINDA node and source data blocks being combined into the single ESATAN $NODES block.

Conductor Data:-The following conductor data mnemonics are fully supported by the translator; 3 blanks, CAL, GEN, SIV, SPV, SIM, SPM, DIV, DPV, DIM, DPM, BIV, PIV and PIM. Negative flags to indicate radiative links, one-way flow links and temperature dependence on one nodal temperature rather than the average link temperature are all supported. The value of all radiative links is adjusted to account for the exclusion of the Stefan-Boltzmann constant in ESATAN input, using the user-supplied value of Stefan-Boltzmann.

Constants Data:- User constant numbers may be of up to five digits and are translated into ESATAN user constant names being K followed by the SINDA constant number. For example, SINDA user constant 10 becomes ESATAN constant K10 and SINDA constant 12345 becomes ESATAN constant K12345. This is done irrespective of the type of the constant.

Both the blank and GEN input formats for user constants are supported with user constant numbers being translated as already described. The type of the constant is deduced from the value assigned to it in the CONSTANTS DATA block.

The following control constants are translated (the ESATAN equivalent in brackets where this differs from the SINDA); CSGMIN, DAMPD (DAMPT), DRLXCA (RELXCA), DRLXCC (RELXCC), DTIMEH (DTMAX), DTIMEI, DTIMEL (DTMIN), DTMPCA (DTPMAX), ENGBAL (ENBALA), LOOPCT, NCSGM (NCSGMN), NLOOP, OUTPUT (OUTINT), TIMEM, TIMEN, TIMEND, TIMEO. In addition the translation of dummy control constants ITEST, JTEST, KTEST, LTEST, MTEST, RTEST, STEST, TTEST, UTEST and VTEST to ESATAN user constants is supported. No other SINDA control constants have user-addressable ESATAN equivalents so their definition and use is flagged as non-translatable.

ESATAN control constant STEFAN is input using the user-supplied value.

Array Data:- Array numbers may be of up to six digits and are translated into ESATAN arrays of name ARn where n reflects the order in which the arrays are encountered in the SINDA deck. That is, if the third array referenced or defined in the deck is array 469, the equivalent ESATAN array is called AR3. The correspondence between SINDA array numbers and ESATAN array names is indicated in comments within the $ARRAYS block.

It is in defining the user arrays that most thought must be given by the user in relation to the translation. The use of arrays is more formally regulated in ESATAN than in SINDA and several uses of arrays in SINDA are not allowed in ESATAN. Within the ARRAY DATA block the major restriction arises from ESATAN's rejection of mixed type arrays, hence any SINDA array which is assigned values of more than one type is marked non-translatable. Similarly SINDA arrays of character data are not translated.

Another major difference between the definition of arrays in SINDA and ESATAN is the ability to define multi-dimension arrays in ESATAN. Where arrays have been referred to in node, source and conductor definitions the dimensions of the array are deduced (mnemonics requiring polynomial coefficients imply a one-dimensional array whilst mnemonics requiring interpolation imply a two-dimensional array), and these dimensions used. Otherwise a one-dimensional array is written preceded by an appropriate warning comment. As already indicated, each array in the ESATAN source file is preceded by a comment giving the SINDA array number.

The final important difference between the definition of arrays in SINDA and ESATAN is the provision of the array length as the first element of the array in SINDA. This is not so for ESATAN arrays and so any use of arrays in the operations blocks which is marked non-translatable must be carefully examined.

G.2.3 Operations Blocks

Although all the operations blocks are converted by the translator, the scope of the translation is significantly less than that within the data blocks. Each of the three types of statement: SINDA, F-type and M-type will be dealt with in turn.

SINDA Statement:- The translator maintains a list of translatable SINDA library routines, together with their ESATAN equivalents. Any SINDA routine with an ESATAN equivalent is translated unless it contains a reference to the title (Hn), to an array element (An + e) or to a non-translatable control constant among its arguments. That is, references to Tn, Cn, Qn, Gn, Kn and An are all translated into the equivalent ESATAN call, where n is a literal integer constant. A reference to any routine other than a

translatable SINDA routine leads to the line being marked non-translatable. A list of translatable SINDA routines with their ESATAN equivalents is given at the end of this appendix.

F-type Fortran Statements:- F-type statements are echoed to the output file with the characters "< F" in columns 74 and 75. No translation is attempted.

M-type Fortran Statements:- The following SINDA entities are translated in M-type statements; Tn, Qn, Cn, Gn, An, Kn, XKn (where n is a literal integer constant) and those control constants identified in Section G.2.2 ("Constants Data") as having ESATAN equivalents. Any other SINDA entities are considered non-translatable. Since the order of storage of quantities and conductances differs between the two programs, no attempt is made to translate SINDA entities where arguments are not literal integer constants. Any line containing non- translatable SINDA entities is marked non-translatable.

Subroutines Block:- Subroutines defined within FSTART/FSTOP or MSTART/MSTOP statements are of the language type implied. Where the language type is not implicit an FCALL statement immediately after the subroutine or function statement implies the Mortran language otherwise Fortran language is assumed. A message is output where the language type is ambiguous.

G.3 Miscellaneous Features

G.3.1 Dynamic Storage

The DIMENSION X(.) statement and NTH/NDIM assignments are recognised and are translated to DYSTOR= on the ESATAN $EXECUTION card.

G.3.2 Comments

Full line comments are translated to Fortran style C comments. In line comments are translated subject to the same condition as applied in the data blocks, that truncation is possible if the ESATAN line is longer than the SINDA line.

G.3.3 F/M-START statements

The FSTART, FSTOP, MSTART and MSTOP statements are fully supported.

G.3.4 PARAMETER RUNS

Both initial and final parameter runs are fully supported by the translator. That is, any valid SINDA parameter run produces an equivalent, valid ESATAN parameter run.

SINDA	ESATAN
CINDSB	CALL SOLVIT
CINDSL	CALL SOLVIT
CINDSM	CALL SOLVIT
CINDSS	CALL SOLVIT
CNBACK	CALL SLFWBK
CNDUFR	CALL SLFRWD
CNEXPN	CALL SLFRWD
CNFAST	CALL SLFRWD
CNFRDL	CALL SLFRWD
CNFRWD	CALL SLFRWD
CNFWBK	CALL SLFWBK
CNQUIK	CALL SLFRWD
CNVARB	CALL SLFWBK
CPRINT	CALL PRNDBL(‘ ‘, ‘C’, CURRENT)
CWFWBK	CALL SLFWBK
D11CYL(P, X, An, Y)	Y=INTCY1(X, ARm, 1, P, 0.0)
D11MCY(P, X, An, Z, Y)	Y=INTCY1(X, ARm, 1, P, 0.0)*Z
D11MDA(X, An1, An2, Z, Y)	Y=INTERP(X, ARm1, ARm2, 1)*Z
D12CYL(P, X, An, Y)	Y=INTCY1(X, ARm, 2, P, 0.0)
D12MDA(X, An1, An2, Z, Y)	Y=INTERP(X, ARm1, ARm2, 2)*Z
D12NCY(P, X, An, Z, Y)	Y=INTCY1(X, ARm, 2, P, 0.0)*Z

Table G-1: SINESA Translatable Calls

SINDA	ESATAN
D1D1DA(X, An1, An2, Y)	Y=INTERP(X, ARm1, ARm2, 1)
D1D1WM(X, An, Z, Y)	Y=INTRP1(X, ARm, 1)*Z
D1D2DA(X, An1, An2, Y)	Y=INTERP(X, ARm1, ARm2, 2)
D1D2WM(X, An, Z, Y)	Y=INTRP1(X, ARm, 2)*Z
D1DEG1(X, An, Y)	Y=INTRP1(X, ARm, 1)
D1DEG2(X, An, Y)	Y=INTRP1(X, ARm, 2)
D1M1DA(X1, X2, An1, An2, Y)	Y=INTERP((X1 + X2) * 0.5, ARm1, ARm2, 1)
D1M1MD(X1, X2, An1, An2, Z,Y)	Y=INTERP((X1 + X2) * 0.5, ARm1, ARm2, 1) *Z
D1M1WM(X1, X2, An, Z, Y)	Y=INGRP1((X1 + X2) * 0.5, ARm, 1) * Z
D1M2DA(X1, X2, An1, An2, Y)	Y=INTERP((X1 + X2) * 0.5, ARm1, ARm2, 2)
D1M2MD(X1, X2, An1, An2, Z,Y)	Y=INTERP((X1 + X2) * 0.5, ARm1, ARm2, 2) *Z
D1M2WM(X1, X2, An, Z, Y)	Y=INTRP1((X1 + X2) * 0.5, ARm, 2) * Z
D1MDG1(X1, X2, An, Y)	Y=INTRP1((X1 + X2) * 0.5, ARm, 1)
D1MDG2(X1, X2, An, Y)	Y=INTRP1((X1 + X2) * 0.5, ARm, 2)
GPRINT	CALL PRNDBL(‘ ‘, ‘GF, GL, GR’, CURRENT)
QIPRNT	CALL PRNDBL(‘ ‘, ‘QR’, CURRENT)
TFPRNT	CALL PRNDBL(‘ ‘, ‘T’, CURRENT)
TPRINT	CALL PRNDBL(‘ ‘, ‘T’, CURRENT)
TPRNTF	CALL PRNDBL(‘ ‘, ‘T’, CURRENT)

Table G-1: SINESA Translatable Calls

Appendix H: ESATAN to SINDA Translator

H.1 Introduction

The ESASIN program will output a named ESATAN input model in SINDA format.

A list of translatable library subroutines and functions is given below. Function references are translated by preceding the statement containing the function reference with an appropriate SINDA library subroutine call and replacing the function reference as shown below:

```
Y = 10.0*INTRP1 (X,ARR,1)        CALL D1DEG1(X,An,XXXXX1)
                                 Y = 10.0*XXXXX1
```

The ESATAN to SINDA Translator does not support translating models with an associated Analysis Case Definition (ACD) file (the assignment of the static and dynamic data from the ACD file will be omitted from the generated SINDA model). See Section 3.10 for information on the data contained within the ACD file.

H.2 Translatable Routines

ESATAN	SINDA
CALL SOLVIT	CALL CINDSB
CALL SLFRWD	CALL CNFRWD
CALL SLFWBK	CALL SLFWBK
CALL PRNDBL(‘ ‘,’T’,CURRENT)	CALL TFPRNT
CALL PRNDBL(‘ ‘,’C’,CURRENT)	CALL CPRINT
CALL PRNDBL(‘ ‘,’GL,GF,GR’,CURRENT)	CALL GPRINT
INTRPI(X,ARR,1)	CALL DIDEG1
INTERP(X,ARR1,ARR2,1)	CALL DIDIDA
INTCYI(X,ARR,1,P,0.0)	CALL D11CYL
INTCYI(X,ARR,2,P,0.0)	CALL D12CYL

Table H-1: ESATAN Translatable Calls

Text continues on next page

Appendix I: A Fluid Property Definition

An example of a user-defined fluid, as described in Section 4.2.2, is shown here.

```
$FLUID WHISKY
#
# 15-year-old single malt
#
# Liquid phase - assume incompressible
# Vapour phase - assume ideal gas


#
# Predefined symbols:-
#
#   P         Pressure                  \
#   T         Temperature               /  FINTRP1, FINTRP2, FINTRPA
#   H         Enthalpy                  \  & PROC data options
#   X         Vapour quality            /
#   FTI       Fluid type integer                  \
#   NODE      Node reference                       |
#   <PROP>L   Liquid saturation value of property  /
#             ($TWO_PHASE only)                   \
#   <PROP>V   Vapour saturation value of property   \  PROC data option
#             ($TWO_PHASE only)                     /
#   PABS      Absolute pressure offset used in     /
#             model                               \
#   TABS      Absolute temperature offset used     |
#             in model                            /
#


   #
   # Density
   #
   $RHO

      $LIQUID
      # Temperature-dependent only - use saturation values
      FTABS = 0.0;  # T absolute
      PROC
C
      DOUBLE PRECISION PS, XPRESS, XRHOS
      PS = XPRESS(T - TABS, FTI, NODE)
      RHO = XRHOS(PS, T - TABS, X, FTI, NODE)
C
      END_PROC

      $SAT_LIQ
      # Celsius
      FINTRP1 (T, 1)
       0.0, 963.0,  10.0, 958.0,  20.0, 953.0,  30.0, 948.0,
      40.0, 943.0,  50.0, 938.0,  60.0, 933.0;

      $TWO_PHASE
      PROC
      RHO = 1.0 / ((1.0 - X) / RHOL + X / RHOV)
      END_PROC

      $SAT_VAP
      # Celsius
      FINTRP1 (T, 1)
      5.0E00,  9.1E-3,   1.0E01,  1.2E-2,   1.5E01,  2.0E-2,   2.0E01,  2.7E-2,
      2.5E01,  4.5E-2,   3.0E01,  6.2E-2,   3.5E01,  8.3E-2,   4.0E01,  1.0E-1,
      5.0E01,  1.4E-1,   6.0E01,  1.8E-1,   7.0E01,  2.2E-1,   8.0E01,  2.5E-1,
      9.0E01,  2.9E-1,   1.0E02,  3.3E-1,   1.2E02,  3.9E-1,   1.4E02,  4.7E-1;

      $VAPOUR
      # Ideal gas
      # Absolute temperature & pressure
      FTABS = 0.0;
      FPABS = 0.0;
      #
      PROC
      RHO = P / (435.2 * T)
      END_PROC
```

```
   #
   # Specific heat
   #
   $CP

      $LIQUID
      # Temperature-dependent only - use saturation values
      FTABS = 0.0;  # T absolute
      PROC
C
      DOUBLE PRECISION PS, XPRESS, XCPS
      PS = XPRESS(T - TABS, FTI, NODE)
      CP = XCPS(PS, T - TABS, X, FTI, NODE)
C
      END_PROC

      $SAT_LIQ
      # degrees Celsius
      FINTRP1 (T, 2)
        0.0, 3800,   10.0, 3780,   20.0, 3772,   30.0, 3761,
       40.0, 3754,   80.0, 3620,  100.0, 3599,  120.0, 3581,
      150.0, 3553,  200.0, 3506,  250.0, 3456;

      $TWO_PHASE
      PROC
      CP = CPL + X * (CPV - CPL)
      END_PROC

      $SAT_VAP
      # degrees Celsius
      FINTRP1 (T, 2)
       0.0, 1572,   10.0, 1576,   20.0, 1580,   30.0, 1582,
      40.0, 1585,   80.0, 1596,  100.0, 1618;

      $VAPOUR
      # Absolute temperature & pressure
      FTABS = 0.0;
      FPABS = 0.0;
      #
      FINTRP2 (P, T, 1)
            P =  0.1E5,  1.0E5,  1.5E5,  2.0E5,
      T = 273.,  1572.,  1601.,  1632.,  1659.,
      T = 293.,  1580.,  1605.,  1630.,  1662.,
      T = 313.,  1585.,  1610.,  1635.,  1667.,
      T = 353.,  1595.,  1620.,  1645.,  1677.,
      T = 403.,  1615.,  1640.,  1665.,  1697.;


   #
   # Conductivity
   #
   $COND

      $LIQUID
      # Temperature-dependent only - use saturation values
      FTABS = 0.0;  # T absolute
      PROC
C
      DOUBLE PRECISION PS, XPRESS, XCONDS
      PS = XPRESS(T - TABS, FTI, NODE)
      COND = XCONDS(PS, T - TABS, X, FTI, NODE)
C
      END_PROC

      $SAT_LIQ
      CONSTANT
      .6;

      $TWO_PHASE
      PROC
      COND = (1.0 - X) * CONDL + X * CONDV
      END_PROC

      $SAT_VAP
      CONSTANT
      0.02;

      $VAPOUR
      # Temperature-dependent only - use saturation values
      FTABS = 0.0;  # T absolute
```

```
      PROC
C
      DOUBLE PRECISION PS, XPRESS, XCONDS
      PS = XPRESS(T - TABS, FTI, NODE)
      COND = XCONDS(PS, T - TABS, X, FTI, NODE)
C
      END_PROC


   #
   # Viscosity
   #
   $VISC

      $LIQUID
      # Temperature-dependent only - use saturation values
      FTABS = 0.0;  # T absolute
      PROC
C
      DOUBLE PRECISION PS, XPRESS, XVISCS
      PS = XPRESS(T - TABS, FTI, NODE)
      VISC = XVISCS(PS, T - TABS, X, FTI, NODE)
C
      END_PROC

      $SAT_LIQ
      CONSTANT
      0.924D-3;

      $TWO_PHASE
      PROC
      VISC = (1.0 - X) * VISCL + X * VISCV
      END_PROC

      $SAT_VAP
      CONSTANT
      2.49D-6;

      $VAPOUR
      # Temperature-dependent only - use saturation values
      FTABS = 0.0;  # T absolute
      PROC
C
      DOUBLE PRECISION PS, XPRESS, XVISCS
      PS = XPRESS(T - TABS, FTI, NODE)
      VISC = XVISCS(PS, T - TABS, X, FTI, NODE)
C
      END_PROC


   #
   # Surface tension
   #
   $SIG

      $SAT_LIQ
      FTABS = 0.0; # Absolute temperatures
      #
      PROC
C
C SIGMA0 - Reference value of surface tension
C TC     - Critical temperature
C TR     - Reduced temperature
C
      DOUBLE PRECISION SIGMA0, TC, TR
C
      PARAMETER (SIGMA0 = 0.072616D0, TC = 628.46D0)
C
      TR = 1.0D0 - T / TC
      SIG = SIGMA0 * TR ** 1.22D0
C
      END_PROC


   #
   # Specific enthalpy
   #
   $ENTH

      $LIQUID
      #
```

```
      # Incompressible - enthalpy independent of pressure
      #
      FINTRP1  (T, 2)
       0.0,       0.0,
      20.0,   76000.0,
      40.0,  151440.0,
      60.0,  226520.0,
      80.0,  300260.0,
      100.0, 372660.0,
      120.0, 444640.0,
      150.0, 552070.0;

      $SAT_LIQ
      FINTRPA (T, 1)
      0.0, 20.0,
      0.0, 76000., 151440., 226520., 300260., 372660., 444640., 516260.,
      587507., 658379., 728875., 798995.;

      $TWO_PHASE
      #
      # Interpolate on vapour quality
      #
      PROC
C
      ENTH = ENTHL + X * (ENTHV - ENTHL)
C
      END_PROC

      $SAT_VAP
      FINTRPA (P, 2)
      0.01D5, 0.01D5,
      2.115200D06, 2.147500D06, 2.161180D06, 2.174860D06, 2.188447D06;
      0.05D5, 0.05D5,
      2.188447D06, 2.225413D06, 2.254819D06, 2.284475D06, 2.313756D06,
      2.342743D06, 2.371594D06, 2.400622D06;
      0.5D5, 0.25D5,
      2.457094D06, 2.570781D06, 2.683232D06, 2.781643D06;

      $VAPOUR
      #
      # Approximately,
      #
      #           h  =  h_s + Cp_s (T - T_s)
      #
      # where saturation properties are evaluated at the current pressure.
      #
      FPABS = 0.0;   # P absolute
      FTABS = 0.0;   # T absolute
      PROC
C
      DOUBLE PRECISION CPS, HS, TS, XCPS, XENTHS, XTEMPS
      TS = XTEMPS(P - PABS, FTI, NODE) + TABS
      HS = XENTHS(P - PABS, TS - TABS, X, FTI, NODE)
      CPS = XCPS(P - PABS, TS - TABS, X, FTI, NODE)
C
      ENTH = HS + CPS * (T - TS)
C
      END_PROC


   #
   # Temperature
   #
   $TEMP

      $LIQUID
      #
      # Incompressible - temperature independent of pressure
      #
      FINTRP1  (H, 2)
      0.0, 0.0,
      76000.0, 20.0,
      151440.0, 40.0,
      226520.0, 60.0,
      300260.0, 80.0,
      372660.0, 100.0,
      444640.0, 120.0,
      552070.0, 150.0;

      $SAT
      FINTRP1 (P, 2)
```

```
      0.006E5,  0.0,  0.01E5,  4.0,  0.02E5, 12.5,
       0.05E5, 23.3,   0.1E5, 33.1,   0.5E5, 95.7,
        1.0E5,158.8,   1.5E5,214.4,   2.0E5,270.0;

      $VAPOUR
      #
      # Approximately,
      #
      #                      h - h_s
      #           T = T_s + -------
      #                       Cp_s
      #
      # where saturation properties are evaluated at the current pressure.
      #
      FPABS = 0.0;   # P absolute
      FTABS = 0.0;   # T absolute
      PROC
C
      DOUBLE PRECISION CPS, HS, TS, XCPS, XENTHS, XTEMPS
      TS = XTEMPS(P - PABS, FTI, NODE) + TABS
      HS = XENTHS(P - PABS, TS - TABS, X, FTI, NODE)
      CPS = XCPS(P - PABS, TS - TABS, X, FTI, NODE)
C
      TEMP = TS + (H - HS) / CPS
C
      END_PROC


   #
   # Pressure
   #
   $PRES

      $SAT
      FINTRPA (T, 1)
      0.0, 5.0,
      0.006E5, 0.011E5, 0.015E5, 0.025E5, 0.034E5;
      20.0, 10.0,
      0.034E5, 0.082E5, 0.14E5;
      40.0, 20.0,
      0.14E5, 0.26E5, 0.39E5, 0.53E5, 0.68E5, 0.84E5, 1.01E5, 1.19E5;


   #
   # Joule-Thompson coefficient
   #
   $JT

      $LIQUID
      CONSTANT
      0.0E0;

      $VAPOUR
      CONSTANT
      0.0;


   #
   # Isothermal compressibility
   #
   $KT

      $LIQUID
      CONSTANT
      0.0;

      $VAPOUR
      FPABS = 0.0D0;   # Absolute pressure
      #
      PROC
C
      KT = 1.0 / P
C
      END_PROC


$END_FLUID # WHISKY
```

Appendix J: The ESASRC Program

The ESASRC program has been withdrawn from version 9 of ESATAN.

Text continues on next page

Appendix K: Physical Correlations in FHTS

K.1 Introduction

The heat transfer and pressure drop correlations used within FHTS describe the calculations made internally, at solution time, where the user requires them. That is, where a linear conductance between a fluid node and a thermal node is assigned as 'GL = *' , or where a fluid node is assigned a surface roughness other than zero. All correlations given are strictly valid for circular tubes only, but may be applied to pipes of non-circular section.

The diameter d used in the correlations is the hydraulic diameter, defined as

$$d = \frac{4A}{P}$$

where A is the flow area and P is the wetted perimeter of the pipe. d and A are given by the nodal entities FD and FLA respectively. Only for circular pipes does $A = \pi d^2/4$ hold; FLA defaults to this value if not specified by the user.

It should be noted that the calculation of natural convection heat transfer coefficient (HTC) requires a characteristic dimension in the expression for Grashof number. For vertical tubes, this would normally be taken as the tube height, which is not available from the Esatan input data. Horizontal tubes require tube diameter as the characteristic dimension, which is available from the nodal data, so this has been used. The implication is, therefore, that the heat transfer coefficient calculation for natural convection systems is strictly applicable to horizontal tubes only. This approximation affects the calculation of local heat transfer coefficient only, and does not directly affect the hydraulic calculation of buoyancy-driven flows in loops, where the fluid motion is influenced by density gradients and relative nodal heights.

A key to the nomenclature employed is given at the end of this appendix.

K.2 Single-Phase Heat Transfer

The method used covers laminar, transition and turbulent flow, allowing for natural convection in the laminar region. The method is taken from a combination of references[5][6][11].

1. Calculate the Reynolds and Prandtl numbers

$$Re = \frac{md}{A\mu}$$

$$Pr = \frac{c_p\mu}{k}$$

with properties evaluated at the bulk fluid temperature.

2. For laminar flow ($Re \leq 2000$), calculate the Grashof number

$$Gr = \frac{\beta g d^3 \rho^2 \Delta T}{\mu^2}$$

with properties evaluated at the bulk fluid temperature. Calculate the laminar Nusselt number[5][6]

$$Nu_{\text{lam}} = 1.75(0.0083(GrPr)^{0.75})^{0.33}$$

If $Nu_{\text{lam}} < 3.66$, put $Nu_{\text{lam}} = 3.66$

3. For turbulent flow ($Re \geq 4000$), use Dittus-Boelter[2][18] to calculate the turbulent Nusselt number:

$$Nu_{turb} = \begin{cases} 0.024 Re^{0.8} Pr^{0.4} & \text{(heating)} \\ 0.026 Re^{0.8} Pr^{0.3} & \text{(cooling)} \end{cases}$$

4. For transition flow ($2000 < Re < 4000$), perform steps 2 and 3 and interpolate on Reynolds number to give transition Nusselt Number:

$$Nu_{tran} = Nu_{lam} + \left(\frac{Re - 2000}{2000}\right)(Nu_{turb} - Nu_{lam})$$

5. Calculate the heat transfer coefficient

$$h = Nu \frac{k}{d}$$

K.3 Two-Phase Heat Transfer

This method is used when the fluid is detected to be at saturation temperature. A correlation for forced convective boiling, allowing for post-dry-out conditions, is used for wall temperatures greater than saturation temperature, and a condensing correlation is used when the wall is below saturation temperature.

K.3.1 Two-Phase Boiling

For vapour quality $x \leq 0.7$, the method selected is that of Chen[12]. This is strictly applicable to upflow and downflow in tubes and annuli, although evidence does exist that satisfactory results have been found for horizontal flow, providing that stratification is not too severe. Such effects are unlikely in zero or low gravity.

The calculation procedure for Chen is as follows:-

1. Calculate the Reynolds number based on liquid properties

$$Re_l = \frac{m_l d}{A\mu_l}$$

2. Calculate the Prandtl number based on liquid properties

$$Pr_l = \frac{c_{pl}\mu_l}{k_l}$$

3. Calculate the convective heat transfer coefficient for liquid

$$h_l = 0.023\frac{k_l}{d}Re_l^{0.8}Pr_l^{0.4}$$

4. Calculate the inverse of the Lockhart-Martinelli parameter

$$\frac{1}{X_{tt}} = \left(\frac{m_v}{m_l}\right)^{0.9}\left(\frac{\rho_l}{\rho_v}\right)^{0.5}\left(\frac{\mu_v}{\mu_l}\right)^{0.1}$$

5. Calculate the convective boiling factor

$$F = 1 \qquad \left(\text{for } \frac{1}{X_{tt}} \leq 0.1\right)$$

$$F = 2.35\left(\frac{1}{X_{tt}} + 0.213\right)^{0.736} \qquad \left(\text{for } \frac{1}{X_{tt}} > 0.1\right)$$

6. Calculate the convective boiling contribution

$$h_{con} = Fh_l$$

7. Calculate the "two phase" Reynolds number

$$Re_{2\varphi} = Re_l F^{1.25}$$

8. Calculate the nucleate boiling suppression factor

$$S = \frac{1}{1 + 2.53 \times 10^{-6} Re_{2\varphi}^{1.17}}$$

9. Calculate the parameter

$$B = 0.00122 \frac{k_l^{0.79} c_{pl}^{0.45} \rho_l^{0.49} S}{\sigma^{0.5} \mu_l^{0.29} (h_{lv} \rho_v)^{0.24}}$$

10. Calculate the nucleate boiling contribution

$$h_{nuc} = B(T_w - T_s)^{0.24} (p_w - p_s)^{0.75}$$

11. Calculate the combined HTC

$$h = h_{con} + h_{nuc}$$

For vapour quality $x \geq 0.9$ a post-dry-out correlation due to Dougall and Rohsenow[15] is used, as follows:-

1. Calculate the single-phase vapour HTC, h_v, using the method of Section K.2.
2. Calculate the two-phase multiplier,

$$\Phi = x + (1 - x) \frac{\rho_v}{\rho_l}$$

3. Calculate the two-phase HTC,

$$h = \Phi h_v$$

For $0.7 < x < 0.9$, cubic spline interpolation is performed between the Chen and Dougall-Rohsenow correlations.

K.3.2 Subcooled Boiling

When the bulk temperature, T, is below saturation, but the wall temperature, T_w, is above it—i.e.

$$T < T_s < T_w$$

—then 'subcooled boiling' occurs. Vapour, at temperature T_s, is produced at the wall while the bulk of the fluid remains liquid.

The method used to calculate the heat transfer coefficient is such as to ensure continuity between the single- and two-phase regimes:-

1. Compute a single-phase HTC, $h_{1\varphi}$, as in Section K.2
2. Calculate a two-phase boiling HTC, $h_{2\varphi}$, as for Chen (Section K.3.1), except that:
   i) The liquid properties are evaluated at temperature T;
   ii) Re_l and h_l are used in place of $Re_{2\varphi}$ and h_{con}, (i.e. steps 4–7 are omitted)
3. Interpolate to obtain the final HTC:

$$h = \frac{(T_s - T)h_{1\varphi} + (T_w - T_s)h_{2\varphi}}{T_w - T}$$

K.3.3 Two-Phase Condensation

This method is based on one due to Boyko and Kruzhilin[13], appropriate to filmwise condensation in uniform channels under forced convection conditions.

The method is as follows:-

1. Calculate the single-phase liquid HTC, h_l, as in Section K.2
2. Calculate the two-phase multiplier,

$$\Phi = \sqrt{1 + x\left(\frac{\rho_l}{\rho_v} - 1\right)}$$

3. Calculate the two-phase HTC,

$$h = \Phi h_l$$

K.3.4 Superheated Condensation

When the bulk temperature, T, is above saturation, but the wall temperature, T_w, is below it—i.e.

$$T > T_s > T_w$$

—then 'superheated condensation' occurs. Liquid, at temperature T_s, is produced at the wall while the bulk of the fluid remains vapour.

The method used to calculate the heat transfer coefficient is such as to ensure continuity between the single- and two-phase regimes:-

1. Compute a single-phase HTC, $h_{1\varphi}$, as in Section K.2
2. Calculate a two-phase condensation HTC, $h_{2\varphi}$, as for Boyko-Kruzhilin (Section K.3.3)
3. Interpolate to obtain the final HTC:

$$h = \frac{(T_s - T_w)h_{1\varphi} + (T - T_s)h_{2\varphi}}{T - T_w}$$

K.4 Single-Phase Pressure Drop

If the fluid nodal entity FF is set to a non-zero value then the pressure drop associated with the pipe wall surface roughness is automatically calculated using the method defined within the publication by Colebrook and White[14]. When FF is set to zero no pressure drop due to pipe wall friction is calculated, effectively allowing users to input their own correlation via fitting loss conductors (GP).

The following outlines the method used internally to translate from the surface roughness (FF) into a pressure drop:-

1. Calculate Reynolds number

$$Re = \frac{md}{A\mu}$$

2. For $Re < 2000$ (laminar flow):

$$f = \frac{16}{Re}$$

3. For $Re > 4000$ (turbulent flow):

   i) Calculate parameter w

   $$w = \frac{\varepsilon}{d}\frac{Re}{18.57} + \log_{10}\frac{Re}{5.02}$$

   where ε is the pipe surface roughness.

   ii) Evaluate y from the table below

w	y
2.0	0.057
10.0	0.044
20.0	0.029
40.0	0.018
100.0	0.0087
1000.0	0.0013
100000.0	0.00008

   iii) Calculate

   $$f = \left(4\left(y + \log_{10}\frac{Re}{5.02} - \log_{10} w\right)\right)^{-2}$$

4. For $2000 < Re < 4000$ (transitional flow):

   A linear interpolation on Reynolds number is taken between the values of f calculated at $Re = 2000$ and $Re = 4000$ in steps 2 and 3 above.

5. The frictional pressure drop is then calculated by

$$\Delta p = 4f\frac{l}{d}\rho\frac{u^2}{2}$$

K.5 Two-Phase Pressure Drop

The approach adopted is that of Dukler et al[16] which proposes that a two-phase viscosity is adopted of the form

$$\mu = x\mu_v + (1-x)\mu_l .$$

This is then used to calculate a two-phase Reynolds number, which in turn is used with a conventional friction factor method appropriate to single-phase flow.

The two-phase friction factor is therefore calculated by the method described in Section K.4, but with the above modification to viscosity.

K.6 Nomenclature

K.6.1 Variables

A	flow area [m^2]
c_p	specific heat [J/kgK]
d	hydraulic diameter [m]
f	friction factor [-]
F	convective boiling factor [-]
Gr	Grashof number [-]
g	acceleration due to gravity [m/s^2]
h	heat transfer coefficient (HTC) [W/m^2K]
h_{lv}	latent heat of vapourisation [J/kg]
k	thermal conductivity [W/mK]
l	length [m]
m	mass flow rate [kg/s]
Nu	Nusselt number [-]
p	pressure [Pa]
Pr	Prandtl number [-]
Re	Reynolds number [-]
S	nucleate boiling suppression factor [-]
T	temperature [K]
u	velocity [m/s]
x	vapour quality [-]
X_{tt}	Lockhart-Martinelli parameter [-]
β	coefficient of thermal expansion [K^{-1}]
Δp	pressure drop [Pa]
ΔT	temperature difference (wall to fluid) [K]
σ	surface tension [kg/s^2]
ε	surface roughness [m]

ρ	density [kg/m^3]
μ	dynamic viscosity [kg/ms]

K.6.2 Subscripts

con	convection
i	transition
l	liquid
lam	laminar
nuc	nucleate
s	saturation
t	turbulent
v	vapour
w	wall
1φ	single-phase
2φ	two-phase

Appendix L: Modelling of Tee Piece Fitting Losses

The inclusion of standard fitting loss correlations into ESATAN/FHTS tee pieces is complicated by two main points.

1. ESATAN/FHTS software is coded under the assumption that a fitting loss conductance is associated with the mass flow rate in the same link as that for which the GP conductor is defined. Fitting loss correlations for tee pieces always take the combined mass flow at the junction for correlating the losses in each branch.

2. ESATAN/FHTS tee piece models contain a node precisely at the junction of the main and side branches in order to correctly model momentum flux terms. Experimental correlations consider only the three points at the inlets and exits to the tee piece.

The following describes how correlations for tee piece losses under the above conditions may be applied to ESATAN/FHTS models. The method is applicable to equal tees only (i.e. where all areas are the same), and also assumes that fluid density changes are negligible across the tee piece.

Consider the following ESATAN tee piece model:-

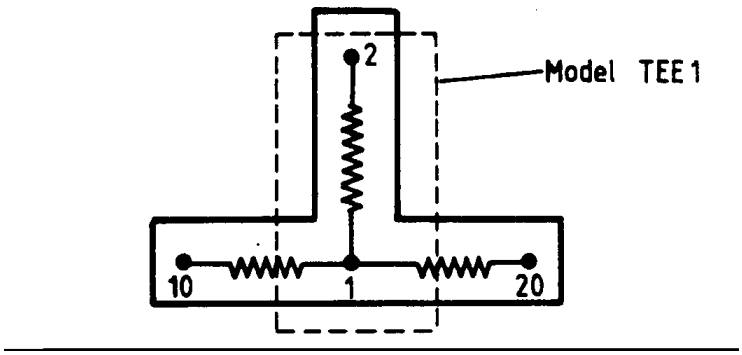


Figure L-1 ESATAN tee piece model

L.1 Case 1 - diverging flow

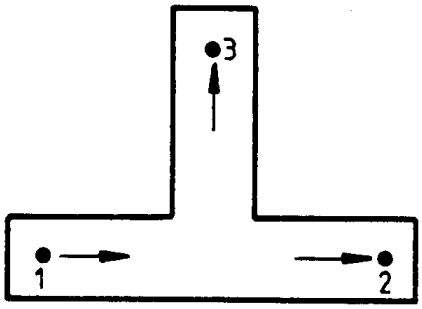


Figure L-2 Case 1 - diverging flow

Tee piece losses are usually correlated as loss factors k_{13} and k_{12}, such that:-

$$\Delta p_{13} = \frac{1}{2} k_{13} \rho u_1^2$$

$$\Delta p_{12} = \frac{1}{2} k_{12} \rho u_1^2$$

Then for the above Esatan model, the tee piece conductance would be:-

$$\mathrm{TGP} = \frac{1}{k_{13}} \left(\frac{\mathrm{M{:}TEE1{:}(1,2)}}{\mathrm{M(10,TEE1{:}1)}} \right)^2$$

and the main branch conductance:-

$$\mathrm{GP(TEE{:}1{:}1,20)} = \frac{1}{k_{12}} \left(\frac{\mathrm{M(TEE1{:}1,20)}}{\mathrm{M(10,TEE1{:}1)}} \right)^2$$

L.2 Case 2 - diverging flow

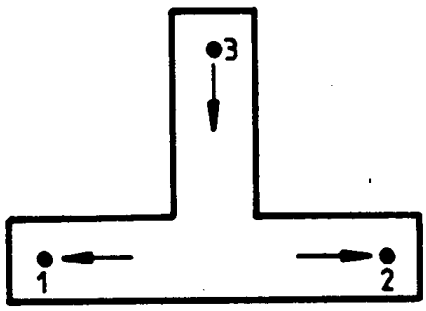


Figure L-3 Case 2 - diverging flow

Loss factors k_{13} and k_{23} would normally be specified. Then:-

TGP = 1.0 E10

$$\text{GP(TEE1:1,10)} = \frac{1}{k_{13}}\left(\frac{\text{M(TEE1:1,10)}}{\text{M:TEE1:(1,2)}}\right)^2$$

$$\text{GP(TEE1:1,20)} = \frac{1}{k_{23}}\left(\frac{\text{M(TEE1:1,20)}}{\text{M:TEE1:(1,2)}}\right)^2$$

L.3 Case 3 - converging flow

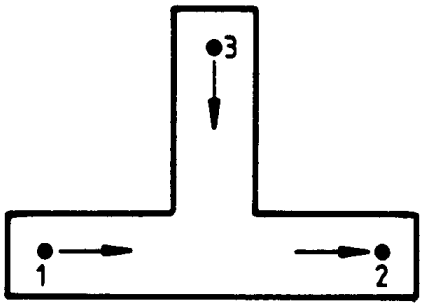


Figure L-4 Case 3 - converging flow

Loss factors k_{12} and k_{23} would be given. Then:-

$$\text{TGP} = \frac{1}{k_{23}}\left(\frac{\text{M:TEE1:(1,2)}}{\text{M(TEE1:1,20)}}\right)^2$$

$$\text{GP(10,TEE1:1)} = \frac{1}{k_{12}}\left(\frac{\text{M(10,TEE1:1)}}{\text{M(TEE1:1,20)}}\right)^2$$

L.4 Case 4 - converging flow

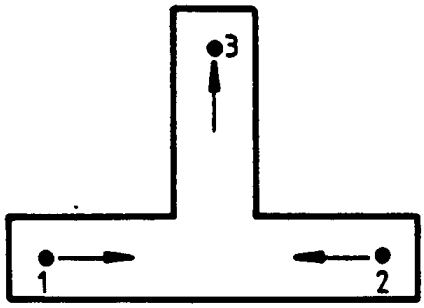


Figure L-5 Case 4 - converging flow

Loss factors k_{13} and k_{23} would be given. Then:-

TGP = 1.0 E10

$$\text{GP(10,TEE1:1)} = \frac{1}{k_{13}}\left(\frac{\text{M(10,TEE1:1)}}{\text{M:TEE1:(1,2)}}\right)^2$$

$$\text{GP(20,TEE1:1)} = \frac{1}{k_{23}}\left(\frac{\text{M(20,TEE1:1)}}{\text{M:TEE1:(1,2)}}\right)^2$$

Appendix M: Post-Processing

Post-processing of the thermal results can be performed using ThermNV or ESATAN-TMS Workbench. Information on these products is provided within separate User Manuals.

Note: The Data Extraction Utility, previously distributed with ESATAN, has now been withdrawn.

Text continues on next page

Appendix N: Material Properties Library

The aim of the material properties library is to provide a series of ESATAN $INCLUDE files which contain properties data for a given material. It is envisaged that over time users and companies will build up their own validated libraries of properties, and to this end the library supplied with ESATAN is illustrative only - whilst every care has been taken to ensure that the data is accurate, no guarantees are given. Users should read carefully the disclaimer at the top of each file.

Another significant aim of the supplied library is to establish a naming convention for material property files and material properties. This is discussed in detail below.

N.1 Nomenclature for include files

The data files are to named by the following convention:

Material name:	≤ 10 characters
Type indicator:	1 characters
Suffix of .inc:	4 Characters

The Material name and the type indicator will be separated by '_'

N.1.1 Material name

Pure substances shall be indicated as the Element symbol, or chemical formula as appropriate, optionally followed by the word "pure":

e.g. Al, Al_pure, H2O, H2O_pure, N2_pure, He_pure, R114

Near pure substances shall be indicated as the element symbol, or chemical formula as appropriate, followed by the purity:

e.g. Cu999

Proprietary names shall be as close as possible to literal (e.g. Delrin). Proprietary part numbers shall be as close as possible to the original, concatenated to an abbreviation of the manufacturer's name:

e.g. Sheldahl materials would be given as ShlG402800 etc.

Steel alloys shall be given where possible as the AISI or DIN type designation:

e.g. AISI301

Where no such designation exists, use the prefix StSt (Stainless Steel):

e.g. StSt17-4PH

When the form of the material is significant, this shall be appended to the material type as a single letter:

e.g. Al6061S (sheet), Al6061B (bar), Al6061T (tube) etc.

N.1.2 Type indicator

The type indicator can be one of five letters:

K	Constant (a constant value with no interpolation)
F	Fine interpolation
M	Medium interpolation
C	Coarse interpolation
S	Actual data computed by a sub-function

N.2 Nomenclature for arrays and constants

Within the data files the arrays and variables will be named as follows:

Material name:	≤ 10 characters
Property name:	≤ 4 characters
Type indicator:	1 character
Source index:	1 character

There shall be two underscore separators:

__ between Material name and Property name; and

__ between Property name and what follows.

N.2.1 Material Name

The materials will be named as above.

N.2.2 Property Name

The property names shall be as follows:

Thermal conductivity	TCon	
Electrical conductivity	ECon	
Thermal expansion coeff.	TExp	
Density	Rho	When unambiguously a solid (e.g. Al)
	RhoS	Solid
	RhoL	Saturated liquid
	RhoV	Saturated vapour
	RhoG	Gaseous

Heat capacity	Cp	When unambiguously a solid (e.g. Al)
	CpS	Solid
	CpL	Saturated liquid
	CpV	Saturated vapour
	CpG	Gaseous
Absorptivity	Alph	
Emissivity (diffuse)	Eps	
Alpha / Epsilon	AlEp	

Other properties will use their commonly known names abbreviated to 4 characters.

N.2.3 Type indicator

The type indicator will differ from the data file's indicator and will be as follows:

C Constant
I Interpolation
F Function

N.2.4 Source index

The source index shall be a lower case letter, starting with 'a'. In order to maintain consistency, the source index shall always be present, even if there is only one source. By segregating the data within a file normally only the index 'a' will be used.

N.3 Supplied library

The library currently contains data for the materials and their indicated properties as shown in Table 1, “Material Data Library”.

The following statement should be noted in respect of each file in this library:

> “ITP Engines UK Ltd. disclaim all warranties with regard to this file, including all implied warranties of merchantability and fitness, in no event shall ITP Engines UK Ltd. be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use of this file.”

MATERIAL	Cp	TCon	TExp	Alph	Eps	AlEp
AISI301	x	x	x	x	x	
AISI302		x	x			
AISI303		x	x			
AISI316	x	x	x	x	x	
AISI416		x	x			
AISI440C		x	x			
StSt17-4PH	x	x	x			
StSt17-7PH	x	x	x	x	x	
Alpure	x	x	x	x	x	
Al 6061			x	x	x	
Al 7075		x	x	x		
Tipure	x	x	x		x	
Invar	x	x	x			
Cupure	x	x	x	x	x	
Nylon		x	x			
Bead Blasted Al				x	x	
Black Anodisation				x	x	
Iridite Aluminium				x	x	
Alodine 1200 S				x	x	
Alodine - Alochrom				x	x	
CFRP				x	x	
Chemglaze Z306				x	x	
Electrodag 501				x	x	
Electrodag 503				x	x	
OSR-CMX-150um				x	x	
OSR-ITO-CMX-150um				x	x	
PCB 119				x	x	

Table N-1: Material Data Library

MATERIAL	Cp	TCon	TExp	Alph	Eps	AlEp
PCB Z				x	x	
PSG 120-FD				x	x	
SG11 FD				x	x	
VDA				x	x	
VD Au				x	x	
Sheldahl materials				x	x	x

Table N-1: Material Data Library

N.4 Example File

```
############################ C O N T E N T S ###################################
#
# Substance:     Stainless Steel AISI302
# Properties:    TCon, TExp
#
# $Id: listings/AISI302_C 1.2 2009/01/13 11:22:56GMT hsoft Released  $
#
########################### A T T R I B U T I O N ##############################
#
# Author:        Chris Lawrence
# Organisation:  EGT Ltd.
# Address:       MEC, Cambridge Road, Whetstone, Leicester LE8 6LH, England
#
# Version:       1.1      1995/08/04
#
# History:       1.1      First release by C. Lawrence
#
#################### C O P Y R I G H T  &  D I S C L A I M E R #################
#
# Copyright (c) 2000 ITP Engines UK Ltd.
#
# Permission to use, copy and distribute this file for any purpose is hereby
# granted provided that this copyright statement and this permission notice
# appear with the file and that the name of ITP Engines UK Ltd. not be used
# in advertising or publicity pertaining to distribution of the file without
# specific, written prior permission and that no charge is made for the
# distribution beyond the reasonable costs for the media, copying and
# distributing. ITP Engines UK Ltd. makes no representations about the
# suitability of this file for any purpose. It is provided "as is" without
# express or implied warranty.
#
# ITP ENGINES UK LTD. DISCLAIM ALL WARRANTIES WITH REGARD TO THIS FILE,
# INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO
# EVENT SHALL ITP ENGINES UK LTD. BE LIABLE FOR ANY SPECIAL, INDIRECT OR
# CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
# USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
# OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OF
# THIS FILE.
#
# Author:  A. Murray Davidson, ESA/ESTEC
#
################################################################################
################################################################################

$ARRAYS
$REAL
#



############################Thermal Conductivity###################################
#
# References:
#
#  [a]  "Thermophysical Properties Of Matter", Touloukian & Buyco,
#       ISI/Plenum 1970, Volume  1, Page 1161
#
```

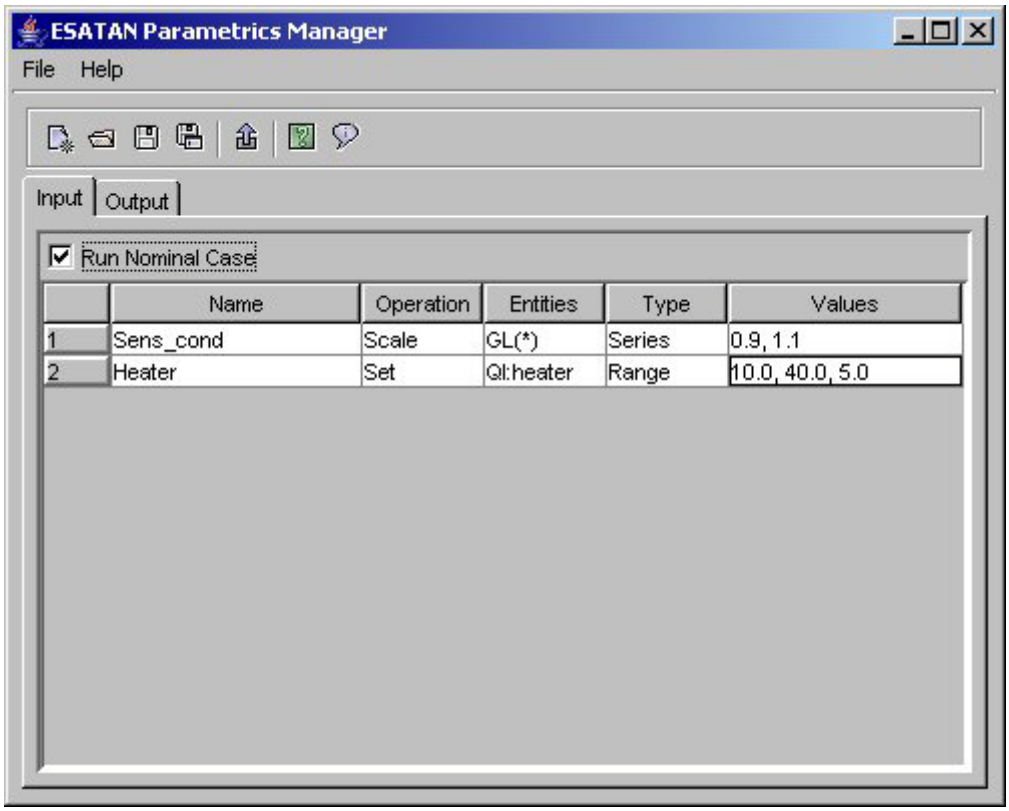
```
#       Units:                [W/m/K     ]
#       Temperature range:  323.20-1173.19K
#       Relative error:       +-2%
#       Absolute error:       +-0.55 [W/m/K     ]
#
AISI302_TCon_Ia   (52)=       323.2000,     7.3468,
 1.2300E+01 , 1.2647E+01 , 1.2994E+01 , 1.3337E+01 , 1.3649E+01 , 1.3961E+01 ,
1.4273E+01 ,
 1.4586E+01 , 1.4898E+01 , 1.5204E+01 , 1.5499E+01 , 1.5794E+01 , 1.6089E+01 ,
1.6384E+01 ,
 1.6679E+01 , 1.6984E+01 , 1.7296E+01 , 1.7608E+01 , 1.7920E+01 , 1.8233E+01 ,
1.8545E+01 ,
 1.8857E+01 , 1.9169E+01 , 1.9482E+01 , 1.9794E+01 , 2.0106E+01 , 2.0417E+01 ,
2.0712E+01 ,
 2.1007E+01 , 2.1302E+01 , 2.1597E+01 , 2.1892E+01 , 2.2192E+01 , 2.2504E+01 ,
2.2816E+01 ,
 2.3128E+01 , 2.3441E+01 , 2.3753E+01 , 2.4056E+01 , 2.4351E+01 , 2.4646E+01 ,
2.4941E+01 ,
 2.5236E+01 , 2.5531E+01 , 2.5839E+01 , 2.6151E+01 , 2.6463E+01 , 2.6775E+01 ,
2.7088E+01 ,
 2.7400E+01;


##########################Thermal Expansion Coefficient#############################
#
# References:
#
#  [a]  "Thermophysical Properties Of Matter", Touloukian & Buyco,
#       ISI/Plenum 1970, Volume 12, Page 1138
#
#       Units:                [m/m per K ]
#       Temperature range:   20.00-1399.99K
#       Relative error:       +-7%
#       Absolute error:       +-1E-6 [m/m per K ]
#
AISI302_TExp_Ia   (52)=         20.0000,     8.1630,
 9.8000E-06 , 1.0457E-05 , 1.0974E-05 , 1.1490E-05 , 1.2053E-05 , 1.2573E-05 ,
1.3024E-05 ,
 1.3509E-05 , 1.4015E-05 , 1.4427E-05 , 1.4829E-05 , 1.5250E-05 , 1.5671E-05 ,
1.6092E-05 ,
 1.6471E-05 , 1.6809E-05 , 1.7147E-05 , 1.7485E-05 , 1.7796E-05 , 1.8106E-05 ,
1.8416E-05 ,
 1.8703E-05 , 1.8956E-05 , 1.9210E-05 , 1.9463E-05 , 1.9669E-05 , 1.9866E-05 ,
2.0063E-05 ,
 2.0239E-05 , 2.0365E-05 , 2.0492E-05 , 2.0619E-05 , 2.0745E-05 , 2.0872E-05 ,
2.0999E-05 ,
 2.1109E-05 , 2.1151E-05 , 2.1193E-05 , 2.1235E-05 , 2.1278E-05 , 2.1320E-05 ,
2.1362E-05 ,
 2.1400E-05 , 2.1400E-05 , 2.1400E-05 , 2.1400E-05 , 2.1400E-05 , 2.1400E-05 ,
2.1400E-05 ,
 2.1400E-05;
```

Appendix O: Parametrics Manager

The ESATAN Parametrics Manager facilitates the definition of a parametric analysis whereby an ESATAN model can be repeatedly rerun, varying any number of parameters within the model. For example, a sensitivity analysis could be defined analysing the variation of a material property, running first the nominal case and then running the analysis between minimum and maximum values. To enhance the support of parametric analysis two looping constructs, “Series” and “Range”, have been introduced into the GUI. A parametric analysis is equivalent to a traditional ESATAN $PARAMETERS model defined through the external parameter file (<*model*>.PAR).

The user interface supports a subset of the ESATAN $PARAMETERS language; for example all cases are assumed to be of type “INITIAL”. Underlying the user interface is a specifically design XML language, this language is fully compatible with the ESATAN $PARAMETERS language. To perform the ESATAN analysis an export facility is provided to generate the traditional ESATAN <*model*>.PAR file read by ESATAN.

A batch-mode command-line interface is also included allowing conversion of the XML definition to the ESATAN $PARAMETER format.



Appendix P: References

1. W.M. Kays and A.C. London, "Compact Heat Exchangers", McGraw Hill (1964)
2. F.W. Dittus and L.M.K. Boelter, Univ. Calif. Pubn. Engg., v2, p443(1930)
3. "Baffled shell and tube heat exchangers: flow distribution, pressure drop, and heat transfer coefficients on the shell side", Item No 83038, Engineering Sciences Data Unit, London (1983)
4. "Convective heat transfer during cross-flow of fluids over plain tube banks", Engineering Sciences Data Unit, London(1973)
5. S.W. Churchill, "Comprehensive correlating equations for heat, mass, and momentum transfer in fully developed flow in smooth tubes", Ind. Eng. Chem. Fundls., 109-116 (1977)
6. "Forced convection heat transfer in circular tubes. Part II: Data for laminar and transitional flow including free convection effects", Item No 68006, Engineering Data Sciences Unit, London (1968)
7. E. Fried and I.E Idelchik, “Flow resistance”, Hemisphere Publishing Corporation.
8. B.S. Massey, "Mechanics of Fluids", 4th edition.
9. "Retran 02 - A program for transient thermal-hydraulic analysis of complex fluid flow systems", Vol. 1, Theory and numerics, EPRI, NP-1850-CCMa (1984)
10. Dong Wook Jerng & Hee Cheon No, “New Computational Method with a Fully Implicit Scheme for a Real Time Accident Simulator”, Nuclear Eng. & Design 99, 101-107, North Holland (1987)
11. "Forced convection heat transfer in circular tubes. Part I: Correlations for fully developed turbulent flow - their scope and limitations", Item No 67016, Engineering Sciences Data Unit, London (1976).
12. "A correlation for boiling heat transfer to saturated fluids in convective flow", Ind. and Engng. Chem - Process and Design Dev., Vol 5, no 3, p 322 (1966)
13. L.D. Boyko and G.N. Kruzhilin, "Heat transfer and hydraulic resistances during condensation of steam in a horizontal tube and in a bundle of tubes", Int. J. Heat Mass Transfer, Vol 10, pp 361-373 (1967)
14. C.F. Colebrook and C.M. White, "Turbulent flow in pipes, with particular reference to the transient regions between the smooth and rough pipe laws", J. Inst. Civ. Engrs, vol. 1, pp 133-156 (1939)
15. R. S. Dougall and W. M. Rohsenow, "Flow Boiling on the inside of Vertical Tubes with upward flow of the Fluid at Low Qualities", Heat Transfer Lab. Rep. 9079-26, Massachusetts Institute of Technology, Cambridge, Mass. (1963)
16. A.E. Dukler, M. Wicks and R.G. Cleaveland, "Frictional pressure drop in a ‘two phase’ flow: B. An approach through similar analysis", A.I.Che.J. pp 44-51 (1964) .

17. S De Palo, ‘Lessons learned from FHTS used for Columbus thermal design’, 12th Thermal & ECLS Software Workshop, ESA (1998).
18. Warren M. Rohsenow, James P. Hartnett and Young I. Cho, “Handbook of Heat Transfer” (3rd Edition), Mcgraw-Hill (1998)
19. J.M. Delhaye, M. Giot, M.L. Riethmuller (Eds.), "Thermohydraulics of Two-Phase Systems for Industrial Design and Nuclear Engineering", Hemisphere (1981)
20. A. E. Bergles, J.G. Collier, J.M. Delhaye, G.F. Hewitt and F. Mayinger, “Two-Phase Flow and Heat Transfer in the Power and Process Industries”, Hemisphere (1981)
21. “Radiative Heat Transfer”, 2nd Edition, Michael F. Modest, Academic Press, 2003