



SERCO ASSURANCE

The SOLOMON Event Tree Program - A User Guide



SERCO ASSURANCE

G J Roberts
G F Holford
C Maidment

October 1999

The information which this report contains is accurate to the best of the knowledge and belief of Serco, but neither Serco nor any person acting on behalf of Serco make any warranty or representation expressed or implied with respect to the use of the computer program described, nor assume any liabilities with respect to the use of, or with respect to any damages which may result from the use of information, method or data disclosed in this report.

Executive Summary

The SOLOMON computer program has been developed by Serco Assurance experts to construct and evaluate large Event Trees. The code was originally developed for safety-critical applications, such as a Level 2 Probabilistic Safety Analysis of a Nuclear Power Plant, but its inherent flexibility makes it ideal for a wide range of applications. Because the number of branches in an event tree increases exponentially with the number of top events, or decision points, special features are required to handle all but the simplest of event trees. The special features included in SOLOMON that make it suitable for handling the largest of event trees include:

1. Branching probabilities are expressed algebraically, one per decision point instead of one per branch. This greatly reduces the amount of input data required as the number of branches increases;
2. Decision points, or nodes, can be grouped into 'supernodes' making the display of an event tree much simpler.

SOLOMON has been designed to run on an IBM compatible PC running under the Windows 9x (or later) operating system (a 16-bit version is available for systems running under Windows 3.x). It includes a Graphical User Interface to assist the user in constructing event trees and analysing the results of SOLOMON calculations. The code uses a combination of key words and C programming language to construct an Event Tree model. The functionality that this provides is first discussed, together with input examples, before a description of the GUI and its features is presented.

Contents

Executive Summary	iv
1 Introduction	1
2 Probabilistic Safety Assessment Methodology	1
3 SOLOMON Features	2
3.1 CONVENTIONAL FEATURES	2
3.1.1 Input File	2
3.1.2 Prior Path Dependency	2
3.1.3 End Categories	3
3.1.4 Jumping over Nodes	3
3.2 NUMERICAL CALCULATIONS	4
3.2.1 Variables	4
3.2.2 Calculations	4
3.2.3 User-Defined Functions	5
3.2.4 Interpolation	5
3.2.5 Conditional Expressions	5
3.3 PRODUCING GRAPHICAL RESULTS	5
3.3.1 Supernodes	6
3.4 UNCERTAINTY AND SENSITIVITY ANALYSIS	7
3.4.1 Importance Measure	7
3.4.2 Uncertainty Analysis - Latin Hypercube Sampling	7
4 Running SOLOMON	7
5 Setting up the Input File	9
5.1 GENERAL COMMANDS	10
5.1.1 Comments	10
5.1.2 Title	11
5.1.3 System States	11
5.1.4 Number of Latin Hypercube Samples	11
5.1.5 Showpaths Frequency	11
5.1.6 Cutoff	12
5.2 NODE INFORMATION	12
5.2.1 Defining a node	12
5.2.1.1 Multi-Path Branches and Branch Names	12
5.2.2 Straightlining, or Jumping	12

5.2.3	Conditional Node Probabilities	13
5.3	DEFINING NUMERICAL FUNCTIONS	13
5.4	SETTING PARAMETERS	14
5.4.1	Arithmetic Expressions	15
5.5	DEFINING AND USING PARAMETERS	16
5.5.1	Real parameters	16
5.5.2	Conditional parameters	16
5.5.3	Enumerated parameters	16
5.5.4	Distributed parameters	16
5.5.5	Interpolated parameters	17
5.5.6	Show Parameters	18
5.6	SUPERNODES	18
5.6.1	Defining Supernodes	19
5.6.2	Drawing the Event Trees	19
5.6.3	Supernode Tree Start Condition	20
5.7	END CATEGORIES	20
5.7.1	Nocheck	21
5.7.2	Endcat parameters	21
5.8	END OF THE DATA FILE	21
5.9	CONDITIONAL EXPRESSIONS	21
5.9.1	If-then-else	22
5.10	EXAMPLE	22
6	The Graphical User Interface	26
6.1	INTRODUCTION	26
6.1.1	Limitations of the GUI	27
6.2	CREATING A NEW SOLOMON INPUT FILE	28
6.2.1	Starting a new file	28
6.2.2	Adding information to the header block using the header creation dialog	28
6.2.3	Inserting a new SOLOMON object	29
6.2.4	Creating a new straight node	30
6.2.5	Creating a new branching node	31
6.2.5.1	Defining branch probabilities	32
6.2.6	Defining Supernode Trees	33
6.2.7	Defining Supernodes	34
6.2.8	Defining Endcats	35
6.2.9	Endcategory default	37
6.3	MODIFYING AN EXISTING FILE	38
6.3.1	Features for modifying files	38
6.3.2	Cut/Copy/Paste object	38
6.3.3	Edit SOLOMON Object Dialog	38
6.3.4	Activate/Deactivate Object	40
6.4	SUB-DIALOGS	41
6.4.1	List parameters	41
6.4.2	Define Parameters	42
6.4.2.1	Define distributed parameter	43
6.4.3	Define Function	44

6.4.4	Set parameter value	45
6.4.5	Define Run Sequence	46
6.4.6	Show parameters	47
6.4.7	Correlate	48
6.5	SUBMITTING CALCULATIONS AND VIEWING RESULTS	49
6.5.1	Run SOLOMON	49
6.5.1.1	The current working directory	49
6.5.1.2	Go to line	49
6.5.2	The Supernode Viewer	50
6.5.2.1	Starting the supernode viewer	50
6.5.2.2	Selecting a file	50
6.5.2.3	Selecting a tree	50
6.5.2.4	Page options	51
6.5.2.5	View options	52
6.5.2.6	Exporting and printing	53
6.5.2.7	Finishing up	53
6.5.3	The PDF/CDF Viewer	53
6.5.3.1	Starting the PDF/CDF viewer	53
6.5.3.2	Viewing a variable or endcat	54
6.5.3.3	Exporting and printing	54
6.5.3.4	Finishing up	54
6.5.3.5	Limitations of PDF/CDF viewer	54
6.5.4	The Output Viewer	55
6.5.4.1	Starting the Output viewer	55
6.5.4.2	Finishing up	55
6.6	CUSTOMISING THE SOLOMON GUI	55
6.6.1	Supplied file	55
6.7	DESIGN LIMITATIONS	57
6.7.1	Functional limitations	57
6.7.2	Limitation hardwired within the code	57
7	References	58

List of Figures

- Figure 1:** An Example of a Supernode Tree
- Figure 2:** An Example of End Categories
- Figure 3:** Use of a straight node, NODE B, in an event tree
- Figure 4:** An Example of a Supernode Tree
- Figure 5:** Example Cumulative Distribution Function (CDF) for the Uncertainty in a SOLOMON End Category
- Figure 6:** A Typical Screen After Loading a Small File
- Figure 7:** Defining General Parameters
- Figure 8:** Defining a Straight Node
- Figure 9:** Defining a Branching Node
- Figure 10:** Defining Branch Probabilities
- Figure 11:** Defining a Supernode Tree
- Figure 12:** Defining a Supernode
- Figure 13:** Defining an End Category
- Figure 14:** The Default End Category
- Figure 15:** Editing a SOLOMON Object
- Figure 16:** Listing of Available Parameters
- Figure 17:** Defining a Parameter
- Figure 18:** Defining a Distributed Parameter
- Figure 19:** Defining a Function
- Figure 20:** Setting Parameter Values
- Figure 21:** Defining the Run Sequence
- Figure 22:** Defining Show Parameters
- Figure 23:** Defining Correlation Parameters
- Figure 24:** The 'Go To' Dialog Box

1 Introduction

The SOLOMON Event Tree code was originally developed for safety critical applications, such as a Level 2 Probabilistic Safety Assessment (PSA) of a Nuclear Power Plant (NPP), but its inherent flexibility makes it ideal for a wide range of applications. Event trees are used throughout the PSA process, and many programs are available to calculate them, but their use in a Level 2 PSA presents some particular problems because of the size of the trees that are required. The complexity and size of an event tree increases exponentially with the number of top events. For a Level 2 PSA of a NPP, there is a trend towards using event trees with many tens of top events to describe an accident progression.

To illustrate the complexity involved, consider an event tree with 6 top events - this could contain up to 64 paths and could be presented on a single sheet of paper. A tree with 12 top events could contain up to 4096 paths - this would need about 64 sheets of paper to display. A tree with 30 top events could contain over a billion paths.

In designing SOLOMON the following questions were asked:

- How can the tree be examined to check that it is set up correctly?
- How can branching probabilities be set, when there may be millions of branching points?
- How can we make it easier to change the tree to include new data?
- How can the results be displayed in reports?
- How can uncertainties and sensitivities be analysed?

2 Probabilistic Safety Assessment Methodology

The event tree needs the probability of each of its top event nodes for it to be quantified. Basic information is generally available in the form of measured and/or calculated values of physical/financial quantities. A link must be made between the event probabilities and the physical/financial information, but this process is not straightforward. In some cases correlations relating event probabilities to physical/financial parameters can be established from measured data and/or mathematical models. In other cases, expert judgement is used to correlate event probabilities with the physical/financial data. This can be an expensive process, particularly if it has to be repeated in the light of new information. In a large tree it may be difficult to consider all the dependencies between the top events.

With SOLOMON, the aim is to systematise and partially automate this process. Expert judgement may still be required, but it is included in the event tree. To achieve this, SOLOMON allows calculations to be performed in the event tree to carry out two tasks:

- i. evaluate simple models that determine the physical/financial state of the system;
- ii. evaluate functions that determine the probability of an event occurring for a given physical/financial state of the system.

The advantages of this approach are:

- It reduces the work needed in converting physical/financial data to event probabilities.
- It is easy to make changes.
- It is systematic and easy to examine - modelling assumptions are contained in a few functions rather than thousands of individual event probabilities.

3 SOLOMON Features

3.1 CONVENTIONAL FEATURES

3.1.1 Input File

The SOLOMON event tree is defined using a keyword-driven input file. This requires the analyst to learn a few keywords, but provides a great deal of flexibility in setting up the event tree. The basic working unit is the event or node which represents a multiple-choice question - a probability of the node following a particular branch can be set. The nodes of the event tree are defined in the order that they occur.

3.1.2 Prior Path Dependency

A list of nodes can be defined, each one with a fixed probability or a probability that depends on the result of previous events. In the example shown in Figure 1 below, there are three nodes, A1, B1 and C1 where the probability of node A1 following branch 1 is fixed at 0.001, while the probabilities of nodes B1 and C1 depend on the outcome of the previous nodes. This feature is called *prior path dependency*.

Title : Solomon 2.20 Example input file
 Run : run1
 Tree : tree1

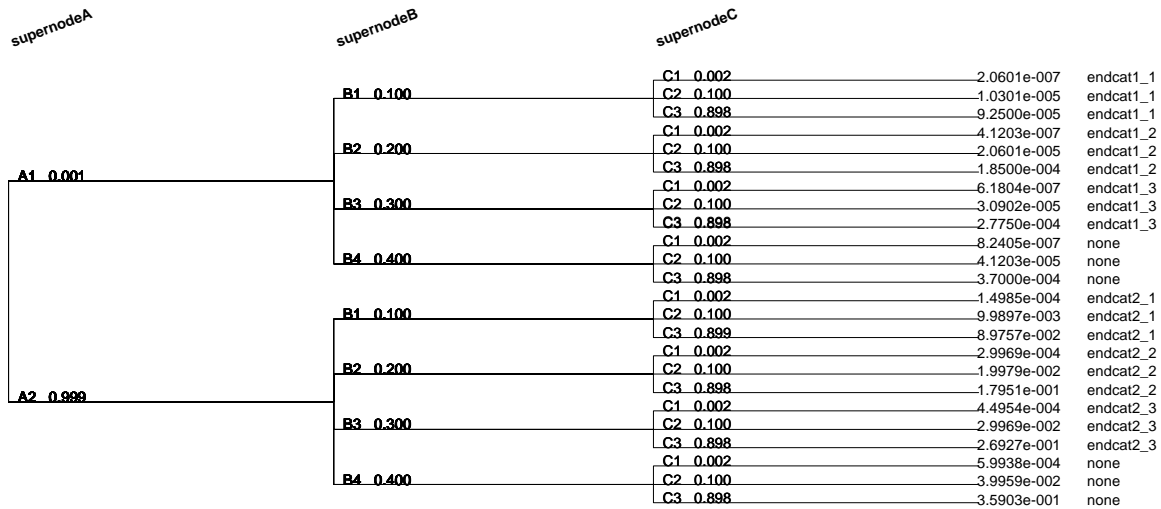


Figure 1: An Example of a Supernode Tree

3.1.3 End Categories

The many paths through the event tree can be allocated to a smaller number of End Categories. This allows a range of scenarios with similar features to be grouped together. An example is given in Figure 2 below, where the eight paths through the event tree are allocated to End Categories ONE or TWO. SOLOMON presents the total probability of each End Category.

Title : No
 Run : 1

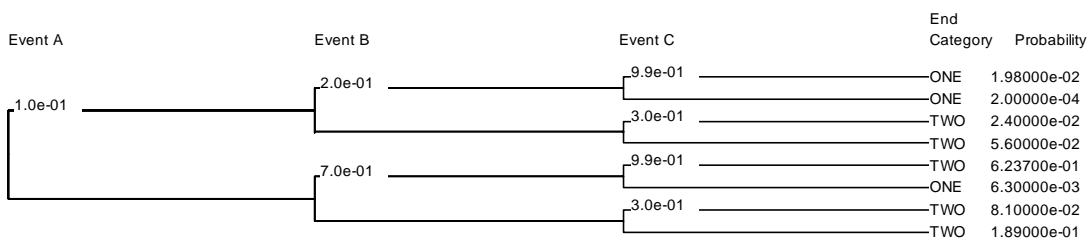


Figure 2: An Example of End Categories

3.1.4 Jumping over Nodes

For some paths, it is often necessary to jump over nodes. This can occur, for instance, in the case of a Level 2 PSA of a NPP when containment failure is predicted early in a plant accident sequence and later actions have no impact on the end state. SOLOMON supports jumping, or *straightlining*, with prior path dependencies.

3.2 NUMERICAL CALCULATIONS

SOLOMON allows calculations of any complexity to be carried out within the event tree. For this feature, SOLOMON provides a new type of node called a *straight node*, which can be used to include system models to represent any process that may impact on the probability of an event. These nodes can be placed at any point the tree and are treated as non-branching events in the tree. This allows the calculations to include prior path dependencies in the same way as the conventional, branching nodes. An example is given in Figure 3 below, where a calculation is performed in straight node Node B (here the variable x is simply set depending on the outcome of Node A, but this could be any calculation) and the result is used to set the probability in Node C.

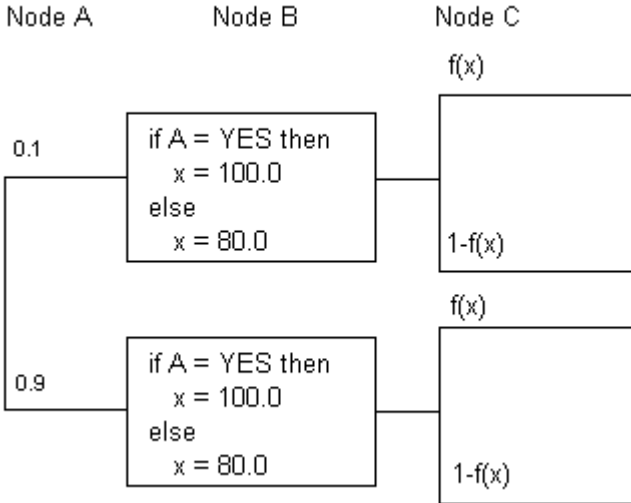


Figure 3: Use of a straight node, NODE B, in an event tree

The following features are supported in straight nodes:

3.2.1 Variables

User-defined variables can be defined to be *real*, *conditional* or *enumerated*. Real variables are treated as double precision floating point. Enumerated variables can take values from a user-defined list of names, for example the variable PRESSURE could be defined as LOW, MEDIUM or HIGH. This is useful for setting switches in the event tree without using meaningless index numbers. Conditional variables are similar to logical variables in FORTRAN. Once defined, variables can be set to constants or the results of calculations and used in further calculations as needed.

A variable can also be defined as being *randomly distributed*, in which case its value is defined by a probability density function. This is for use with uncertainty analysis (see below).

3.2.2 Calculations

Standard calculation operators are +, -, * and /. In addition, a number of standard numeric functions are included:

pow(x,y)	Raise x to power y
log(x)	Natural log of x
exp(x)	Exponential of x

3.2.3 User-Defined Functions

It is possible to define numeric or conditional functions for repeated use within the event tree. This can be used to calculate a physical model.

In SOLOMON, user-defined functions are entered using “C” code, which links with the user-defined variables. This gives the user a great deal of flexibility, as all the features of the “C” programming language are available.

3.2.4 Interpolation

A function of one variable can be supplied to the event tree in the form of a set of x,y values, and used to interpolate values within the event tree. This is useful for using experimental data to set probability values.

3.2.5 Conditional Expressions

The *if-then-else-elseif* construction is supported, with conditional expressions of any complexity. The available conditional operators are:

=	Equal to
<>	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
not	Not
and	And
or	Or

Prior path dependencies can be included in conditional expressions.

3.3 PRODUCING GRAPHICAL RESULTS

SOLOMON has a unique way of producing graphical output from the event tree which allows the analyst to condense the entire event tree or to focus attention on selected nodes.

3.3.1 Supernodes

Using the nodes of the existing event tree, a new set of nodes can be defined which are logical combinations of some or all of the original nodes. These new nodes are called *supernodes*. SOLOMON produces a new event tree (the supernode tree) which shows the individual supernode probabilities as well as the path probabilities. This is a very flexible system which can be used in a number of ways:

- Showing all the paths leading from a single point in the tree. This is useful for checking that the tree has been set up correctly, as individual node probabilities can be compared with the event tree definition.
- Showing the results from one part of the tree. In this case only the nodes of interest are included in the supernodes. This is useful for diagnosis and illustrating the operation of part of the tree.
- Displaying End Category logic. A supernode tree can be constructed which mirrors the logic used to group the event tree paths into end categories. This is useful in discussing the choice of End Categories.
- Reporting results in a Simplified Event Tree (SET). With the correct choice of supernodes, a large tree can be condensed into a SET which displays important features of the original event tree. A number of SETs may be needed to illustrate all the main features.

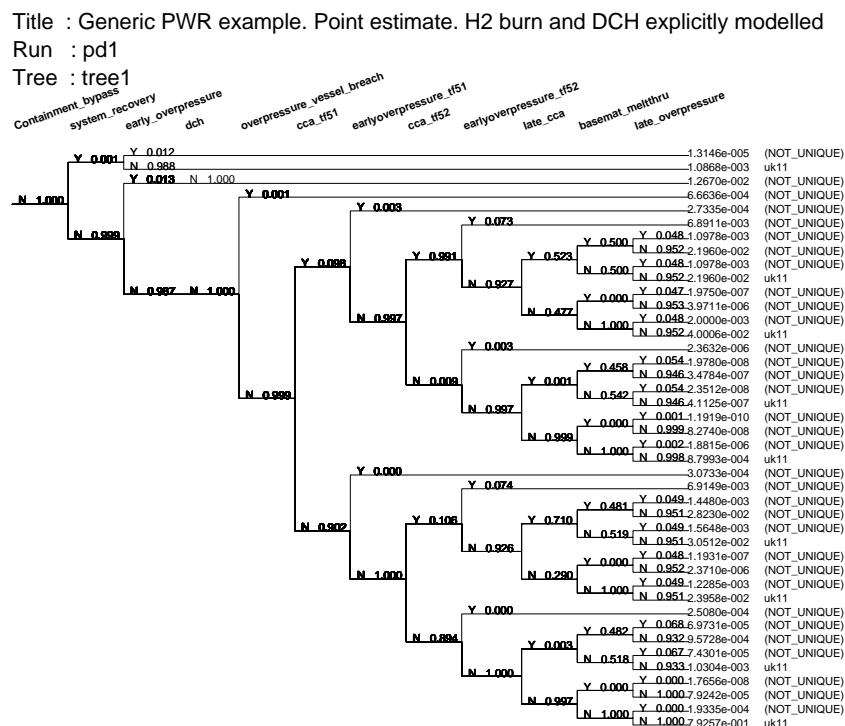


Figure 4: An Example of a Supernode Tree

An example of a supernode tree is shown in Figure 4. This was the result of condensing a 30 node event tree into 12 supernodes. (Note that in this figure, as before, the *up* branch

represents YES, the *down* branch represents NO. Each node is labelled with the probability of its outcome being YES. If only one branch of a node is present it is labelled with Y for YES, or N for NO, as appropriate).

3.4 UNCERTAINTY AND SENSITIVITY ANALYSIS

SOLOMON provides a number of features for performing uncertainty and sensitivity analysis.

3.4.1 Importance Measure

A set of importance measures is automatically calculated which give an estimate of the sensitivity of each End Category to each node probability.

3.4.2 Uncertainty Analysis - Latin Hypercube Sampling

Variables used in calculating node probabilities can be defined as being randomly distributed. Normal, lognormal, uniform and triangular functions are available. SOLOMON uses Latin hypercube sampling (an enhanced Monte Carlo technique) to calculate the uncertainty in each End Category resulting from uncertainty in the input. The result of this is an uncertainty distribution for each End Category. An example is given in the cumulative distribution function in Figure 5. SOLOMON also calculates statistics for the distributions: mean, median, variance, standard deviation and 95% confidence limits.

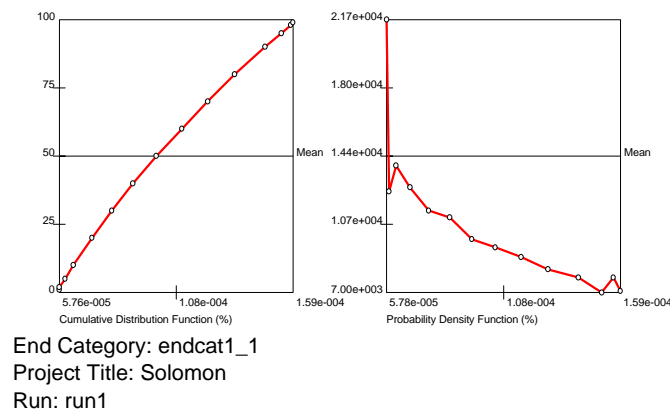


Figure 5: Example Cumulative Distribution Function (CDF) for the Uncertainty in a SOLOMON End Category

4 Running SOLOMON

SOLOMON consists of a suite of programs. A Graphical User Interface (GUI) is provided to run the suite automatically. When SOLOMON is installed, the GUI allows the user to select and edit an input file, launch a SOLOMON calculation, and view the results. The following

section details how to set up the input file. The output is printed to a number of files which are described below.

Filename	Description
<code><run name>.out</code>	This text file contains the main output including the endcat output - either an error message if the input file is incorrect, or details of the probability of each end category. Each <code><run name></code> refers to a system state set up in the input file. This output can either be used as printed text output or processed using the PDF/CDF viewer.
<code><run name>.sh1</code>	This text file contains the main output of each variable (corresponding to a 'show' command in the input file) for each Latin Hypercube sample. This output can either be used as printed text output or processed using the PDF/CDF viewer
<code><run name>.sh2</code>	This text file contains statistical data (mean, median, 95% confidence limits, standard deviation) for each variable (corresponding to a 'show' command in the input file) in CSV format.
<code><run name>.sh3</code>	This text file contains the cumulative distribution function values of each variable (corresponding to a 'show' command in the input file) in CSV format.
<code><run name>.sh4</code>	This text file contains the probability density function values of each variable (corresponding to a 'show' command in the input file) in CSV format.
<code><run name>.sh5</code>	This text file contains the minimum, maximum and weighted mean values of each variable (corresponding to a 'show' command in the input file).
<code><run name>.end</code>	This text file contains the point value of each end category for each Latin Hypercube sample.
<code><run name>.snd</code>	This text file contains the supernode descriptions used by the supernode viewer.
<code><run name>.top</code>	These text files contains a list of the top events and the branch names for each node.
<code><run name>.pds</code>	This text file contains the parameter statements for the end categories and is used to define run sequences in later calculations.
<code><run name>.ecs</code>	These text files contain the comma delimited results of the end category probabilities.
<code>control.log</code>	This text file contains details of the SOLOMON run such as compilation messages and program warnings. If an error has occurred in any of the SOLOMON programs, or with the system, the file will contain relevant error messages.
<code>generate.log</code> <code>compile.log</code> <code>run.log</code> <code>show.log</code>	These text files contain details of individual steps of the SOLOMON run such as compilation messages and program warnings. If an error has occurred in any of the SOLOMON programs, or with the system, these files will contain relevant error messages.

The GUI provides the following features:

1. It provides a convenient way of running the SOLOMON suite: it provides dialogs to select input files and a menu for running the Event Tree programs and the supernode viewer.
2. It helps the user to manage the files that SOLOMON generates: these include the log files from each step, which can be browsed directly, the supernode descriptions, which can be browsed using the supernode viewer; and the comma-separated files, which can be fed into spreadsheets or graphical display programs.

3. It helps the user to set up input files: the input file is broken down into small units and lists of available parameters can be displayed. The GUI also helps the user to locate line numbers whenever errors are found.

5 Setting up the Input File

The user must provide instructions that describe the event tree to SOLOMON in a text file. The GUI provides some help in performing this task, mainly by dividing up the input file into 'objects' and by providing lists of available parameters which can be pasted into the user input file. Each SOLOMON Object is represented in a coloured rectangle in the left side of the window, and the text of the current object is displayed in the right side of the window. Whole objects can be manipulated using insert, delete, and copy/paste commands, while the GUI prevents objects being moved to invalid locations within the input file. The object text may be edited directly or through the object editor which provides a number of additional features to help the user to edit the text within the objects. The GUI also allows the user to define objects as 'active' or 'inactive' so that, for example, a user file can have alternative versions of objects.

When SOLOMON is run, the input file is checked; any errors are written to a log file. If errors are found, the program is halted, and a message is displayed to the user. The user can then inspect the log file, which gives a list of the problems found, with the line number where the problem is probably caused. The user can then go to the line indicated using an option to locate the requested line and correct the error. SOLOMON may detect more than one error in a single run.

If no errors are found, the event tree will be calculated and the results analysed.

The input file is keyword driven; each line represents a new command and keywords define the command. The structure of the input file is divided into SOLOMON Objects as follows:

1. Header information (title, specifying system states, number of samples etc.) displayed dark blue.
2. Node information (specifying nodes 1..n in order). Straight nodes are displayed yellow, branching nodes are displayed green.
3. Supernode information (specifying supernodes 1..m in order). Supernodes can be organised into multiple "supernode trees". Supernode trees are delimited by grey-blue regions, supernodes themselves are light blue.
4. End category information. The default end category is displayed orange, other end categories are pink.
5. End-of-data marker, displayed as white.

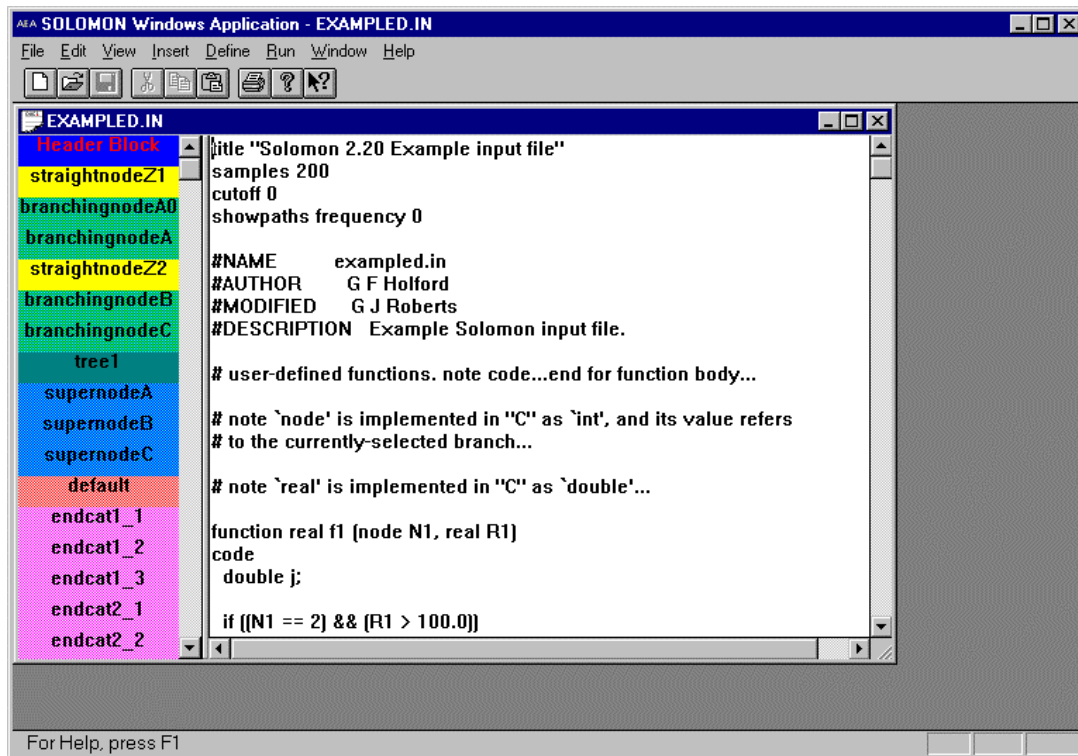


Figure 6: A Typical Screen After Loading a Small File

The title of each object is displayed as text within the list of coloured rectangles. The current object has red text. The other objects have black text if they are active, grey text if they are inactive.

This section details the commands available in the input file, with some examples.

5.1 GENERAL COMMANDS

In this section, the following conventions have been used to describe the syntax in the SOLOMON input:

1. example lines of input are shown in a fixed-width font, as in `samples 150`
2. optional text is shown in square brackets, as in `[global]`
3. user-supplied text is shown in angle brackets; for example `samples <n>` means that the user must supply a value for `n`.
4. Alternatives are separated by a vertical bar `|`. For example, `real | cond` means that the user must enter either 'real' or 'cond'.

5.1.1 Comments

Comment lines must begin with the character '#'. **Note that ### should not be used to comment a line as this is reserved for use by SOLOMON.**

5.1.2 Title

A one-line title, delimited by " " can be specified with the command

```
title "<title up to 500 characters long>"
```

for example

```
title "PWR Event Tree for Plant Damage State A15."
```

5.1.3 System States

System states can be calculated by including conditional expressions based on *enumerated* parameters (i.e. ones that take discrete values). The syntax is

```
run <run name 1> (<parameter> <value>, ... <parameter> <value>)  
run <run name 2> (<parameter> <value>, ... <parameter> <value>)  
..  
run <run name N> (<parameter> <value>, ... <parameter> <value>)
```

where the <run name *n*> is the name of the *n*th system state and the parameter/value list shows what values are used. For example, the following instructions define three system states (referred to as plant damage states in this instance) PD1, PD2 and PD3, which are characterised by the state of the containment sprays (on or off) and whether containment isolation has occurred,

```
define enum sprays    (sprays_on, sprays_off)  
define enum isolation (iso_on, iso_off)  
run PD1 (sprays sprays_off, isolation iso_on)  
run PD2 (sprays sprays_on,  isolation iso_on)  
run PD3 (sprays sprays_off, isolation iso_off)
```

SOLOMON automatically calculates the event tree for each system state, with the parameters set as indicated.

5.1.4 Number of Latin Hypercube Samples

The number of samples to be produced by the Latin Hypercube Sampling Program can be specified by the optional command

```
samples <n>
```

for example

```
samples 150
```

The default is one sample. The number of samples chosen must either be one or greater than the number of distributed variables defined (this restriction is imposed by the Latin Hypercube Sampling program).

5.1.5 Showpaths Frequency

The number of top paths displayed can be limited by the use of the command

```
showpaths frequency <x>
```

which causes only the paths with a frequency greater than <x> to be displayed in SOLOMON output.

5.1.6 Cutoff

This keyword is used in the header block to define a cutoff frequency at or below which a path is ignored. Default = 0.0 (i.e. show all non-zero paths).

5.2 NODE INFORMATION

5.2.1 Defining a node

Nodes can be of type 'branching' or 'straight'. The operations that can be done in a node depend on its type. Valid constructions to define a node are one of the following

```
node <name> type straight
```

which defines a straight node or

```
node <name> type branching
```

which defines a branching node with the probability of success as yet undefined, or

```
node <name> type branching prob <value>
```

(where the value can be a parameter or constant) which defines a branching node with a given probability of success. Some examples are

```
node A0 type straight
node Z20 type branching
node containment_bypass type branching prob 0.1
node early_overpressure type branching prob P12
```

5.2.1.1 Multi-Path Branches and Branch Names

Branching nodes may have more than two branches. In this case, the branches are named, following the keyword "branch" as in the following example:

```
node multiple_branch type branching
branch branch1
branch branch2
branch branch3
```

Note that a binary branch is also a multi-path branch. If branch names are not given, the 'up' branch has the name 'Y' and the 'down' branch has the name 'N'.

5.2.2 Straightlining, or Jumping

This operation can only be done from a branching node. After it has branched at this node, the tree can be made to jump straight to another node. The instruction can be unconditional or conditional. To jump to another node on success at this one, use

```
jumpup [cond (<conditional expression>)] to [<node_name> or end]
```

To jump to another node on failure at this one, use,

```
jumpdown [cond (<conditional expression>)] to [<node_name> or end]
```

where <conditional expression> is an expression, either true or false, that can include path dependencies and conditional parameter expressions. For details on the required syntax for the conditional expression, see the section on conditional expressions below. Some examples are

```
jumpup to end
jumpdown to B7
jumpdown cond ((sprays off) and (+A1-A2)) to A4
```

For multi-branching nodes, a slightly different syntax must be used:

```
jump [cond (<conditional expression>)] from <branch name> to
[<node_name> or end]
```

where <branch name> is defined in Section 5.2.1.1. Note that this syntax can also be used for binary branching nodes, since the ‘up’ branch is assumed to have the name ‘Y’, and the ‘down’ branch is assumed to have the name ‘N’.

5.2.3 Conditional Node Probabilities

This operation can only be done at a branching node. To set the probability of selecting a given branch at this node dependent on the path or parameter values, use

```
cond (<conditional expression>) {prob [<branch name>] [<real or
distributed parameter, or constant>]}
```

For example

```
cond ((response CoreMelt3) or (A1 N, A2 Y)) prob Y P5
```

For binary branches, the <branch name> may be omitted, in which case a branch name of ‘Y’ is assumed. This ensures that older-style input files will still work. Note that probabilities must be assigned for one less than the number of branches. The probability of the last branch is calculated automatically.

5.3 DEFINING NUMERICAL FUNCTIONS

This operation can only be done from the header block or from straight nodes. Real-valued or conditional-valued functions of real, conditional or branching node parameters can be defined within the node information. Functions can be used to set the values of real or conditional parameters (see the following section). They are defined thus:

```
function [<type>] <function name> (<parameter_list>)
code
< lines of "C" code, for the body of the function >
end
```

The <type> declaration may be real or cond. If it is omitted, real is assumed. For real functions, the “C” code must return a value of type double; for cond functions, the “C” code must return a value of type int, where non-zero represents TRUE and zero represents FALSE.

The input parameters can be node names, or real or conditional variables. The parameter list is specified in this way:

```
(<real | cond | node> <parameter 1>, <real | cond | node> <parameter 2>, .. <real | cond | node <parameter N>)
```

To use the parameters inside the function body, use a variable of type double for a real parameter and a variable of type int for a conditional parameter, as for the function type. The names of the variables inside the function body are the same as the names of the corresponding parameters in the parameter list. Node parameters are implemented as variables of type int, and have the value of 1 for branch Y or the first-named branch, 2 for branch N or the second-named branch, 3 for the third-named branch, and so on. In the current version of SOLOMON it is not possible to use branch names inside functions.

An example of a function definition is

```
function StaticPressure(real T, real V, cond failed, node N)
code
double pressure;
if (failed) pressure = 1.0e5;
else
{ if (node == 1) pressure = 5000.0 * T/V;
  else pressure = 6000.0 * T/V;
}
return pressure;
end
```

which defines a function StaticPressure which accepts two real arguments, one conditional argument and one node argument.

5.4 SETTING PARAMETERS

This can only be done from the header block or from a straight node. The only parameters that can be set are real, conditional and enumerated - the others (interpolated and distributed) can be used only on the right hand side of an assignment.

```
set [cond (<expression>)] <parameter> to <value>
```

where the value must be valid for that parameter type. Some examples are

```
set response to CoreMelt3
set Pressure to 100.0
set Temperature to T1
set cond ((-A1+B1) and not (Time early)) Pressure to P5
```

Real and conditional parameters can also be set to the values of a pre-defined function, or a calculation. For a function, the syntax is

```
set [cond (<expression>)] < parameter> to <function name>
(<parameter_list>)
```

for example, with the function `StaticPressure` defined in the above section, use

```
set cond (-A1+B1) Pressure to StaticPressure(Temperature, Volume,
IsFailed, Node25)
```

To set the real parameter to the result of a calculation, use the keyword `calc`, i.e.

```
set [cond (<expression>)] <real parameter> to calc(<numeric
expression>)
```

for example

```
set Pressure to calc(5000.0 * Temperature/Volume)
```

A real parameter can be set to the interpolated value of a curve supplied by the user using the command

```
set <real parameter> to <curve name> at <real parameter>
```

See Section 5.5.5 on how to set up a parameter that represents a curve for interpolation.

5.4.1 Arithmetic Expressions

Arithmetic expressions can take one of the following forms:

<element>, where <element> is one of:

<arithmetic constant>, e.g. 3.1415927

<variable name>

<element> <operator> <expression>, where <operator> is one of

+, -, *, /

(<expression>)

<user-function>(<argument-list>)

<standard “C” function>(<argument-list>) where the functions supported are

sin	cos	tan
asin	acos	atan
sinh	cosh	tanh
exp	log	log10
sqrt	ceil	floor
fabs		

pow atan2 fmod

The functions frexp, modf and ldexp are not supported.

5.5 DEFINING AND USING PARAMETERS

Parameters can be defined and set within the header block or within straight nodes and they can be of five types: *real*, *conditional*, *enumerated*, *distributed* and *interpolated*.

5.5.1 Real parameters

These are defined with the instruction:

```
define real <name> [global]
```

for example

```
define real Pressure
```

If the keyword `global` is absent, the value of the variable must be set in the same node as it is defined. In subsequent nodes, it becomes read-only. If 'global' is present, the value of the variable may be altered in subsequent nodes.

5.5.2 Conditional parameters

These are defined with the instruction:

```
define cond <name> [global]
```

for example

```
define cond Containment_failed
```

If the keyword `global` is absent, the value of the variable must be set in the same node as it is defined. In subsequent nodes, it becomes read-only. If it is present, the value of the variable may be altered in subsequent nodes.

5.5.3 Enumerated parameters

These are defined with the instruction:

```
define enum <name> (<value>, <value>, ..<value>)
```

for example

```
define enum Time (early, middle, late)
```

5.5.4 Distributed parameters

Parameters can be defined as randomly distributed using the instruction

```
define distribution <name> <distribution type> <parameters>
```

where the distribution types are those permitted by the Latin Hypercube Sampling Program. Each distribution type requires a set of real constants to specify it. The distributions and their associated parameters are as follows:

Distribution	Parameters	Description
Normal	a b	A normal distribution where a is the mean and b is the standard deviation.
Lognormal	a b	A lognormal distribution where a is the median and b is the log standard deviation.
Uniform	a b	A uniform distribution where a is the mean and b is the standard deviation. This value is $\sqrt{1/12}$ of the distance from the mean to the upper (or lower) limit.
Triangular	a b	A symmetrical triangular distribution where a is the mean and b is the standard deviation. This value is $\sqrt{1/24}$ of the distance from the mean to the upper (or lower) limit.

For example, the instruction

```
define distribution Pressure triangular 6.5 1.2247
```

defines the parameter Pressure as randomly distributed according to a triangular distribution with endpoints at 0.5 and 12.5, and mean at 6.5.

Correlations

Any two defined distributed parameters P and Q can be correlated. The instruction to specify a correlation coefficient of r is

```
correlate P Q r
```

Since SOLOMON uses a sampling technique to treat randomly distributed parameters, it cannot be guaranteed that the sample values are correlated exactly to r . Note that values of exactly ± 1 are not allowed - use ± 0.99 instead.

5.5.5 Interpolated parameters

These are defined using the instruction:

```
read <file_name> into <curve name> (<xtype>, <ytype>)
```

where <file_name> is the name of a file that contains a list of points on the curve, and <xtype> and <ytype> specify the form of the interpolation at each axis: this can be either 'log' for logarithmic interpolation or 'lin' for linear interpolation. Where this type of parameter is used, the user must supply a text file called <file_name> written in the format

```
<x1> <y1>
<x2> <y2>
...
<xn> <yn>
```

where <x> are the ordinates and <y> the abscissae of the curve. Up to 700 data points are allowed. For example, with the instruction

```
read "pressure.dat" into PressureFailureCurve (lin,log)
```

and a file 'pressure.dat' which contains the text

```
10.0  1.0e-4
20.0  1.0e-3
30.0  5.0e-3
40.0  6.5e-2
60.0  1.7e-1
90.0  3.0e-1
120.0 9.5e-1
150.0 1.0
```

the parameter PressureFailureCurve would be set up to interpolate the data in 'pressure.dat' on a lin,log scale.

5.5.6 Show Parameters

The parameters used in SOLOMON can be saved to a file for analysis using a number of tools. These tools include the PDF/CDF viewer described in Section 6.5.3 which is a simple graphical tool for displaying the range of values at various points in the CET. The data are written to files in comma-separated-variable (CSV) format which enables a range of applications such as spreadsheets to be used to analyse the data. To use this feature, the command

```
show <parameter_name>
```

is used at any straight node. The values of the parameter at the node are then saved for later analysis.

5.6 SUPERNODES

To help the user make sense of a possibly complicated tree, and to aid the user with assignment of end categories, the idea of 'supernodes' has been introduced. A supernode is either a branching node or a Boolean combination of branching nodes defined by the user. For example, a supernode defined as (path A Y, B N) is 'up' when node A follows branch Y and node B follows branch N; otherwise it is 'down'. The supernodes can be given meaningful names; for example path A Y, B N could be called Explosion_with_vessel_failure.

When SOLOMON is run, during the first LHS sample unique combinations, or paths, of the supernodes are stored and their probabilities summed. This information can then be drawn by using the supernode viewer. The resulting tree can be helpful in defining the conditions for the end categories.

SOLOMON allows the user to specify multiple supernode trees. This can be useful if there is more than one way to simplify the event tree. The syntax to do this is

```
supernodetree <tree name 1>
```

```
<supernode definitions>
supernodetree <tree name 2>
<supernode definition>
etc.
```

5.6.1 Defining Supernodes

The syntax to define a supernode is

```
supernode <supernode name> cond (<conditional path expression>)
```

for example

```
supernode EarlyOverpressure cond ((-A1+A2) or (+B1))
```

or

```
supernode EarlyOverpressure cond ( A1 N, A2 Y or B1 Y)
```

defines a supernode EarlyOverpressure that is true for any path in which the node A1 is down and A2 is up, or node B1 is up. Note the following restrictions applied to supernodes:

- Only one conditional expression (of any complexity) is permitted, unlike the end categories where a list of conditional expressions can be given.
- Only prior path dependencies can be included in the conditional expression defining a supernode; parameters cannot be included.

Therefore the following statement is legal:

```
supernode One cond ((+A-B+C) or ((-A+B-C) and (-A+D)))
```

or:

```
supernode One cond (A Y, B N, C Y or (A N, B Y, C N and A N, D Y))
```

whereas the following is illegal:

```
supernode Two cond ((+A-B+C) and (sprays off) and (Pressure <
125.0))
```

5.6.2 Drawing the Event Trees

Supernodes can be used to print out parts of the event trees. A supernode viewer (which is activated from the 'run' menu of the GUI) is available which reads the supernode information printed by SOLOMON and prints a diagram that represents the supernode trees. If a supernode tree can be drawn, i.e. if the supernodes represent a valid event tree that is not too large, the supernode viewer draws the tree producing a graphical image.

Note that, if a supernode is bypassed in a path, then its value is defined as 'null', and this is indicated on the diagram as a non-branching node.

The supernode viewer can zoom in and out, pan the image, and alter the size of the image. In addition, the image can be copied to a graphical clipboard and then pasted as a graphical image into another program, such as Word for Windows.

5.6.3 Supernode Tree Start Condition

The supernode tree start condition allows the user to instruct the supernodes tree to start at a given place in the event tree. This is useful for examining a small part of a large tree without having to condense it. The syntax is

```
start cond <conditional expression>
```

For example given an event tree with three nodes, A, B and C, the part of the tree where A has occurred can be examined with the instructions

```
start cond (+A)
supernode XB cond (+B)
supernode XC cond (+C)
```

5.7 END CATEGORIES

The large number of paths produced by the event tree can be reduced to a smaller number of end categories. The end categories are specified following the node information in the SOLOMON input file. The first statement, defining the default endcat into which all paths are grouped that do not belong to any other end category, must be

```
endcategory default <name>
```

followed by each end category and a list of the conditional expressions (usually path dependencies) that apply to it, i.e.

```
endcat <name>
cond (<expression 1>)
cond (<expression 2>)
..
cond (<expression n>)
```

In use, if a path satisfies any of the conditions in the list, then that path will be assigned to the end category. SOLOMON checks whether each path belongs to more than one end category; this undesirable situation is called an *endcat conflict* and, if detected, the program stops with an appropriate error message.

An example of a system of end categories for a tree with nodes A, B and C is

```
endcategory default DefaultEndcat

endcat SourceTerm1
cond (-A+B-C)
cond ((-B) and (sprays off))

endcat SourceTerm2
cond (+A+B-C)
cond ((-B) and (sprays on))
```

which defines end categories DefaultEndcat, SourceTerm1 and SourceTerm2.

5.7.1 Nocheck

By default, the code checks for conflicts in end category definition. If a particular path can be binned into two or more endcats, the code will return an error and terminate the calculation. However, an advanced feature of the code allows the user to disable the end category checking, which is performed by entering the key word “nocheck” in the header block. The user is advised to exercise care in the use of this feature.

5.7.2 Endcat parameters

SOLOMON provides a method to allow the output from one calculation to be used to define the run sequences for another. When a run sequence is performed, SOLOMON writes details of the endcat parameters to a file named

```
all.pds
```

To use this feature, the user inputs the following text in the endcat objects :

```
frequency <run_frequency>
<enumerated_parameter> (<values>*) (repeat line as required)
```

The run frequency is a value between 0.0 and 1.0.

To use the file, the user writes

```
pdsfile '<PDS_file_name>'
```

in the header block. (This has to be done using the text editing features as there is no special field for this in the GUI). It is then possible to define a run sequence by entering in the header block

```
run <run_name> <system_state>
```

where <system_state> is the name of an endcat listed in the PDS file. The values of the parameters to define the run sequence are then read from the PDS file.

5.8 END OF THE DATA FILE

This last keyword, which must be included, tells the program that no more data is coming:

```
enddata
```

The GUI adds this keyword automatically.

5.9 CONDITIONAL EXPRESSIONS

Conditional expressions take the form

```
cond ( [not] <condition>
[and|or <condition>]
[and|or <condition>]
...
)
```

where <condition> is of the form

```
cond ([path] +<node>-<node>+...+<node>)
```

or

```
(<real or distribution parameter> {=|>|<|<>|>|=|<=} <real>)
```

or

```
(<conditional parameter>)
```

or

```
(<enum parameter> <value>, <value>, ..., <value>)  
...)
```

and can contain any number of elements up to a total line length of 500 characters. Additional parentheses may be added to define the order in which the and, or operators are calculated.

For example, the expression

```
cond ((path +A-B) and (temperature <= 100.0) and (response = AE,AB))
```

is true when node A is up, node B is down, temperature <= 100 and response is either AE or AB. Note that the keyword 'path' is optional, e.g.

```
cond (path +A-B+C-D)
```

can be simplified, if required, to

```
cond (+A-B+C-D)
```

which can simplify some complex path conditions.

5.9.1 If-then-else

The conditions can be included in an if-then-else structure, i.e.

```
if cond(<conditional expression>)  
  set...  
elseif cond (<conditional expression>)  
  set...  
else  
  set...  
endif
```

5.10 EXAMPLE

In this section we present an annotated example SOLOMON input file. Note that SOLOMON is case-sensitive. In this example, for clarity, all the SOLOMON keywords are in lower case and other words are in upper case. In use, the keywords must be in lower case but other words can be in upper or lower case. The example involves a five node event tree with nodes A, B, C, D and E where nodes A and D are straight, the rest are branching. The event tree uses

numerical, enumerated, interpolated and randomly distributed parameters, a user-defined function, conditional node probabilities, supernodes and endcats.

EXAMPLED.IN

```
title "Solomon 2.20 Example input file"
samples 200
cutoff 0
showpaths frequency 0

#NAME          exempld.in
#AUTHOR        G F Holford
#MODIFIED       G J Roberts
#DESCRIPTION    Example Solomon input file.

# user-defined functions. note code...end for function body...
# note `node' is implemented in "C" as `int', and its value refers
# to the currently-selected branch...
# note `real' is implemented in "C" as `double'...

function real f1 (node N1, real R1)
code
  double j;
  if ((N1 == 2) && (R1 > 100.0))
    j = 3.0;
  else
    j = 4.0;
  return j;
end

function cond f2 (node N1, real R1)
code
  int j;
  if ((N1 == 2) && (R1 > 100.0))
    j = 1;
  else
    j = 0;
  return j;
end

# note `cond' is implemented in "C" as `int' (the same as a
boolean)...

function cond f3 (cond c1, real r1)
code
  return c1;
end

# define parameters...

# local enumerated parameter...
define enum e1 (true, false)

# global real parameter...
define real p global

# local real parameter...
define real q

# global conditional parameter...
define cond c1 global

# local conditional parameter...
define cond c2
```

```

# local enumerated parameters... used to define run sequences...
define enum E1 (E1_1, E1_2, E1_3)
define enum E2 (E2_1, E2_2, E2_3, E2_4)

# define interpolated parameter d1 by reading from file
"press.dat"...
read press.dat into d1 lin, lin

# define run sequences...
run run1 (E1 E1_2, E2 E2_1) frequency 1.0
run run2 (E1 E1_3, E2 E2_2)

# THIS IS THE END OF THE HEADER BLOCK
*****

node straightnodeZ1 type straight
# first node.
# assign values to parameters...
set p to 0.4
set q to 1.0
set c1 to cond(branchingnodeA 2, branchingnodeB B4)
set e1 to true
set p to ((d1 at 50) + normal 0.6 0.06)
#
# define correlated distributions...
define distribution d2 uniform 0.1 0.01
define distribution d3 lognormal 0.2 0.02
correlate d2 d3 0.99
#
define real prob1
set prob1 to calc( d2*d2*d2)

node branchingnodeA0 type branching
# branching node, old style definition.
# probability of '+' branch defined using frequency distribution...
prob normal 0.5 0.1
# conditionally override the probability...
cond((c1 and c2) and not (E1 E1_2)) prob 0.4
cond (e1 true) prob 0.3

node branchingnodeA type branching
branch A1
branch A2
# branching node, new style definition.
# define probability of branch A1...
prob A1 prob1
# examples of overridden probability...
# cond (e1 false) prob A2 0.3
# cond (e1 true) prob A2 0.2
# examples of straightlining...
# jumpup cond(e1 false) to end
# jump cond(e1 true) from A1 to branchingnodeC

node straightnodeZ2 type straight
# straight node.
# assign value to p, depending on which way branching node A went...
set cond(branchingnodeA A1) p to calc(p / 2.0)
set cond(branchingnodeA A2) p to calc(p / 4.0)
# more assignment statements... use of interpolated parameters and
functions...
set p to ((d1 at 50) + (50.0 * f1(branchingnodeA, 0.3)))
set p to f1(branchingnodeA, 0.3)
set c1 to cond(f2(branchingnodeA, p) and f2(branchingnodeA0, 0.1))

```

```

set p to f1(branchingnodeA, 500.0)
# mark parameter "p" for monitoring with the "show" program...
show p

node branchingnodeB type branching
branch B1
branch B2
branch B3
branch B4
# a 4-way branching node.
# define the probabilities of each branch...
prob B1 0.1, B2 0.2, B3 0.3

node branchingnodeC type branching
branch C1
branch C2
branch C3
# a 3-way branching node.
# define the probabilities of each branch...
prob C1 calc (p/2000), C2 0.1
# examples of overriding the probabilities...
# cond (((+branchingnodeA+branchingnodeB) and (branchingnodeA A2,
branchingnodeB B4)) and ((e1 true) or not (p > 0.1))) prob C1 0.8,
C3 0.0001
# cond (+branchingnodeA+branchingnodeB) prob C1 0.8, C2 0.1

# THIS IS THE END OF THE NODES
*****
supernodetree treel

supernode supernodeA
# defining supernode supernodeA
node branchingnodeA

supernode supernodeB
# defining supernode supernodeB
node branchingnodeB

supernode supernodeC
# defining supernode supernodeC
node branchingnodeC

# THIS IS THE END OF THE SUPERNODES
*****

endcategory default none

endcat endcat1_1
# define endcat1_1
frequency 0.01
parameters (E1 E1_1, E2 E2_3)
cond (branchingnodeA A1, branchingnodeB B1)

endcat endcat1_2
# define endcat1_2
frequency 0.02
parameters (E1 E1_2, E2 E2_1, e1 true)
cond (branchingnodeA A1, branchingnodeB B2)

endcat endcat1_3
# define endcat1_3
cond (branchingnodeA A1, branchingnodeB B3)

endcat endcat2_1

```

```

# define endcat2_1
cond (branchingnodeA A2, branchingnodeB B1)

endcat endcat2_2
# define endcat2_2
cond (branchingnodeA A2, branchingnodeB B2)

endcat endcat2_3
# define endcat2_3
cond (branchingnodeA A2, branchingnodeB B3)

# THIS IS THE END OF THE ENDCATS
*****
enddata

PRESS .DAT
0 10
1 20
3 30
100 40

```

6 The Graphical User Interface

6.1 INTRODUCTION

SOLOMON is provided with a Graphical User Interface (GUI) which offers the following facilities:

1. Management of SOLOMON's input and output files.
2. Help with creating and editing the input files.
3. A method of launching a SOLOMON calculation.
4. Access to the Supernode Viewer.

When the GUI is first entered, the user sees a set of options on the menu bar:

File View Run Help

where 'File' offers the standard options for opening, saving and printing files. Only 'Print Object' will be unfamiliar to the experienced Windows user. This option allows the user to print a single SOLOMON object (e.g. a node definition). This option is only available when a file is open for editing.

'View' controls whether or not the toolbar and status bar are visible. (Additional options are available when a file is open for editing.)

'Run' contains four sub-options. 'Run / Solomon' allows the user to run SOLOMON via a file selection dialog to specify the name of the input file - this option allows the user to bypass the procedure for opening a file for editing. 'Run / Supernode Viewer' allows the user to

access the supernode viewer, described in Section 6.5.2. Run / PDF/CDF viewer allows the user to examine the probability distribution and cumulative distribution functions for end categories and pre-defined physical parameters, described in Section 6.5.3. Run / Output Viewer allows the user to examine any of the textual output via a user-defined software package which is specified in the solomon.ini file, as described in Section 6.5.4.

'Help' gives access to the SOLOMON online help, which is built around the standard Windows help system, and offers the same user image.

Note that the principle of the input file editor in the SOLOMON GUI is that it generates text for the SOLOMON suite. It provides a number of dialogs for creating text, and enables the user to avoid the more serious syntax errors, but once the text is generated, it can only be modified by either regenerating the text in the dialog boxes, or by modifying the text using text editing methods. In other words, it behaves as though it were made of 'Syntax Wizards' which provide the basic framework for the user, but still require the user to edit some text by hand.

There are two reasons for this. Firstly, it ensures that the GUI is compatible with text-generated SOLOMON input files, since it uses the same format of file as a text editor. Secondly, it enables the GUI to allow the user to input data using either dialog boxes or text, whichever is easier for the particular user (experienced or novice) or the task being performed, because the user always sees the input data set in terms of the text being generated.

There are two principal ways of editing a SOLOMON input file; either by editing the file as a normal text file using the standard text editing features (or even outside the SOLOMON GUI using another text editor) or by using the object editing dialog boxes provided in the SOLOMON GUI. The object editing dialog boxes can be accessed in two ways. The user may select an object by clicking on its rectangle, and then selecting Define / Object... from the menu; alternatively the user may double-click on the object's rectangle. If the object contains no text, apart from its name and type, the GUI provides an object creation dialog which allows the user to enter the required data using forms. If the object already contains text, the GUI provides an object editing dialog, which is suitable for making minor modifications and is described in Section 6.3.

6.1.1 Limitations of the GUI

The GUI is not able to implement all features of SOLOMON input by dialog boxes alone. Some features still have to be entered by the user by typing in text. (The GUI specifically permits this method of data entry except in 'create object' dialogs. The user may need to exit a 'create object' dialog, and re-edit the object using the 'edit object' dialog or by typing in the object display window).

Specific features not implemented in 'create object' dialogs are:

1. System state files (*.pds) as input.
2. User-defined parentheses (to control complex logic involving AND and OR) when defining endcats and supernodes.

3. Conditional jumps in branching nodes.
4. Implementation of the ‘nocheck’ keyword (see section 5.7).

6.2 CREATING A NEW SOLOMON INPUT FILE

6.2.1 Starting a new file

The user opens a new file for editing by selecting ‘File / New’ from the opening menu. The menu bar now looks like this:

```
File  Edit  View  Insert      Define      Run  Window
Help
```

and a sub-window now appears inside the main SOLOMON window. The sub-window consists of two parts, referred to as the left-hand side and right-hand side. The left-hand side shows the SOLOMON objects defined for this file; initially a header block (dark blue background) and the end-of-data marker (white background). The right-hand side shows the text of the currently-selected object; initially the header block containing only a default title. This text can be edited as a normal text-edit window (see Figure 6 on page 10).

Multiple files can be open at once, and multiple views of the same file can be open using the Window option.

6.2.2 Adding information to the header block using the header creation dialog

This dialog, headed ‘Define General Parameters’, contains panels and buttons for entering the data required for the SOLOMON input file header block.

The features provided are:

1. Title of the file. This is a character string of up to 500 characters.
2. Header block comments. Any number of lines of text, to help identify the file or to explain some of its features. Note that the SOLOMON comment symbol, ‘#’, is added automatically by the GUI.
3. The number of Latin hypercube samples. Any positive integer.
4. The cut-off value. A real number in the range 0.0 to 1.0.
5. The frequency of showpaths. See Section 5.1.5.
6. A button, ‘Define Parameter...’, to define parameters. See Section 6.4.2.
7. A button, ‘Define Function...’, to define functions. See Section 6.4.3.
8. A button, ‘Set Parameter Values...’, to set parameter values. See Section 6.4.4.
9. A button, ‘Define Run Sequence...’, to define run sequences or system states. See Section 6.4.5.

10. A button, 'Correlate...', to define a correlation between two distributed parameters. See Section 6.4.7.
11. An 'OK' button to transfer the generated text to the working file.
12. A 'Cancel' button to return to the object selection window without saving the changes.
13. A 'Help...' button to access online help.
14. A 'preview' panel which allows the user to see the text generated, as it is built up, and to select the location of the next insertion operation (i.e. an operation initiated by one of the buttons 6.-10.).

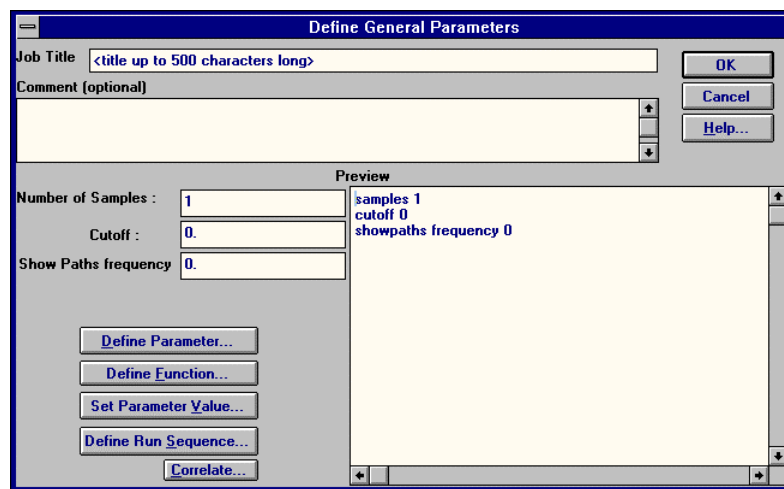


Figure 7: Defining General Parameters

An insertion operation can be performed at any location in the header block except the first six lines, which is where the GUI puts the samples, cut-off and showpaths values. The GUI automatically repositions any attempted insertions in this area to the seventh line.

6.2.3 Inserting a new SOLOMON object

New objects are added to the SOLOMON input file in two stages: firstly, the user selects the location of the new object by clicking on the rectangle of an object (thereby making it the current object). The new object will be inserted *before* the current object. Secondly, the user selects 'Insert' and the type of object to be added from the menu. The GUI will prevent objects being inserted in incorrect locations in the file. For example, no objects may be inserted before the header block (which must always be the first object) and nodes may not follow supernodes or endcats in the file.

When a new object is inserted, it becomes the current object, and the GUI automatically opens the appropriate object creation dialog.

6.2.4 Creating a new straight node

A straight node in SOLOMON is essentially a series of statements to define and set parameters and functions for later use. For this reason, the dialog box for creating straight nodes contains little more than a set of buttons which open other text-creation dialogs.

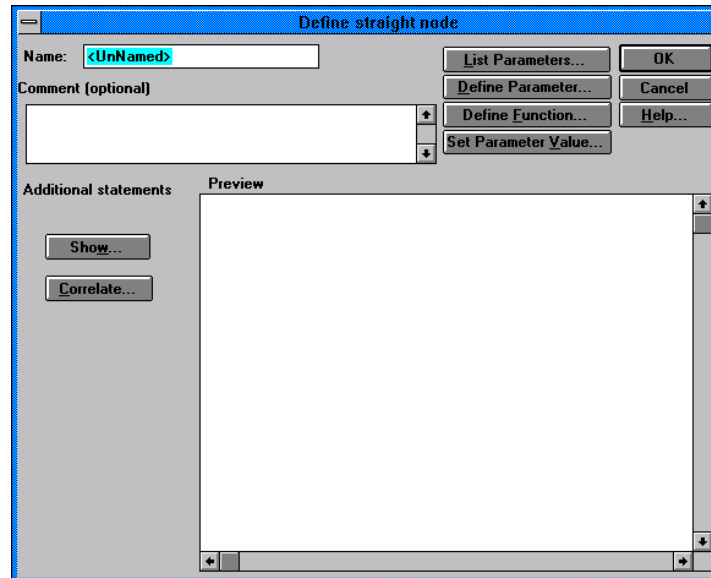


Figure 8: Defining a Straight Node

The features provided by the 'Define Straight Node' dialog are:

1. The name of the node. This is a character string up to 30 characters.
2. Straight node comments. Any number of lines of text, to help identify the node or to explain some of its features. Note that the SOLOMON comment symbol, '#', is added automatically by the GUI.
3. A button, 'List Parameters...', to list the parameters defined in objects above the current object. See Section 6.4.1.
4. A button, 'Define Parameter...', to define parameters. See Section 6.4.2.
5. A button, 'Define Function...', to define functions. See Section 6.4.3.
6. A button, 'Set Parameter Values...', to set parameter values. See Section 6.4.4.
7. A button, 'Show...', to select parameters for display using the 'show' keyword. See Section 6.4.6.
8. A button, 'Correlate...', to define a correlation between two distributed parameters. See Section 6.4.7.
9. An 'OK' button to transfer the generated text to the working file.

10. A 'Cancel' button to return to the object selection window without saving the changes.
11. A 'Help...' button to access online help.
12. A 'preview' panel which allows the user to see the text generated, as it is built up, and to select the location of the next insertion operation (i.e. an operation initiated by one of the buttons 4.-8.).

An insertion operation can be performed at any location in the straight node.

6.2.5 Creating a new branching node

Branching nodes are the points in the SOLOMON input file which define the probability of following a particular branch in the tree.

The 'Edit Branching Node' dialog provides the following features:

1. The name of the node. This is a character string up to 30 characters.
2. Straight node comments. Any number of lines of text, to help identify the node or to explain some of its features. Note that the SOLOMON comment symbol, '#', is added automatically by the GUI.
3. A 'Branch Names' panel. The user enters the names of the branches here. Each branch name is written on a separate line and may be up to 30 characters.
4. A button, 'List Parameters...', to list the parameters defined in objects above the current object. See Section 6.4.1.
5. A 'Define Probabilities...' button to define the probability values for this node. See Section 6.2.5.1.
6. A text window to enter the name of the node to jump to from a branch. The user highlights a branch in the 'Branch Names' panel, and then enters the name of the 'jump to' location in the text window. The user may enter 'end' as a jump destination. Note (a) the GUI does not provide an equivalent to 'jumpup to' or 'jumpdown to' and (b) does not verify that the destination entered is a valid node name.
7. An 'OK' button to transfer the generated text to the working file.
8. A 'Cancel' button to return to the object selection window without saving the changes.
9. A 'Help...' button to access online help.

A 'preview' panel which allows the user to see the text generated, as it is built up.

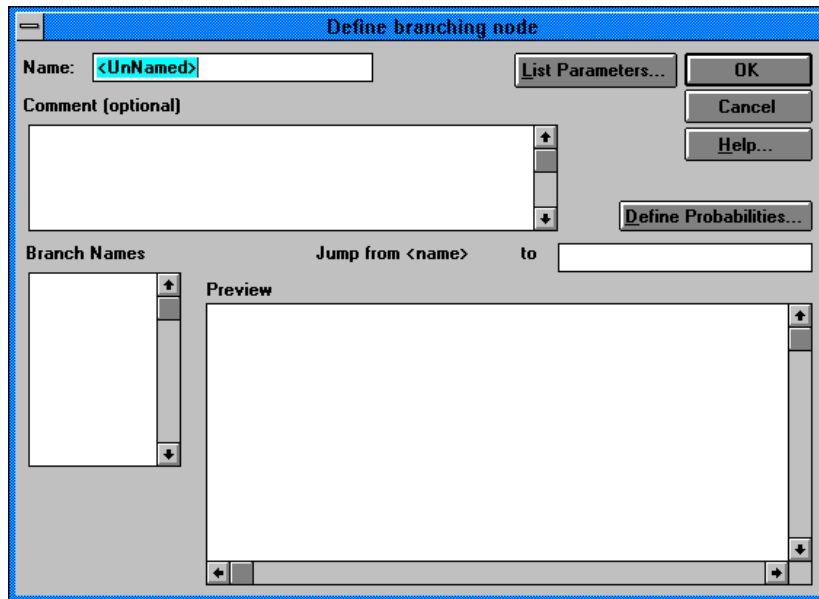


Figure 9: Defining a Branching Node

6.2.5.1 Defining branch probabilities

The 'Define Branching Probabilities' dialog allows the user to enter the branching probabilities in the same way as the SOLOMON text-based input.. The user selects a branch from the list of branch names, and enters the probability in the 'Probability of Branch' window. One fewer than the number of branches must have its probability defined, and the GUI checks that this is the case. The user may make the entire set of probabilities conditional. The dialog box may be entered as many times as the user requires.

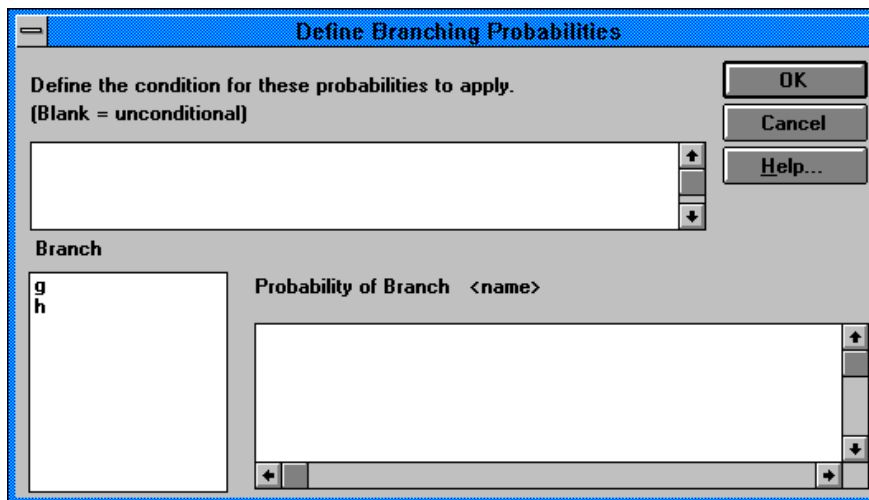


Figure 10: Defining Branch Probabilities

The dialog has the following features:

1. A multi-line text window for entering the condition for the defined probabilities to apply. If blank, the defined probabilities will apply by default.. As with text-based

SOLOMON, default probabilities apply unless the user also supplies conditional probabilities and the supplied condition becomes true.

2. A selection list of branch names, from which the user selects a branch name which becomes the current branch.
3. A multi-line text window for entering an expression for defining the probability that the node will follow the current branch.
4. An 'OK' button to transfer the generated text to the working file.
5. A 'Cancel' button to return to the object selection window without saving the changes.
6. A 'Help...' button to access online help.

6.2.6 Defining Supernode Trees

The syntax of SOLOMON requires that the supernodes are organised into supernode trees. To define a supernode tree, the user can enter the 'Define Supernode Tree' dialog.

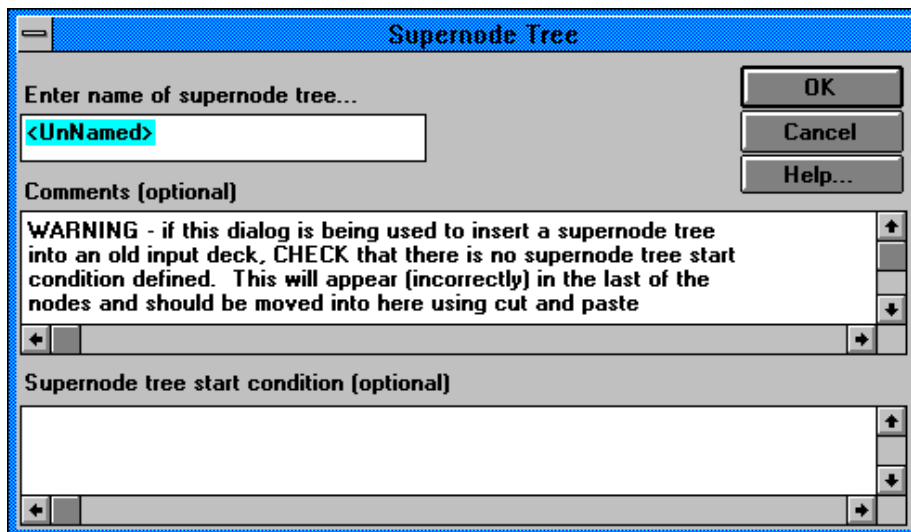


Figure 11: Defining a Supernode Tree

This dialog provides the following features:

1. The name of the supernode tree. This is a character string up to 30 characters.
2. Supernode tree comments. Any number of lines of text, to help identify the node or to explain some of its features. Note that the SOLOMON comment symbol, '#', is added automatically by the GUI.
3. Supernode tree start condition. A multi-line text window in which to define the start condition for the supernode tree. May be empty.

4. An 'OK' button to transfer the generated text to the working file.
5. A 'Cancel' button to return to the object selection window without saving the changes.
6. A 'Help...' button to access online help.

Note that if a supernode tree is added to an existing input file which previously contained a supernode start condition but no supernode trees, then the supernode start condition will have to be moved from the last node to the supernode tree. This is because the GUI does not recognise the supernode start condition as an object delimiter, so associates it with the last node.

6.2.7 Defining Supernodes

Supernodes are defined in one of two ways: either a supernode is a branching node, or is a combination of paths through the event tree. The 'Define Supernode' dialog can be used for either specification.

If the supernode is a branching node, the user simply selects the node from the list of branching nodes and clicks on the 'Supernode=Node' button. If this is the required effect, the user finishes by clicking on the OK button. Note that the 'Supernode=Node' button changes its name to 'Supernode=Tree' when it is pressed. This enables the user to change between types of specification.

If the supernode is a path, the user selects a branching node from the selection list, and the dialog responds by presenting the list of branches for that node. The user selects a branch from this list, and then clicks on the Preview window, which then shows the supernode condition so far. At this point, the 'Insert AND' and 'Insert OR' buttons become ungreyed (i.e. available). The user may select one of these and then add another component to the path, finishing by clicking on the Preview window. When all the components of the required path have been added, the users finishes by clicking on the 'OK' button.

The dialog offers the following features:

1. The name of the supernode. This is a character string up to 30 characters.
2. Supernode comments. Any number of lines of text, to help identify the node or to explain some of its features. Note that the SOLOMON comment symbol, '#', is added automatically by the GUI.
3. A selection list of branching nodes.
4. A selection list of branches for the selected node.
5. An 'Insert AND' button which allows the user to add a path component in series with the currently-defined path. This is initially greyed-out.
6. An 'Insert OR' button which allows the user to add a path component in parallel with the currently-defined path. This is initially greyed-out.

7. A button, labelled 'Supernode=Node' or 'Supernode=Tree' depending on the context, which enables the user to change the mode of supernode specification.
8. An 'OK' button to transfer the generated text to the working file.
9. A 'Cancel' button to return to the object selection window without saving the changes.
10. A 'Help...' button to access online help.
11. A 'Preview' text window which enables the user to see the supernode definition as it is built up.

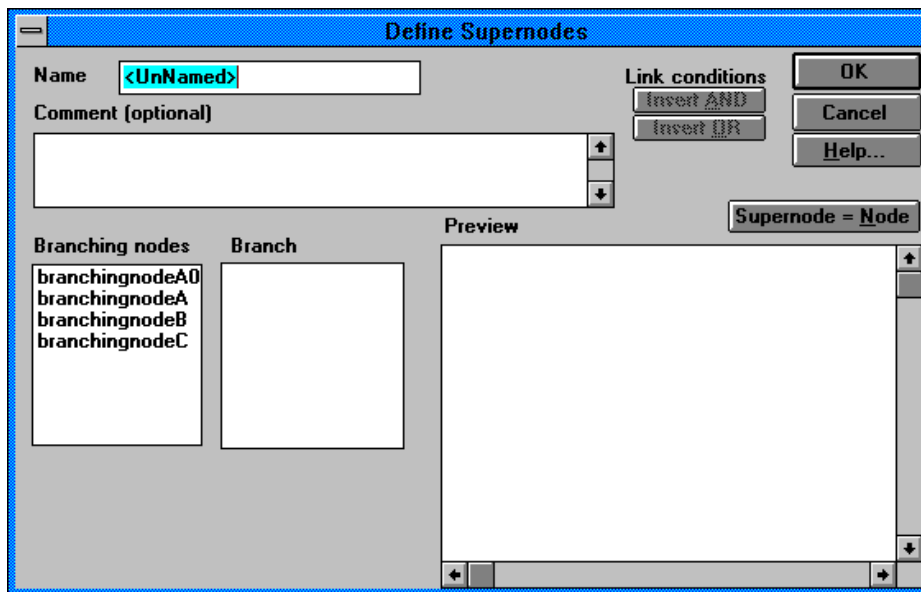


Figure 12: Defining a Supernode

6.2.8 Defining Endcats

The 'Define End Category' dialog appears very similar to the 'Define Supernode' dialog, mirroring the similarities between the text syntax of Endcats and Supernodes. Endcats are, in fact, a series of conditions. The dialog allows these conditions to be defined either as paths through the event tree or as 'custom conditions' (i.e. arithmetic and logical statements) and to link these conditions with 'and' and 'or'. The dialog box also allows multiple statements, as in the text version of SOLOMON. The dialog behaves like the supernode dialog.

To construct a condition which defines a set of paths through the tree, the user selects a branching node from the selection list, and the dialog responds by presenting the list of branches for that node. The user selects a branch from this list, and then clicks on the Preview window, which then shows the supernode condition so far. At this point, the 'Insert AND' and 'Insert OR' buttons become ungreyed (i.e. available). The user may select one of these and then add another component to the path, finishing by clicking on the Preview window.

To construct a condition which is an arithmetic and logical expression, the user types in the required text in the 'custom condition' window, and then clicks on the Preview window. Custom conditions and path conditions may be linked using the 'Insert AND' and 'Insert OR' buttons.

Endcats may have more than one condition. To finish declaring one condition and begin another, the user clicks on 'Add Condition'.

When all the required conditions have been defined, the user finishes by clicking on the 'OK' button.

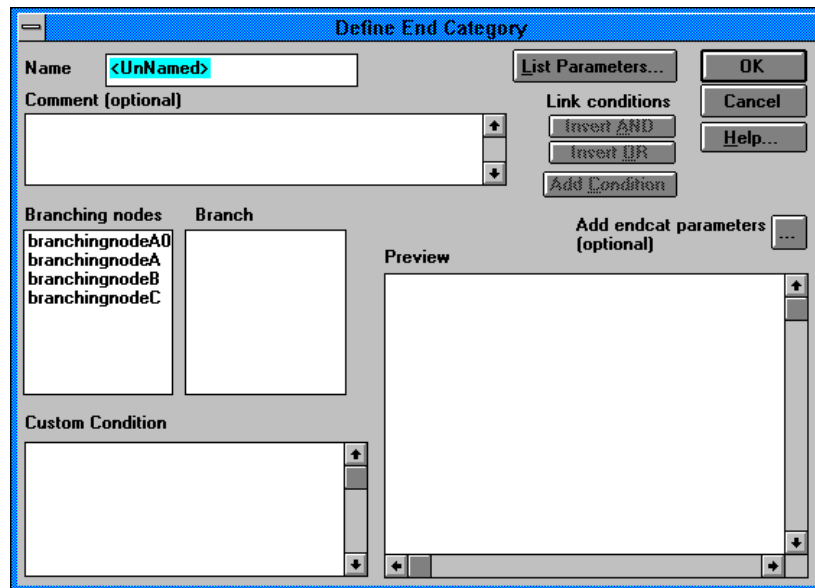


Figure 13: Defining an End Category

The dialog offers the following features:

1. The name of the endcat. This is a character string up to 30 characters.
2. Endcat comments. Any number of lines of text, to help identify the node or to explain some of its features. Note that the SOLOMON comment symbol, '#', is added automatically by the GUI.
3. A selection list of branching nodes.
4. A selection list of branches for the selected node.
5. A multi-line text window for entering 'Custom Conditions' which may be any logical expression. The GUI does not validate this expression.
6. A button, 'List Parameters...' which displays a list of variables and functions defined in preceding objects which the user may want to reference when defining custom conditions.

7. An 'Insert AND' button which allows the user to add a path component in series with the currently-defined path. This is initially greyed-out.
8. An 'Insert OR' button which allows the user to add a path component in parallel with the currently-defined path. This is initially greyed-out.
9. An 'Add Condition' button which allows the user to begin a new condition.
10. An 'OK' button to transfer the generated text to the working file.
11. A 'Cancel' button to return to the object selection window without saving the changes.
12. A 'Help...' button to access online help.
13. A button labelled '...' alongside the 'Define Endcat Parameters (optional)' string which enables the user to specify parameters associated with the endcat; this is an advanced feature enabling the endcats from one calculation to be used as the basis for defining the system states of another calculation.
14. A 'Preview' text window which enables the user to see the supernode definition as it is built up.

6.2.9 Endcategory default

SOLOMON requires that the first statement of the endcat data is the endcategory default statement. Note that the GUI does not enforce this, in order to allow the user more flexibility when constructing the input file.

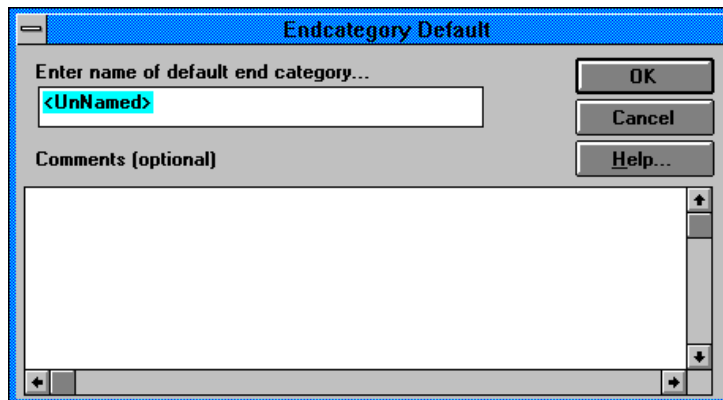


Figure 14: The Default End Category

The 'Define End Category Default' dialog offers the following features:

1. The name of the default end category. This is a character string up to 30 characters. It may be the word 'none'.

2. Comments. Any number of lines of text, to help identify the node or to explain some of its features. Note that the SOLOMON comment symbol, '#', is added automatically by the GUI.
3. An 'OK' button to transfer the generated text to the working file.
4. A 'Cancel' button to return to the object selection window without saving the changes.
5. A 'Help...' button to access online help.

6.3 MODIFYING AN EXISTING FILE

6.3.1 Features for modifying files

The GUI offers a number of features to help the user to modify existing text. These can be grouped as follows:

1. Cut/Copy/Paste of entire objects. See Section 6.3.2.
2. 'Edit SOLOMON Object' dialog. See Section 6.3.3.
3. Activate/Deactivate object. See Section 6.3.4.
4. Sub-dialogs. See Section 6.4.

6.3.2 Cut/Copy/Paste object

Beneath the 'Define' option of the file editing menu are three options, labelled 'Cut Object', 'Copy Object' and 'Paste Object'. These options work in a similar way to Cut/Copy/Paste for text, except that:

1. Only one object may be cut or copied at a time.
2. The GUI prevents the header block or enddata being cut or copied.
3. The GUI prevents objects being pasted into illegal locations in the file.
4. When an object is copied, the name of the copy is the name of the original with '\$' appended. This helps to prevent objects having the same names.
5. Objects can be copied from one view to another, or even from one file to another within the GUI, but cannot be transferred between the GUI and other applications.

6.3.3 Edit SOLOMON Object Dialog

This dialog is entered when the user selects Define / Object from the file edit window's menu or double clicks on the rectangle of an object, *and* the object already contains text.

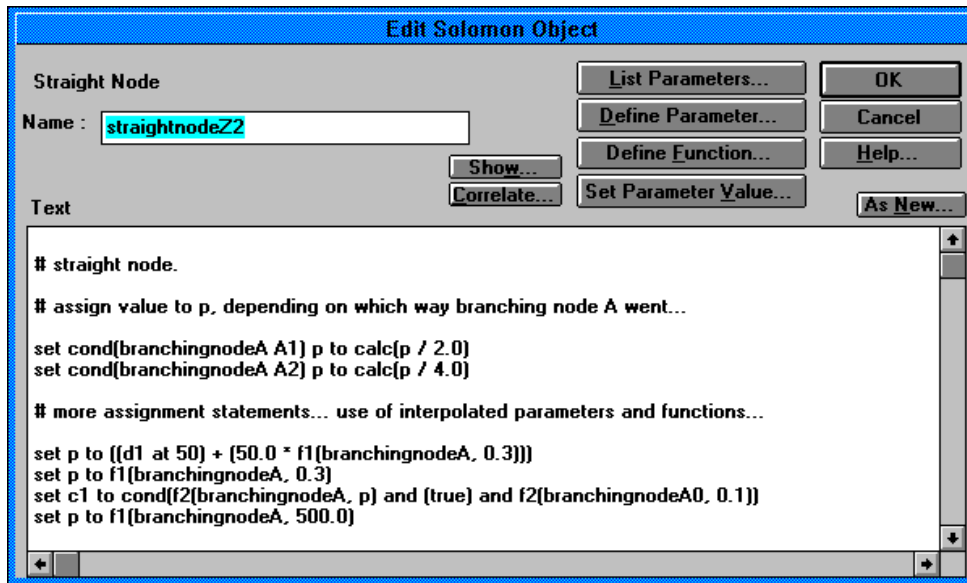


Figure 15: Editing a SOLOMON Object

The dialog offers the following features:

1. A line of read-only text indicating the type of object being edited.
2. The name of the node. This is a character string up to 30 characters.
3. The text window. Any number of lines of text, containing any SOLOMON input statements. The dialog initially fills this window with the previously-generated text, which may be edited using standard text-editing methods. Note that (a) cut/copy/paste of text is only possible using CTRL+X, CTRL+C and CTRL+V keys, and (b) the GUI does not validate the user input. Comments may be included, but the SOLOMON comment symbol, '#', must be included by the user. **(Do not use ### as this is a special character used by SOLOMON).** This window is also used to mark the location where the insertion operations 4.-5. and 7.-9. are intended to add their text.
4. A button, 'Show...', to select parameters for display using the 'show' keyword. Available only for straight nodes. See Section 6.4.6.
5. A button, 'Correlate...', to define a correlation between two distributed parameters. Available only for the header block and for straight nodes. See Section 6.4.7.
6. A button, 'List Parameters...', to list the parameters defined in objects above the current object. Not available for the header block. See Section 6.4.1.
7. A button, 'Define Parameter...', to define parameters. Available only for the header block and for straight nodes. See Section 6.4.2.
8. A button, 'Define Function...', to define functions. Available only for the header block and for straight nodes. See Section 6.4.3.

9. A button, 'Set Parameter Values...', to set parameter values. Available only for the header block and for straight nodes. See Section 6.4.4.
10. An 'OK' button to transfer the generated text to the working file.
11. A 'Cancel' button to return to the object selection window without saving the changes.
12. A 'Help...' button to access online help.
13. A button, 'As New...' which allows the user to rebuild the object from scratch using the appropriate object creation dialog. All the already-existing text is transferred to the preview window, and the appropriate object creation dialog is entered. Where appropriate, the dialogs are populated with values read from the object - for example, if the user redefines a branching node, the names of the branches already defined appear in the 'list of branches' box. If this dialog is terminated using the OK button, the text generated is transferred directly to the working file. If the dialog is terminated by using the Cancel button, then, as one would expect, the dialog returns to the state it was in before the 'As New...' button was pressed

6.3.4 Activate/Deactivate Object

The GUI provides a means for alternative versions of an input file to exist within the same physical file. Objects may be active or inactive. One use of this feature is to allow alternative versions of objects.

To deactivate the current object, the user selects 'Define / Deactivate' from the edit window's menu. Note the header block and enddata cannot be deactivated.

To activate the current object, the user selects 'Define / Activate' from the edit window's menu.

Note that Activate and Deactivate buttons are greyed out when they are not appropriate.

When the current object is inactive, each line of text in the text window begins with '###'. Objects can still be edited when they are inactive - it is safest to do this via the dialogs as these are guaranteed not to corrupt the '###' marker.

Inactive objects which are not the current object are displayed with grey text in their rectangles, as opposed to black text for active objects.

An alternative method of activating or deactivating an object is to position the mouse pointer over the object's rectangle and click the *right* mouse button. This toggles the active/inactive status of the object. This operation does not make the object become the current object.

Files can be run, saved and loaded with inactive objects.

6.4 SUB-DIALOGS

6.4.1 List parameters

This dialog, which has the title ‘List of Available Names’, displays the names of all the parameters and functions defined in objects above the current object, and helps the user to construct expressions. It is activated when the user presses the ‘List Parameters...’ button on one of the main definition dialogs. Unlike most of the other dialogs in SOLOMON, this dialog is ‘Modeless’ i.e. it can remain visible even when not in use. The dialog disappears when either its ‘Close’ button is pressed or when the dialog from which it was opened is terminated.

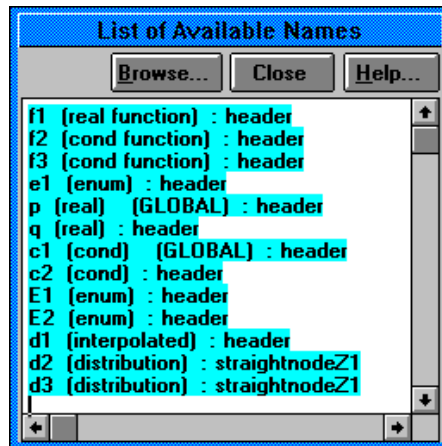


Figure 16: Listing of Available Parameters

The dialog offers the following features:

1. A read-only text window listing all the parameters and functions defined in all nodes preceding the current node. The parameters and functions are listed one per line. The first word is the name of the parameter. This word can be copied from the window using CTRL+C. The second part of the line is the parameter or function type, e.g. ‘(enum)’ for enumerated parameter or ‘(real function)’ for real-valued function. The last word is the name of the object in which the parameter or function was defined. Global parameters are identified by having ‘(GLOBAL)’ inserted before the object name.
2. A ‘Browse...’ button to allow the user to view the parameter or function definition. This opens another modeless dialog, described below.
3. A ‘Close’ button, to close the window and all sub-windows opened with the ‘Browse’ button.
4. A ‘Help...’ button to access the on-line help.

The ‘Browse...’ button opens a modeless dialog which contains a read-only text window showing the text used to define a parameter or function. The user first selects a parameter or function and then clicks on the ‘Browse...’ button. The dialog shows the name of the parameter or function, the name of the object, and the defining text. It also includes a ‘Close’

button to dismiss the dialog and a 'Help...' button to access on-line help. Text can be copied from the dialog using CTRL+C.

Up to 10 'Browse...' dialogs can be visible at once. They are closed when one of the following occurs:

1. The dialog's close button is pressed; or
2. The 'List parameters...' dialog is closed; or
3. The object definition dialog is terminated.

6.4.2 Define Parameters

This dialog provides an easy method for adding parameter definition statements to the SOLOMON input file. The user selects the location in the text window or preview window where the definition is to be placed. (Note that the definition will always be placed on a line on its own, therefore the location only needs to specify which line the inserted text is to precede. The default location is the first line.) The user then presses the 'Define Parameter...' button on the object definition dialog and the GUI opens the 'Define Parameter' dialog.

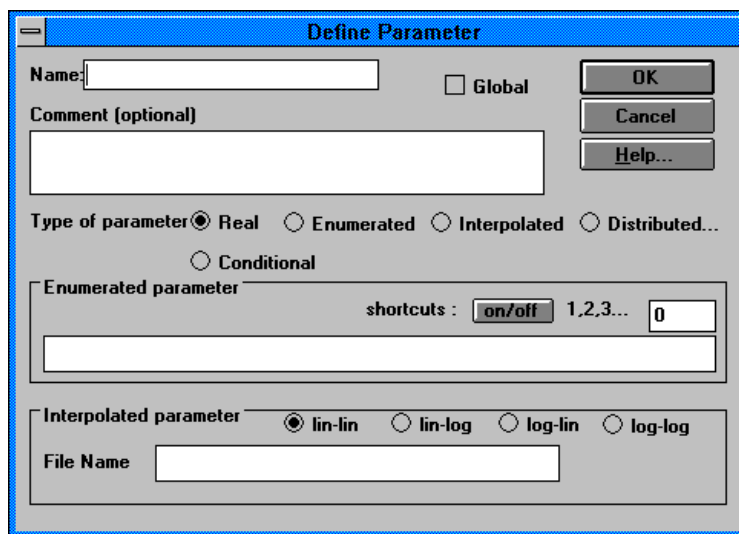


Figure 17: Defining a Parameter

The dialog offers the following features:

1. The name of the parameter. A character string up to 30 characters.
2. The type of parameter. The user selects one of the following radio buttons: Real (the default), Conditional, Enumerated, Interpolated and Distributed. If Distributed is chosen, the 'Define Distributed Parameter' dialog opens, described in Section 6.4.2.1.
3. A box for defining the values for an enumerated parameter. It is only necessary to enter data here if the type of parameter is enumerated. The names can either be

entered directly via the text window, or indirectly using one of the shortcuts provided. This is best described with an example. If the parameter name is 'param' then pressing on/off creates the value names param_on and param_off. Alternatively, entering '3' in the 1,2,3... window creates the names param_1, param_2, param_3. The value names can be altered by editing the text window. Note that changing the parameter name does not change the names of the values.

4. A box for defining an interpolated parameter. It is only necessary to enter data here if the type of parameter is interpolated. The user selects the type of interpolation from the radio buttons lin-lin (the default), lin-log, log-lin and log-log and enters the name of the file containing the interpolation data.
5. A check box indicating whether the parameter is global. This is only appropriate for real or conditional parameters.
6. An 'OK' button to transfer the generated text to the calling dialog.
7. A 'Cancel' button to return to the calling dialog without saving the changes.
8. A 'Help...' button to access on-line help.

6.4.2.1 Define distributed parameter

This dialog is entered when the user selects the 'Distributed...' radio button from the 'Define Parameter' dialog.

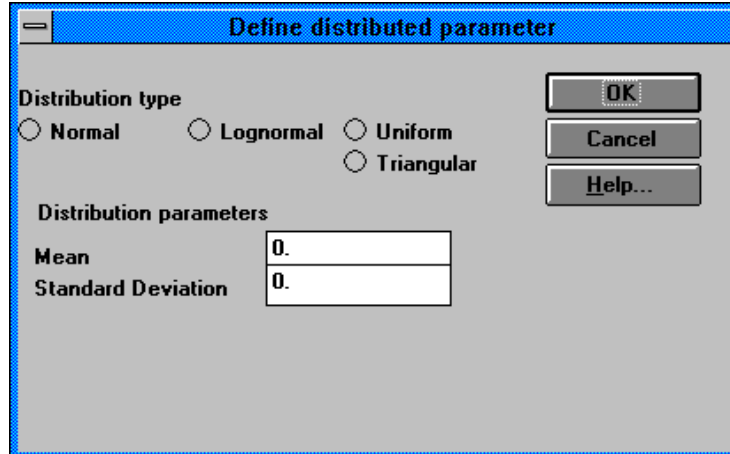


Figure 18: Defining a Distributed Parameter

The dialog offers the following features:

1. A set of radio buttons; Normal, Lognormal, Uniform and Triangular to select the shape of the distribution.
2. A text window to enter the mean of the distribution.
3. A text window to enter the standard deviation of the distribution (for the lognormal distribution, the log-standard-deviation).

4. An 'OK' button to transfer the generated text to the calling dialog.
5. A 'Cancel' button to return to the calling dialog without saving the changes.
6. A 'Help...' button to access on-line help.

6.4.3 Define Function

This dialog provides an easy method for adding function definition statements to the SOLOMON input file. The user selects the location in the text window or preview window where the definition is to be placed. (Note that the definition will always be placed on a line on its own, therefore the location only needs to specify which line the inserted text is to precede. The default location is the first line.) The user then presses the 'Define Function...' button on the object definition dialog and the GUI opens the 'Define Function' dialog.

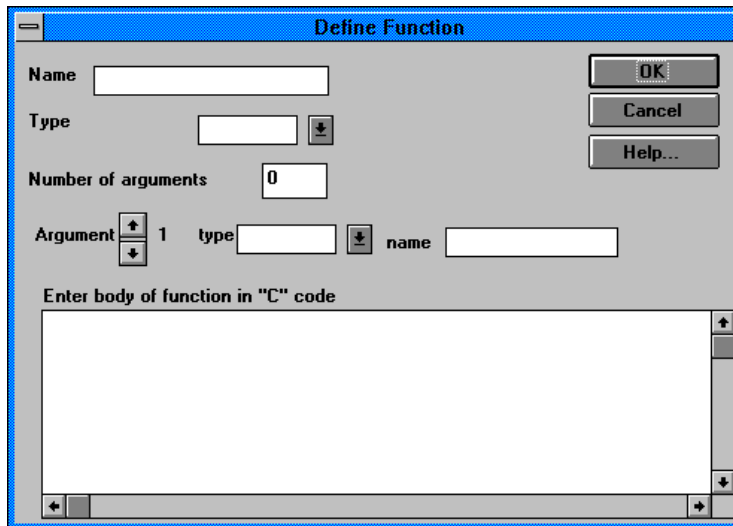


Figure 19: Defining a Function

The dialog offers the following features:

1. The name of the function. A character string up to 30 characters.
2. The type of function. The user selects either real or cond (conditional) from the drop-down selection box.
3. The number of arguments, default 0.
4. A selector for the argument number. The user presses the upward arrow to move to the next argument, and the down arrow to move to the previous argument. This is only appropriate when the number of arguments is greater than 0.
5. A drop-down selection box for the type of argument. The user selects from real, cond or node.

6. A text window for the name of the argument. A character string up to 30 characters.
7. A multi-line text window for entering the body of the function. The lines 'code' and 'end' are added automatically. See Section 5.3 for further details.
8. An 'OK' button to transfer the generated text to the calling dialog.
9. A 'Cancel' button to return to the calling dialog without saving the changes.
10. A 'Help...' button to access on-line help.

6.4.4 Set parameter value

This dialog provides a template for adding statements to set parameter values in the SOLOMON input file. The user selects the location in the text window or preview window where the statement is to be placed. (Note that the statement will always be placed on a line on its own, therefore the location only needs to specify which line the inserted text is to precede. The default location is the first line). The user then presses the 'Set Parameter Value...' button on the object definition dialog and the GUI opens the 'Set Parameter Value' dialog.

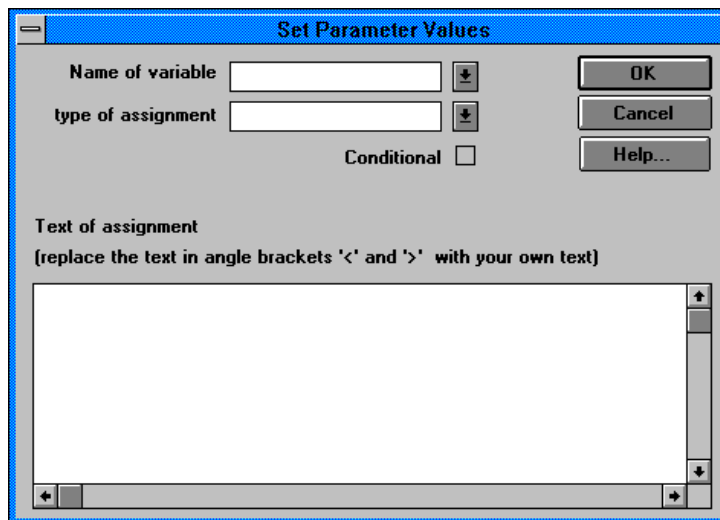


Figure 20: Setting Parameter Values

The dialog offers the following features:

1. The name of the parameter to be set. The user chooses a name from a drop-down selection box which includes all global real and conditional variables defined in preceding objects, and all real and conditional variables defined above the current location in the current object.
2. The type of assignment. The user chooses a template from a drop-down selection box. Selecting from this box causes the text window to be updated.

3. A check box indicating whether the assignment is to be conditional. If selections have been made from 1. and 2., altering the state of this check box causes the text window to be updated.
4. A text window to enter the full text of the set parameter statement. Actions 1,2,3 cause a template to be entered into this window. The user must edit this window replacing the text in angle brackets <> by appropriate text.
5. An 'OK' button to transfer the generated text to the calling dialog.
6. A 'Cancel' button to return to the calling dialog without saving the changes.
7. A 'Help...' button to access on-line help.

The dialog will ask the user for confirmation if action 2. or 3. will lead to overwriting any text in the text window.

6.4.5 Define Run Sequence

This dialog provides an easy method for adding run sequence definition statements, or system states, to the SOLOMON input file. It is only available from the header block. The user selects the location in the text window or preview window where the definition is to be placed. (Note that the definition will always be placed on a line on its own, therefore the location only needs to specify which line the inserted text is to precede. The default location is the first line.) The user then presses the 'Define Run Sequence...' button on the object definition dialog and the GUI opens the 'Define Run Sequence' dialog.

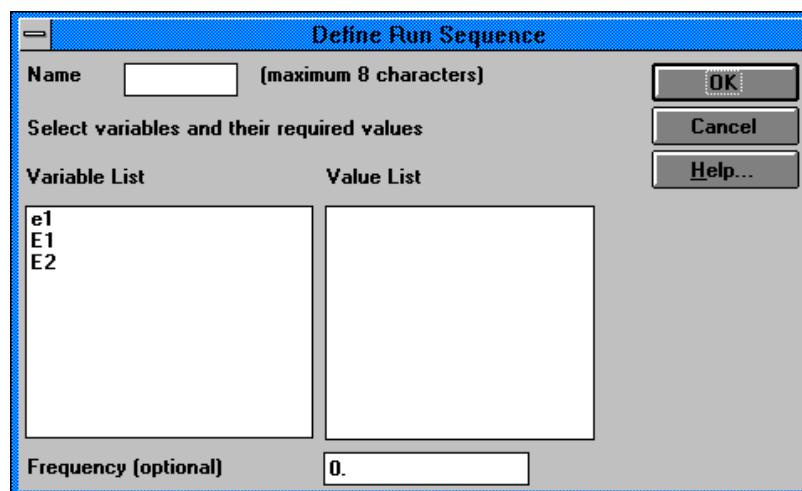


Figure 21: Defining the Run Sequence

The dialog offers the following features:

1. The name of the run sequence. A character string up to 8 characters. The limit on the name length is due to the fact that the name is used as a base in defining file names which, under Windows 3.1, are limited to eight characters.

2. A selection list containing the names of the enumerated parameters defined above the current location in the header block. The user selects one of these, which becomes the current parameter. Note that this action does not deselect previously-selected parameters. To deselect a parameter, the user clicks on it again.
3. A selection list containing the names of the values of the currently-selected parameter. The user selects one of these. Selecting from this list deselects every other value in this list.
4. A text window for specifying the frequency parameter for the run sequence. This is optional.
5. An 'OK' button to transfer the generated text to the calling dialog.
6. A 'Cancel' button to return to the calling dialog without saving the changes.
7. A 'Help...' button to access on-line help.

Operations 2. and 3. may be repeated as many times as required to specify the set of conditions for the run sequence or system state.

6.4.6 Show parameters

This dialog provides an easy method for adding 'show' statements to the SOLOMON input file. The user selects the location in the text window or preview window where the definition is to be placed. (Note that the statement will always be placed on a line on its own, therefore the location only needs to specify which line the inserted text is to precede. The default location is the first line.) The user then presses the 'Show...' button on the object definition dialog and the GUI opens the 'Show Parameters' dialog.

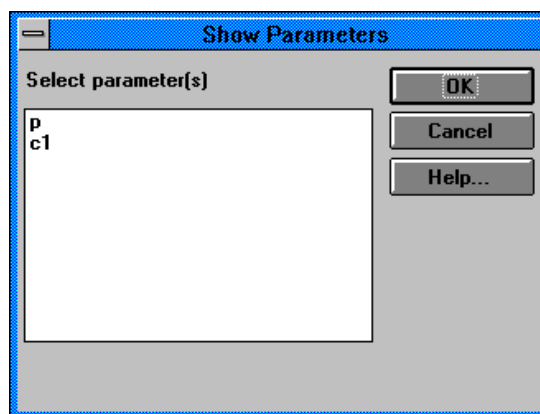


Figure 22: Defining Show Parameters

The dialog offers the following features:

1. A selection list containing the names of the parameters defined above the current location in the current object and the list of global parameters defined in preceding objects. This list includes all parameters whose value may have changed in the

current object, and hence includes all parameters where it may be desirable to insert a 'show' statement. The user selects any number of these, which become marked for showing. To deselect a parameter, the user clicks on it again.

2. An 'OK' button to transfer the generated text to the calling dialog.
3. A 'Cancel' button to return to the calling dialog without saving the changes.
4. A 'Help...' button to access on-line help.

6.4.7 Correlate

This dialog provides an easy method for adding 'correlate' statements to the SOLOMON input file. The user selects the location in the text window, or preview window, where the definition is to be placed. (Note that the statement will always be placed on a line on its own, therefore the location only needs to specify which line the inserted text is to precede. The default location is the first line). The user then presses the 'Correlate...' button on the object definition dialog and the GUI opens the 'Define Correlation Coefficient' dialog.

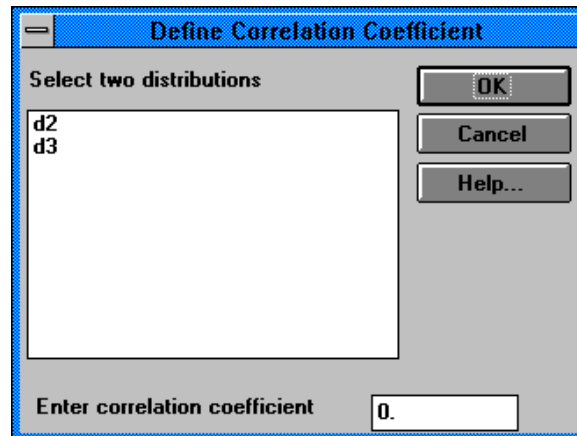


Figure 23: Defining Correlation Parameters

The dialog offers the following features:

1. A selection list containing the names of the distributed parameters defined above the current location in the current object. The user selects two of these. To deselect a parameter, the user clicks on it again.
2. A text window where the user enters the value of the required correlation coefficient between the two selected distributed parameters. The value must be a real number in the range [-1.0, +1.0].
3. An 'OK' button to transfer the generated text to the calling dialog.
4. A 'Cancel' button to return to the calling dialog without saving the changes.
5. A 'Help...' button to access on-line help.

6.5 SUBMITTING CALCULATIONS AND VIEWING RESULTS

6.5.1 Run SOLOMON

When the user has an input file ready to submit to SOLOMON, the user selects 'Run' from the editing window's menu, and then selects 'SOLOMON' from the resulting drop-down menu. The GUI will prompt the user to save the file if this has not already been done.

The GUI then performs some checking. As an example of the sort of errors the GUI is able to detect, if the 'endcategory default' is omitted from the SOLOMON input file the message "ERROR. Need at least an endcategory default definition." will be displayed.

If the GUI did not detect errors, the SOLOMON 'generate' step will then run. If there are no errors, SOLOMON will automatically proceed to each subsequent stage in the calculation. A message box keeps the user informed about the progress of the calculation.

If errors are found, the user is informed of this (and the stage where the failure occurred) via the message box. If more information is required, the user opens the appropriate log file (the SOLOMON log file is an amalgamation of all the log files) by selecting 'View / <log file name>' from the editing window's menu.

If the log file options are greyed out, they can be ungreyed by selecting an object on the left hand side of the window.

Note that because SOLOMON produces a large number of output files, with automatically-generated names, it is advisable to maintain different SOLOMON input files in different directories in order to prevent output files being over-written.

6.5.1.1 The current working directory

When the Solomon GUI is started the file open dialog will initially default to the directory containing the executable.

When the user selects an input file AND launches a job, the input file's directory will be recorded in 'solomon.ini'. The directory will be saved regardless of the success or failure of the launched job.

When the viewers are opened, they will check 'solomon.ini' for a last directory entry. If none is present, the viewers will point to their default directory.

If an entry exists, but the directory has been deleted the viewers will, again, point to their default directory.

If an entry exists (and that directory exists) it will be used by the viewers in their file selection dialogs.

6.5.1.2 Go to line

Occasionally, the generate step will fail, giving a message of the form 'SOLOMON (generate) ERROR at line <n>: <error message>'

The line <n> can be located in the GUI by selecting 'View / Go to line...' from the editing window's menu. This opens the 'Go to line' dialog which offers the following features:

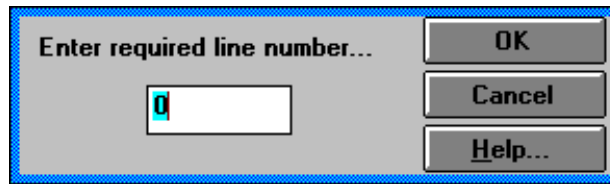


Figure 24: The 'Go To' Dialog Box

1. A text window to enter the required line number. The user enters an integer value.
2. An 'OK' button to open the appropriate object and highlight the required line.
3. A 'Cancel' button to return to the editing window.
4. A 'Help...' button to access on-line help.

6.5.2 The Supernode Viewer

6.5.2.1 Starting the supernode viewer

The supernode viewer is accessed by selecting 'Run / Supernode Viewer' from either the main window's menu or the editing menu's window. If the option is greyed out, it can be ungreyed by selecting an object on the left hand side of the window.

The supernode viewer's main window then opens, and the user sees the following options:

File Tree Page View

6.5.2.2 Selecting a file

The user selects 'File / Open' to open a file. The GUI provides a file selection dialog offering the user a selection '*.*' files. These files would normally correspond to the defined run sequences of the calculation. (If they appear to be missing, the user should check that the directory name in the file selection dialog corresponds to the directory in which the input file is located, and that the calculation did indeed run to completion without errors).

The user selects a file name; the GUI responds with a message indicating whether the file has opened.

6.5.2.3 Selecting a tree

The user selects 'Tree' from the main menu, and is offered the names of the supernode trees defined for that run. The user selects a tree, and the supernode viewer displays it.

The appearance of the tree can be enhanced by using the options in the 'Page' menu.

The whole tree can be made to appear larger or smaller using the options in the 'View' menu in the main menu.

The tree can be panned either horizontally or vertically using the scroll bars.

Text drawn in the body of the display uses the same font size, with supernode labels displayed in bold. Title text is displayed twice the font size of the body text.

6.5.2.4 Page options

The 'Page' menu options are:

- 'Copy' allows the user to capture the tree for inclusion in another application (e.g. Word). This copies the complete image to the Windows clipboard from which it may be pasted into another application. Note that the whole image is captured, regardless of the scaling options that may have been used.
- 'Width +', 'Width -', 'Height +' and 'Height -' allow the space between the displayed supernodes and branches to be tuned. These options only alter the display on the screen - the printed version is not affected.
- 'CharSize +' and 'CharSize -' allow the character size to be altered.

The 'Page' menu also contains options to control the display of the tree title, branch probabilities/labels and endcat probabilities.

- 'Branch Names' has three sub-options.
 - 'Visible' displays branch names.
 - 'Invisible' does not display branch names.
 - 'Minimal' only displays branch names required to avoid ambiguity. Labels are only displayed where branches are 'missing'. If a branching node has one or more branches with a probability of zero, those branches are not displayed. The remaining branches are displayed and labelled with their name and probability. Additionally, if any branching node has a single branch (which must have a probability of one) the probability is not displayed.
- 'Header' has two sub-options:
 - 'Visible' does display the title.
 - 'Invisible' does not display the title. The height of the rest of the display is increased to use the space that would have been used by the title.
- 'Endcat Probabilities' has three sub-options:

- ‘E-Format’ displays the endcat probabilities in E-Format (in a field width of 12 with 4 decimal places and a 3 digit exponent).
- ‘F-Format’ displays the endcat probabilities in F-Format (in a field width of 8 with 4 decimal places).
- ‘Variable-Format’ displays Endcat probabilities less than ‘endcat_thresh’ in E format. Values greater than or equal to ‘endcat_thresh’ are displayed in F format.

If the threshold setting is missing from the file ‘solomon.ini’, or the value is not sensible, the warning message "*Supernode viewer. Warning, could not find endcat threshold setting in file solomon.ini. Factory setting used.*" will be displayed.

- ‘Node Probabilities’ has three sub-options:
 - ‘E-Format’ displays the node probabilities in E-Format (in a field width of 9 with 2 decimal places and a 3 digit exponent).
 - ‘F-Format’ displays the node probabilities in F-Format (in a field width of 6 with 3 decimal places).
 - ‘Variable-Format’ displays Node probabilities less than ‘node_thresh’ in E format. Values greater than or equal to ‘node_thresh’ are displayed in F format.

If the threshold setting is missing from the file ‘solomon.ini’, or the value is not sensible, the warning message "*Supernode viewer. Warning, could not find node threshold setting in file solomon.ini. Factory setting used.*" will be displayed.

The ‘Page’ menu also contains options to save current settings for future use, to recall saved settings or to restore the default appearance:

- ‘Reset Page’ is used to recall the saved settings.
- ‘Save Settings’ is used to save the current settings (these settings are remembered between uses of the supernode viewer).
- ‘Standard Settings’ is used to restore the default appearance (this options does not change the values remembered by ‘Save Settings’; hence ‘Reset Page’ will still restore the saved settings).

6.5.2.5 View options

The ‘View’ menu options are:

- ‘Zoom in’ and ‘Zoom out’ alter the linear scale by a factor of approximately 1.2.

- ‘Fast Zoom in’ and ‘Fast Zoom out’ alter the linear scale by a factor of 2. The tree can be panned either horizontally or vertically using the scroll bars.

6.5.2.6 Exporting and printing

The ‘Page / Copy’ option enables the tree to be included in other packages.

The user may print the tree by selecting ‘File / Print’ from the main menu. The number of pages used depends on the width of the tree and the current zoom factor. If the tree has been zoomed using the ‘View’ menu, such that the whole width of the tree is no longer visible on the screen, then the tree will be printed on a number of pages. The number of pages can be estimated by taking the square of the number of times the screen needs to be scrolled in order to view the entire width of the tree.

If it is essential that the tree is printed on a single page, the user should ensure that the whole tree is visible on the screen when ‘File/Print’ is requested.

6.5.2.7 Finishing up

To view another tree in the same file, the user selects the tree from the drop-down menu ‘Tree’ from the main menu.

To open another file, the user selects ‘File / Open’ from the main menu.

To exit the Supernode viewer, the user selects ‘File / Exit’ from the main menu.

6.5.3 The PDF/CDF Viewer

6.5.3.1 Starting the PDF/CDF viewer

The PDF/CDF viewer is accessed by selecting ‘Run / PDF/CDF Viewer’ from either the main window’s menu or the editing menu’s window.

The PDF/CDF viewer’s main window then opens, and the user sees the following options:

```
File  I_tem  P_age  V_iew
```

The user selects ‘File / Open’ to open a file. The GUI provides a file selection dialog offering the user a selection ‘*.out’ files. These files would normally correspond to the defined run sequences of the calculation. (If they appear to be missing, the user should check that the directory name in the file selection dialog corresponds to the directory in which the input file is located, and that the calculation did indeed run to completion without errors.)

The ‘*.out’ files contain information corresponding to the endcats defined for the calculation. By clicking on the ‘List Files of Type:’ box on the bottom left of the file selection dialog, the user can display the list of variable files (*.sh1) instead. These files contain information corresponding to the ‘show’ commands in the input file.

The user selects a file name; the GUI responds with a message indicating whether the file has opened.

The PDF/CDF executable is specified in Solomon.ini. The 'pdfviewer' entry is set to 'c:\solomon.200\bin\pdfview.exe' in the supplied version of the software. You may wish to change this (to Microsoft Excel for example) if you wish to have more control over the format of the plots.

6.5.3.2 Viewing a variable or endcat

The user selects 'Item' from the main menu, and is offered a list of names. If the file was selected from the '*.out' file selection list, the list of names corresponds to the endcats defined in the input file. If the file was selected from the '*.sh1' file selection list, the list of names corresponds to the 'show' statements defined in the input file, and are of the form '<variable name> at <node name>'.

The user selects a name, and the PDF/CDF viewer displays two graphs. The first graph shows the cumulative distribution function of the endcat or variable, the second graph shows the Probability Distribution Function. The mean value of the variable is also displayed on each graph.

The display can be made to appear larger or smaller using the 'View / Zoom in' and 'View / Zoom out' options in the main menu. The tree can be panned either horizontally or vertically using the scroll bars.

The 'View' menu also offers 'Save' and 'Reset' options. 'Save' causes the GUI to remember the current amount of Zoom of the display. 'Reset' causes the display to revert to the last saved Zoom level, if the user had previously used 'Save', otherwise to its initial Zoom level.

6.5.3.3 Exporting and printing

The user may capture the display for inclusion in another application (e.g. Microsoft Word) by selecting 'Page / Copy' from the main menu, which copies the image to the Windows clipboard from which it may be pasted into another application.

The user may print the display by selecting 'File / Print' from the main menu.

6.5.3.4 Finishing up

To view another variable or endcat in the same file, the user selects the variable or endcat from the selection list given by clicking on 'Item' from the main menu.

To open another file, the user selects 'File / Open' from the main menu.

To exit the PDF/CDF viewer, the user selects 'File / Exit' from the main menu.

6.5.3.5 Limitations of PDF/CDF viewer

In certain circumstances, the PDF/CDF viewer can produce surprising results. These results may be caused by either of two main reasons :

Sample too small

For very small samples, (less than 5 values) there may be too few data points for the numerical methods in SOLOMON to produce meaningful results.

Limitations of the PDF algorithm

SOLOMON condenses the data points generated by all the samples to 15 data points per variable or endcat. The PDF viewer produces a graph from these points. Normally, this is adequate for viewing the overall shape of a distribution. The limitations of this approach are most apparent for the simplest distributions, the uniform distribution and the triangular distribution.

For the uniform distribution, the user would expect to see a perfectly-straight horizontal line for the PDF graph. Instead, there is often a noticeable unevenness in the line, especially at its edges. This is caused by SOLOMON attempting to put each data point into a category in order to calculate the density function. Because of the random nature of the raw data, SOLOMON calculates a density distribution which is not exactly uniform. This effect is most marked at the edges because the edges are plotted with more densely-packed data points. The effect diminishes with larger numbers of samples.

For the triangular distribution, the user may notice that the PDF plot has been ‘cropped’ at the centre, showing a flat-top distribution instead of a distribution with a point at the centre. This occurs because SOLOMON calculates the PDF line by differentiating the CDF line. This process loses some data, including the data for the centre of the distribution which contains the peak value.

6.5.4 The Output Viewer

6.5.4.1 Starting the Output viewer

The Output viewer is accessed by selecting ‘Run / Output Viewer’ from either the main window’s menu or the editing menu’s window.

The user specified viewer (Notepad by default) then opens.

The Output Viewer executable is specified in Solomon.ini. The ‘logviewer’ entry is set to notepad.exe in the supplied version of the software. You may wish to change this (to Microsoft Word for example) if the file size limitation of Notepad causes problems.

6.5.4.2 Finishing up

When the user exits the Output Viewer, the main SOLOMON GUI is again visible.

6.6 CUSTOMISING THE SOLOMON GUI

6.6.1 Supplied file

A number of GUI settings can be modified by changing entries in the file ‘solomon.ini’. The following table lists and describes the settings in the supplied file:

File entry	Description
[engine]	
system=c:\solomon.220	The directory where the SOLOMON software package is installed.
temp=c:\solomon.230\temp	Whilst SOLOMON is executing a number of intermediate files are generated. This setting specifies the directory for the intermediate files.
pdfviewer=c:\solomon.230\bin\pdfview.exe	The PDF/CDF viewer option uses this setting to launch the required application.
logviewer=notepad.exe	The Output viewer option uses this setting to launch the required application.
msvc=c:\msvc	The path to the Microsoft Visual C compiler.
[pdfview]	
zoom_percent=100	The zoom factor to be applied by the PDF viewer.
[Recent File List]	This list contains up to 4 entries. The entries are the last four SOLOMON input files loaded.
[logic]	
cols=190	Setting controlling the overall width of the display produced by the Supernode viewer. Changes to this setting must be made by editing 'solomon.ini'.
spacing=2	Setting controlling the overall height of the display produced by the Supernode viewer. Changes to this setting must be made by editing 'solomon.ini'.
charsize=8	Setting controlling the height of characters on the display produced by the Supernode viewer.
bnvis=1	Flag for display of branch names in the Supernode viewer: 1 - Always display branch names 2 - Never display branch names 3 - Minimal branch name display.
headvis=1	Flag for display of the header block in the Supernode viewer: 0 - Do not display header block 1 - Display header block.
endcat_format=1	Flag for display of the endcat probabilities in the Supernode viewer: 0 - Display in E format (field width of 12 with 4 decimal places). 1 - Display in F format (field width of 8 with 4 decimal places). 2 - Display in variable format.

File entry	Description
node_format=2	Flag for display of the node probabilities in the Supernode viewer: 0 - Display in E format (field width of 9 with 2 decimal places). 1 - Display in F format (field width of 6 with 3 decimal places). 2 - Display in variable format.
endcat_thresh=0.0001	Endcat probabilities less than 'endcat_thresh' are displayed in E format. Values greater than or equal to 'endcat_thresh' are displayed in F format. Changes to this setting must be made by editing 'solomon.ini'.
node_thresh=0.001	Node probabilities less than 'node_thresh' are displayed in E format. Values greater than or equal to 'node_thresh' are displayed in F format. Changes to this setting must be made by editing 'solomon.ini'.
viewport=100.000000	The zoom factor to be applied by the Supernode viewer.
[custom]	
LastWorkDir=C:\SOLOMON.230\TEMP	This entry records the directory and name of the file last run by SOLOMON.

6.7 DESIGN LIMITATIONS

The original version of SOLOMON was first developed on a UNIX platform. Whilst porting the product to a PC platform a number of the original limitations were removed. Some remain - the most noteworthy are described below.

6.7.1 Functional limitations

- You can only run one instance of the SOLOMON GUI.
- You can only run one instance of the Supernode viewer.
- You can only run one instance of the CDF/PDF viewer.
- The standard (for the compilers used to develop SOLOMON) for E-mode numbers is to display 3 digit exponents.

6.7.2 Limitation hardwired within the code

The calculational parts of SOLOMON have the following limits:

- The maximum lengths of character strings [STRINGLENGTH] (for example, for supernode names and branch names) is 30.
- The maximum number of nodes [MAX_NODES] is 600.
- The maximum number of endcategories [MAX_END_CATS] is 100.
- The maximum number of branches at each node [MAX_BRANCHES] is 10.
- The maximum number of supernode trees [MAX_SUPERNODE_TREES] is 20.
- The maximum number of data points in each interpolation data file [MAX_PARAMS] is 700.
- The maximum number of branch/show value pairs (for 1 sample) or sample values at branches (for samples greater than 1) [MAX_VALUES] is 10,000.
- The maximum length of a line in the input file [LINELENGTH] is 500.
- All log and error filenames are fixed within the code.

It should be noted that it is almost certain that it will not be possible/sensible to create a model using all of the maximum values. Other external considerations (such as the availability of memory and processing power) are likely to influence that maximum size of model created. In addition to the list above there may also be platform specific limitations (like the DOS 8.3 character file naming conventions).

7 References

Kernighan, Brian W. and Ritchie, Dennis M. (1988). "The C Programming Language." Second Ed. Prentice Hall.

Rudge, T., Robertson, M. (1992a). "A Review of PSAs Performed by AEA-T and comparison with NUREG-1150, German Risk Study and Sizewell B (WCAP) PSAs," HAD/P(92)5.

Rudge, T., Robertson, M. (1992b). "A Review of PSA Methodology and Available Computer Codes Within AEA-T and Recommendations for Future Development," HAD/P(92)9