U S E R ' S G U I D E

TDGEN

Version 3.4

Test Data Generator

SOFTWARE RESEARCH, INC.

This document property of:

Name:_____

Company:_____

Address:_____

_____

Phone_____

SOFTWARE RESEARCH, INC.

625 Third Street
San Francisco, CA 94107-1997
Tel: (415) 957-1441
Toll Free: (800) 942-SOFT
Fax: (415) 957-0730
E-mail: support@soft.com
http://www.soft.com

Documentation: Ann Kuchins

(Last Update November 19, 1997)

documentation/user-manuals/unix/advisor/97advisor.book/advisor.book/97tdgenux.b

Table of Contents

List of Figures

Preface

This preface explains how this user's guide is organized.

Congratulations!

By choosing the TestWorks integrated suite of testing tools, you have taken the first step in bringing your application to the highest possible level of quality.

Software testing and quality assurance, while becoming more important in today's competitive marketplace, can dominate your resources and delay your product release. By automating the testing process, you can assure the quality of your product without needlessly depleting your resources.

Software Research, Inc. believes strongly in automated software testing. It is our goal to bring your product as close to flawlessness as possible. Our leading-edge testing techniques and coverage assurance methods are designed to give you the greatest insight into your source code.

TestWorks is the most complete solution available, with full-featured regression testing, coverage analyzers, and metric tools.

Audience

This manual is intended for software testers who are using *TDGEN* tools. You should be familiar with the X Window System and your workstation.

Format of Chapters

This manual is organized to aid you after installation has been completed. (See the *Installation Instructions* if you are trying to install.)

This manual is divided into the following sections:

Chapter 1	*THE TEMPLAT FILE* describes *TDGEN's* template file, particularly its file structure, including details on the descriptors' format specifications.
Chapter 2	*THE VALUES FILE* provides *TDGEN* with the data to be supplied to the generated file output file.
Chapter 3	*TDGEN CALLS* describes how *TDGEN* is invoked from either the command line, ASCII menus or the X windows system's GUI (Graphical User Interface).
Chapter 4	*GRAPHICAL USER INTERFACE TUTORIAL* demonstrates using *TDGEN* in the OSF/Motif style X Window System environment.
Chapter 5	*TEMPLATE, BALUES FILE SYNTAX SUMMARY* shows the proper syntax for both *TDGEN's* template and values files.
Chapter 6	*TDGEN SAMPLES* provides examples that show several kinds of *TDGEN* use.

Identifying Special Text

This section explains the typographical conventions that are used throughout this manual.

boldface — Introduces or emphasizes a term that refers to TestWorks' window, its sub-menus and its options.

italics — Indicates the names of files, directories, pathnames, variables, and attributes. Italics is also used for manual, chapter, and book titles.

"Double Quotation Marks"

Indicates chapter titles and sections. Words with special meanings may also be set apart with double quotation marks the first time they are used.

`courier` — Indicates system output such as error messages, system hints, file output, and *CAPBAK/X*'s keysave file language.

`Boldface Courier`

Indicates any command or data input that you are directed to type. For example, prompts and invocation commands are in this text. (For instance, **stw** invokes TestWorks.)

CHAPTER 1

The Template File

This chapter describes *TDGEN*'s template file, particularly its file structure, including details on the descriptors' format specifications.

TDGEN offers a potent system commands capability via the command descriptor format {**%! <command>**}.

The **template** file tells *TDGEN* how to format the generated output file. Field names found in the template file should have a counterpart and appropriate entries in the **values** file, or else be one of the four pre-defined descriptors. Otherwise, if that field name is called for, an error message occurs. Execution will not halt, however; the field descriptor will simply not be reproduced in the output file, and the error message goes to `stderr`, leaving the output untarnished. This is the only format error in *TDGEN* that does not result in an immediate exit from the program.

TDGEN offers a potent system commands capability via the command descriptor format {**%! <command>**}.You can use a myriad of utilities available on the UNIX system to process the output file. You could also conceivably have a series of commands appending the template file to the output file and then calling *TDGEN* on that output file.

A sample template file follows below.

```
A line with an octal {%o i:normal } value.
A line with a floating number: {%8 f:spe-
cial }
Here is a brace { in the sentence.
A system command goes here: {%! echo
hello}.
A right-adjusted integer {%11 i:special }
only.
A string of blanks follows:{% blanks}.
A left-adjusted string {%-14 s:string }.
Floating point numbers {% f:normal}, {%-10
f:special}.
A: {%25 s:string }  {%c comment in tem-
plate file }
      B: {%-3 i:special }
```

```
                C: {% f:special }
          D: {% s:string }
                E: {%14 i:normal }
                    F: {%-7 f:normal }
End of example to "tdgen".
```

1.1 File Structure

The template file is a regular text file with variable fields called descriptors; these descriptors may be replaced by other text as specified by the user. Actual text is reproduced verbatim, and various descriptors of the form:

```
{%[<format spec>] <field name>}
```

are used to denote fields that may take on various values as enumerated in the values file. Sample entries in the template file may look like this:

```
Employee {% Employee# } is the person in
charge.
The job number is {% Job#}.
```

In the sample above, `Employee#` and `Job#` are both the field descriptors. Other text is reproduced verbatim in the output.

Any {% sequence, except in comments, is expected to conform to the format required of descriptors. The closing brace is mandatory.

1.1.1 Descriptor Format -- Field Name

The field name must be preceded by at least one blank, and may contain up to 40 characters. The permitted character set is any printing character. There is no embedded white space allowed in the field name except for new lines preceded by a backslash. Neither the backslash nor the new line appears in the output. A field name must appear in every descriptor except in the case where a system command is specified by the presence of the **!** character immediately following the **{%**.

1.1.1.1 Descriptor Format -- Format Specification

The format specification <format spec> has the following composition:

```
[field length] [conversion]
```

or:

```
system command
```

or:

```
comment
```

You can specify right or left alignment of the output, minimum field width, and octal or hexadecimal conversion of decimal integer input.

In addition, there is a mechanism provided for comments and system command execution. Here is a summary of the various components:

Field Length

Syntax:`[-]d`

Here, **d** represents a decimal integer. A preceding minus sign forces left alignment of the field; the default is right alignment. Examples of correct syntax:

```
{%5 name}
{%-10 name}
```

Conversion

Syntax: `d, o,` or `x`

Here **d**, **o** and **x** are decimal, octal or hexadecimal conversion respectively. If **o** or **x** is specified, then the input integer must be an unsigned decimal; otherwise the results of the conversion will be unpredictable. Some examples of correct syntax:

{%x hexnum}

{%10o i:normal}

System Command

Syntax: `! <system call>`

Any sequence of characters after the **!** and before the closing brace **}** will be regarded as a system call. Its output will go to `stdout` unless an error is encountered. Some examples of correct syntax:

```
{%! ls}
{%! echo hello}
```

Comment

Syntax: `c <command>`

All characters after the **c** will be ignored; scanning will start again after the closing brace } is found. Comments are excluded from the output file. They may contain an embedded new line if it is preceded by a backslash. An example of a comment is:

```
{%c This is some comment.}
```

1.1.1.2 Pre-defined Descriptors

There are four pre-defined descriptors which, when used, should not be identified in the values file. If they are identified in the values file, they will be processed as user-defined descriptors. An example of each of the four descriptors is given below:

```
{%-10 ascii 4}

{% alpha 150}

{% decimal 160}

{% real 3.6}
```

Pre-defined descriptors have the following form:

```
{%[<format spec>] <field name> <field
size>}
```

Here is a summary of the various components:

```
<format spec>
```

As with user-defined descriptors, the field name may be preceded by a format specifier. The **d**, **o** and **x** options are not allowed.

```
<field name> : ascii, alpha, decimal or
real
```

The **ascii** set includes all printable characters. The **alpha** set includes the characters <a-z>,<A-Z> and <0-9>. The **decimal** and **real** sets include the characters <0-9>.

```
<field size>
```

For **alpha**, **ascii**, and **decimal** the valid field sizes are 1 through 254. For **real**, the valid field size takes the form of "n.m" where ((n+m) < 254)) and (n >= 0) and (m > 0).

1.1.1.3 Escapes

Arbitrarily long lines can be produced by escaping the new line character; this is done by preceding it with a backslash. Both the new line and the backslash are ignored. The backslash receives no special handling otherwise.

For example, if this line appeared in the **`template`** file:

```
{%-10 retailer} {%-10 cost} {%-10 retail}
{%-10 net} \
{%-10 discount} {%-10 date}
```

the output would be on one continuous line, and the backslash would be omitted.

CHAPTER 2

The Values File

The values file provides *TDGEN* with the data to be supplied to the generated output file.

2.1 Sample Values File

The field names that appear in the values file should only appear once. There is no error checking for uniqueness. If a field name should appear twice, only the first occurrence and its associated values will be considered. The maximum allowable number of distinct field names is 1024. A sample values file follows below.

```
{%c Sample values file.}

i:normal{%r 1..100}
i:special0 1 2 3 4 5
f:normal2.1753.1244.765
f:special99E09-8E101
s:stringa  ab  abc  abdc  abcde
blanks{%1} {%2} {%3} {%5}
```

2.2 File Structure

The *TDGEN* **values** file is a regular text file containing field names and their potential values. You have a number of options for giving values in the *TDGEN* values file. *TDGEN* uses a data table to associate particular values to field names that you write into the template file. Sample entries in the values file may look like this:

```
<field names><values>

Job#1001 1002 1003 1004 1005
Employee#5006 5007 5008 5009 5010 5011.
```

All field names, for example, **Job#**, **Employee#**, must be unique.

2.2.1 Field Name Specifications

The field names are assumed to begin with the first non-white character that follows a new line, and to terminate with blanks or tab characters. If all the values associated with a field name cannot be written on one line, then the new line character must be preceded by a backslash. The only exception to this new line rule is the format for comments, detailed in Section the section on that topic (See Section 3.2 - "Specific Invocation" on page 15.).

Embedded white space in field names is not allowed. Each field must have at least one value following it; otherwise, an error occurs. Individual values are separated by white space; the only mechanism by which white space can be considered part of a value is if it is preceded by a backslash (See Section 3.2 - "Specific Invocation" on page 15.).

The field names contained in the values file should have a counterpart in the template file. Otherwise, if a value is specified for that field on a call to *TDGEN*, an "unknown type" error message occurs. An example of what might be in the template file corresponding to the example for the values file given previously is:

```
Employee : {%-20o Employee# }
Job :      {%10d Job#}
```

Field names in the template file always start with **{%** and end with**}**. All other text is reproduced in the output verbatim.

2.2.2 Ranges of Values, Comments, Escapes, and Blanks

TDGEN provides ways to handle blanks and other white space characters, and also provides for comments.

Ranges of Values

```
Syntax:name  {%r n1..n2}
```

The user can save time and effort using the range specification for integer values. This format generates *TDGEN* values starting at **n1** and incrementing by 1 until **n2** is reached. An example is:

```
number  {%r 1..9}
```

which is equivalent to

```
number 1 2 3 4 5 6 7 8 9.
 Comments
Syntax:{%c ...<comment text>...}
```

values file may contain comments, and the individual values may contain blanks. It may contain new line characters only if they are escaped with a backslash. Comments may not be embedded in the field name or data. An example is:

```
{%c Here are some \
comments}
```

Escapes

The backslash "\\' is considered special if it appears prior to these characters:

```
"  "- escape blank
"\"- output a backslash
```

Other than in those cases listed above, and in the case where it is embedded in a field name, the backslash has no effect and is not reproduced in the output. Some examples are:

```
{%\ name}{%\ midname}
```

Blanks

```
Syntax:{%d}
```

A blank or tab within a value item can be escaped with "\\'; this appears on the value as " " and circumvents the use of white space as the separator. The sequence **{%d**}, where **d** is a positive integer, expands to a value that contains exactl**y d** blanks. This allows the possibility of having a string of blanks as an item. An example is:

```
{%5}
```

which gives five blanks.

CHAPTER 3

TDGEN Calls

This chapter describes how *TDGEN* is invoked from either the command line, ASCII menus, or the X windows system's GUI (Graphical User Interface).

There are several different ways to invoke *TDGEN*. You can select values from the **values** file specifically or randomly. You can also invoke *TDGEN* sequentially, generating every possible combination of the given values. You can use *TDGEN* either with command line options, with ASCII menus, and with X Window System graphical user interfaces (GUI). The menus are described at the end of this chapter. The GUIs are described in .(See CHAPTER 4 - Graphical User Interface Tutorial" on page 37.)

You can tabulate the number of possible test data combinations that *TDGEN* generates. You can then study the table to see if you want to adjust the number of combinations before actually running *TDGEN* by making necessary modifications to the values file. *TDGEN* further gives you the option of invoking it as many times as you wish. Invoking it repeatedly allows you to automatically run a specified number of combinations of the test. *TDGEN* may also be invoked to access standard input in place of the template file. Default output is standard output. You can specify output to go to a file, if you wish.

3.1 Call Summary

```
tdgen values <template> n1 n2 n3 n4...nn
tdgen -r values <template>
tdgen -R <number> values <template>
tdgen -s values <template> sfile
tdgen -S <number> values <template> sfile
tdgen -T <-r> values <template>
```

There is one global switch, **`-g file`**, that must appear as the last argument on the command line. The **`-g`** switch, along with all other switches, is explained later in this section.

3.2 Specific Invocation

The values for all or some of the descriptors are specified by integers following the template filename. The same examples from the first parts of of later chapters, reprinted here, help illustrate an instance of a specific invocation. The example template file and values file follows.

```
This is an example template file for
“tdgen”.

A line with an octal {%o i:normal } value.
A line with a floating number: {%8 f:spe-
cial}
Here is a brace { in the sentence.
A system command goes here: {%! echo
hello}.
A right-adjusted integer {%11 i:special}
only.
A string of blanks follows:{% blanks}.
A left-adjusted string {%-14 s:string}.
Floating point numbers {% f:normal}, {%-10
f:special}.
A: {%25 s:string}   {%c comment in tem-
plate file}
      B: {%-3 i:special}
            C: {% f:special}
          D: {% s:string}
              E: {%14 i:normal}
                  F: {%-7 f:normal}

End of example to "tdgen".

{%c Sample values file. }

i:normal{%r 1..100}
i:special0 1 2 3 4 5
f:normal2.1753.1244.765
f:special99E09-8E101
s:stringa  ab  abc  abdc  abcde
blanks{%1} {%2} {%3} {%5  }
```

Using these two files, type:

```
tdgen values template 10 1 3 4 5 6 7 8 9 10
1 2 3 4
```

The integer in position 10 refers to the first descriptor found in the template file: **`i:normal`**. *TDGEN* looks under **`i:normal`** in the data table, and outputs the 10th value listed, 10. In this case, the value will be con-

verted to octal, as indicated by the **o** after the '%' sign in the **template** file. Likewise, the integer in position 1 refers to the second descriptor in the file, **f:special**. The first value is output, 99E09.

Descriptors with the same field name may appear more than once in the template file but not in the values file. For example, **f:special** appears three times in the **template** file but only once in the **values** file.

If there are more integers in the invocation than there are descriptors in the **template** file, the unmatched integers will be discarded. Similarly, if there are less integers in the invocation than there are descriptors, then the extra descriptors will be treated as regular text and output verbatim.

If the value of any integer exceeds the number of total possible items associated with the descriptor, the last item will be output.

Note that pre-defined descriptors are not listed in the values file. Therefore, any number may be used to specify the inclusion of one in the specific invocation of *TDGEN*. The descriptor is randomly generated. Continuing with the same example, the generated output file is shown below.

```
This is an example template file for
''tdgen''.

A line with an octal 12 value.
A line with a floating number:   99E09
Here is a brace { in the sentence.
A system command goes here: hello.
A right-adjusted integer           3 only.
A string of blanks follows:     .
A left-adjusted string abcde           .
Floating point numbers 4.765, -8E101.
A:                      abcde
     B: 5
     C: 99E09
         D: ab
     E:              3
        F: 4.765

End of example to "tdgen".
```

3.3 Random Invocation

The correct syntax for the random option is:

```
tdgen -r values <template>
tdgen -R <number> values <template>
```

The **-r** or **-R** switch notifies *TDGEN* to take one value from each field in the **values** file at random. For each field name encountered in the **template** file, a uniformly distributed random number is used to select a particular value from those corresponding to that field name. The distribution can be weighted towards certain values by enumerating them several times in the values file. For example, an entry in the values file such as:

```
cost 2.50   3.75   1.25   1.25   1.25
```

makes "cost" three times more likely to be 1.25 than 2.50.

The **-r** option invokes *TDGEN* once. The **-R** option invokes *TDGEN* repeatedly, as defined by <number> switch. The default is 1, so that

```
tdgen -R values template
```

invokes *TDGEN* once.

An additional switch, **-R 0**, runs *TDGEN* indefinitely, until you interrupt it. See a later section for omitting the optional template file (See Section 3.6 - "Invocation without template File" on page 21.).

3.4 Sequential Invocation

Use the sequential invocation repeatedly to generate exhaustively distinct data sets. There are two ways to invoke *TDGEN* sequentially:

```
tdgen -s values <template> sfile
tdgen -S <number> values <template> sfile
```

sfile is a user-supplied file that lists integers. Integers should be separated by a space.

The **sfile** contains the integers n1 ... nn. If you give a non-existent file name, *TDGEN* acts as though it has been given a file of 1's for all the descriptors found in the template file. It will then write the next sequence of integers into that file according to the following scheme:

1. Start with the left-most integer.
2. Increment the integer.
3. If this value is larger than the number of values associated with the first descriptor found in the template file, *TDGEN* will replace it with a 1 and increment the next integer.
4. Steps 2 and 3 are repeated until either some value has been incremented or all the integers are exhausted.

Continuing with the same example from a later section (See Section 4.2 - "Using TDGEN" on page 39.), when you invoke *TDGEN* with the **-s** option, using the following values in **sfile**:

```
100 2 3 4 1 1 1 1 1 1 1 1 1 1
```

After execution, **sfile** reads:

```
1 3 3 4 1 1 1 1 1 1 1 1 1 1
```

This shows how *TDGEN* "increments" the **sfile** to remember the inputs for the next invocation of the system.

If **sfile** is non-existent, *TDGEN* acts as though it had been given a file of 1's; it also creates a file, **sfile**, containing the next sequential call, as described above.

The sequential option can be used repeatedly to generate exhaustively distinct data sets. Note that *TDGEN* expects only digits in the **sfile**; other text, if present, will produce unpredictable results. The **-s** option invokes *TDGEN* once, and the **-S** option invokes *TDGEN* repeatedly, as defined by **<number>** switch. The default is 1, so that:

```
tdgen -S values template
```

invokes *TDGEN* once. An additional switch, **-S 0**, runs *TDGEN* until you interrupt it. See the section that describes omitting the optional **tem-**

plate file (See Section 3.6 - "Invocation without template File" on page 21.).

3.5 Table Summary

You may want to see how many combinations you have specified *TDGEN* to generate before you generate actually them. The -T switch calculates the number of combinations of all fields in the values file. The correct syntax is:

```
tdgen -T (-r) values <template>
```

Continuing with the example from above, typing:

```
tdgen -T values template
```

would give you the following table:

Field	No. Table Entries	Cumulative Total Combinations
% i:normal	100	100
% i:special	6	600
% f:normal	3	1800
% f:special	2	3600
% s:string	5	18000
% blanks	4	72000

TABLE 1 Field Values for TDGEN

The total number of combinations is 72,000 for **<number>**. If you use the sequential invocation mode *TDGEN* will exhaustively generate and test all 72,000 data combinations. You may not want to generate all combinations at once, so you should make adjustments to the fields accordingly.

3.6 Invocation without template File

If you invoke *TDGEN* without specifying a template file, standard input is accessed. Possible invocations are:

tdgen -<s|S> values sfile

tdgen -<r|R> values

tdgen values 1 2 3 4 5 13 2 1 3 3 3

tdgen -T values

After you type in the invocation and press **Return**, *TDGEN* stops and waits for you to type in the field descriptors. An example would be:

```
{% name}{% age} Press Return when you have
entered all desired field descriptors.
TDGEN processes the data, but you must
type the end-of-file character for your
system to terminate the session.
```

Not using a template file allows you to use piping, as shown in the following example:

```
capkey -f file | tdgen -r values | appli-
cation
```

3.7 Calling TDGEN from a Script

TDGEN also returns the number of values substituted into the field descriptors. This simplifies some kinds of scripting. Here is an example in the UNIX context:

```
tdgen -r values template -g temp
while test $? -gt 0
do
cp temp template
tdgen -r values template -g temp
done
```

This would have the effect of running *TDGEN* until it made NO conversions, thus simplifying operation in some cases.

3.8 Redirecting Output

Default output goes to the screen. If you want output to go to a file, use the **`-g file`** switch. The **`-g file`** switch must be the last argument on the command line. Use it with any invocation of *TDGEN*. For example, if you wanted output from a certain test to go to a file called **`File1`**, type:

```
tdgen -r values template -g File1
```

3.9 TDGEN Menus

Menus help users in two ways: by providing a fixed structure for collecting test coverage information, and by providing a convenient way to customize a sequence of operations.

3.9.1 Invoking TDGEN

Start *TDGEN*'s menus with the command:

```
tdgen
```

TDGEN uses the default configuration file, **tdgen.rc**. You can change any settings during an invocation of *TDGEN*. To save any changes you make during a session, save the settings upon exiting *TDGEN*.

3.9.2 Menu Tree

The menu tree is shown below.

```
TDGEN

MAIN:
|          Selects Actions, Files or Options menu
|          Shows option settings
|          Shows current execution statistics
|          Saves option settings
|          Exit from TDGEN
|          On-line help frames
|          !<system commands>
|
|____ACTIONS:
|          Selects basic TDGEN operations
|          Shows option settings
|          Returns to prior menu
|          On-line help frames
|          !<system commands>
|
|____OPTIONS:
|          Helps select all user-settable options
|          Shows option settings
|          Returns to prior menu
|          Sets showmenu flag
|          On-line help frames
|          !<system commands>
|
|______FILES:
             Shows all current file settings
             Allows changing file settings
             Returns to prior menu
             On-line help frames
             !<system commands>
```

After *TDGEN* starts, you will see the title information, version control indication, and the prompt: **TDGEN**:.

To see available menu options, type from any prompt within *TDGEN*:

```
?
```

TDGEN then displays the available options for that menu. This feature works for all menus throughout *TDGEN*.

The current menu is redrawn whenever you give an unrecognized command.

3.9.2.1 Issuing Commands

You can issue commands by typing the first few letters of each command's name. The only requirement is that the letter sequence be unique to that command. *TDGEN* will inform you when a command you issue matches two or more possible commands.

To set variables [See the **Files** menu description(See Section 3.9.5 - "Files Menu" on page 30.)], you must type the entire variable name. This is done in order to be consistent with configuration file processing.

3.9.2.2 Displaying Current Parameter Settings

You can display the current settings of options and file names known to *TDGEN* at any time using the settings command, get on-line help with the help command, and exit the current menu using exit. The configuration file read in the settings is automatically used.

3.9.2.3 TDGEN Menu Stack

When you move from the main menu to any other menu, *TDGEN* remembers the sequence of your choice of menus in an internal "stack". The stack lets you return to the menu you were just in by typing the exit command. This feature is provided to prevent you from entering conflicting or incorrect data during a run.

If you wish, you can issue a series of exit commands that will eventually return you to the main menu to exit *TDGEN*. That is, your moves between the three subsidiary menus are "stacked" and must be "unstacked" before returning to the main menu.

Pressing the **Interrupt** key returns you immediately to the main menu.

3.9.3 Main Menu

When you invoke *TDGEN*, the following menu is displayed:

```
TDGEN:MAIN:
Options:
       release   -- Show release and ver-
sion number

       actions     -- Go to the ACTIONS
menu
         files     -- Go to the FILES menu
       options     -- Go to the OPTIONS
menu

      settings     -- List the current
settings for TDGEN option
   help [opt]      -- Display HELP text
for a command

          save     -- Save the current
settings for TDGEN
          exit     -- Exit from TDGEN to
system
```

3.9.4 Actions Menu

The Actions menu is displayed below:

```
TDGEN:ACTIONS:
Options:
       release   --Show release and ver-
sion number
         files   -- Go to the FILES menu
       options   -- Go to the OPTIONS menu
            go   -- Execute TDGEN
  infinite go   -- Execute TDGEN until
interrupted
  go <number>   -- Execute TDGEN <num-
ber> of times
      settings   -- Display current runt-
ime settings
    help [opt]   -- Display HELP text for
command
          exit   -- Exit current level
```

3.9.5 Files Menu

The Files menu is displayed below:

```
TDGEN:FILES:
Options:
release     --   Show release and version
number
actions     --     Go to the ACTIONS menu
options     --   Go to the OPTIONS menu

values<file>--  File for values (default
values)
template <file>--File for template
(default template)
sequence <file>-- File for sequence
information (default Seq)
output <file>-- File for output (default
standard output)
input <file>--  File for user values
(default standard input)

settings--Display current runtime set-
tings
help [opt]--Display HELP text for com-
mand

exit        --   Exit current level
```

3.9.6 Options Menu

The Options menu is displayed below:

```
TDGEN:OPTIONS:
Options:

  release     --   Show release and version
  number

  actions     --   Go to the ACTIONS menu
  files       --   Go to the FILES menu

  random      --   Use random generation
  mode
  sequential  --   Use sequential genera-
  tion mode
  table       --   Output the summary table
  settings    --   Display current runtime
  settings.
  help [opt]  --     Display HELP text for
  a command

  exit        --   Exit to the system
```

3.9.7 Saving Changed Options Settings

Before leaving *TDGEN*, you are prompted about saving the current settings. Answer **`y`** to the prompt, and your settings overwrite the **`tdgen.rc`** file.

You can also change current settings in the main menu with the **`save`** command.

3.9.8 Running System Commands

You may issue a command directly to the operating system by using the **!** symbol, as follows:

```
TDGEN: !<any system command>
```

TDGEN regains control after the command is executed. This feature is useful for editing files and for other activity related to a *TDGEN* session. For example,

```
! cat values
```

lets you view the values file.

3.9.9 TDGEN Configuration File

TDGEN reads the configuration file before beginning any processing. The configuration file may contain modifications to the default settings of a variety of *TDGEN* parameters.

TDGEN's configuration file is a simple ASCII text file that you can modify with an editor. You can also create a new configuration file with the **Save** option in *TDGEN*'s main menu.

A typical example of *TDGEN*'s default configuration file is shown below.

```
# Parameters

help="/usr/lib/tdgen/tdgen.hlp"
execute="tdgen"
values="values"
template="template"
sequence="Seq"
output=""
input=""
opts="-R"
times=1
noshowmenu
```

You can change any of the run-time parameters, except for `help` and `execute`. Configuration file lines can contain any set of commands in any order. Comment lines must begin with #. All white space, tabs and blanks are ignored, except those appearing within quotes.

Configuration file entries are explained below, listed with default values.

```
# <comment>
```

A line beginning with # is treated as a comment.

```
help="/usr/lib/tdgen/tdgen.hlp"
```

This parameter gives the path where the install script places the on-line help frame.

```
execute="tdgen"
```

This parameter defines the command *TDGEN* accepts for execution. It is equivalent to the go option in the Actions menu.

```
values="values"
```

This parameter names the user-supplied values file.

```
template="template"
```

This parameter names the user-supplied template file.

```
sequence="Seq"
```

This parameter names the file *TDGEN* calls when you use the sequence option in the Files menu.

```
output=""
```

This parameter specifies where output goes. The default is standard output. If you want output to go to a file, specify a filename. In successive outputs, *TDGEN* overwrites the file unless you specify another filename.

```
input=""
```

This parameter specifies where the input comes from. The default is standard keyboard entry. If you want to use a file as input, specify that filename, which must be located in the current directory. The file has the same function as the list of integers for field descriptors in the example

```
tdgen values 1 2 3 2 12 32.
```

```
opts=""
```

This parameter specifies the switch that *TDGEN* uses when you type `go`. The default is null: *TDGEN* executes without any options.

```
times="1"
```

This parameter specifies the number of times *TDGEN* is executed at once. It is equivalent to the number in the `-R` number switch.

```
noshowmenu
```

This parameter determines whether the entire menu is re-drawn on the screen when a command is issued. You will probably prefer to use `noshowmenu` after you are familiar with the program.

CHAPTER 4

Graphical User Interface Tutorial

This chapter demonstrates using ***TDGEN*** in the OSF / Motif style X Window System environment.

4.1 Invocation

To invoke type:

```
Xtdgen
```

The result is the main menu (See Figure 1 "Xtdgen Main Menu" on page 37.). It has a window menu button (available for all windows) that allows you to restore, move, size, minimize, maximize, or to lower the window and to close the program. This menu button can be used at any time during the X Window System program. The two buttons in the upper right hand corner of the window allow you to maximize or to minimize the window.

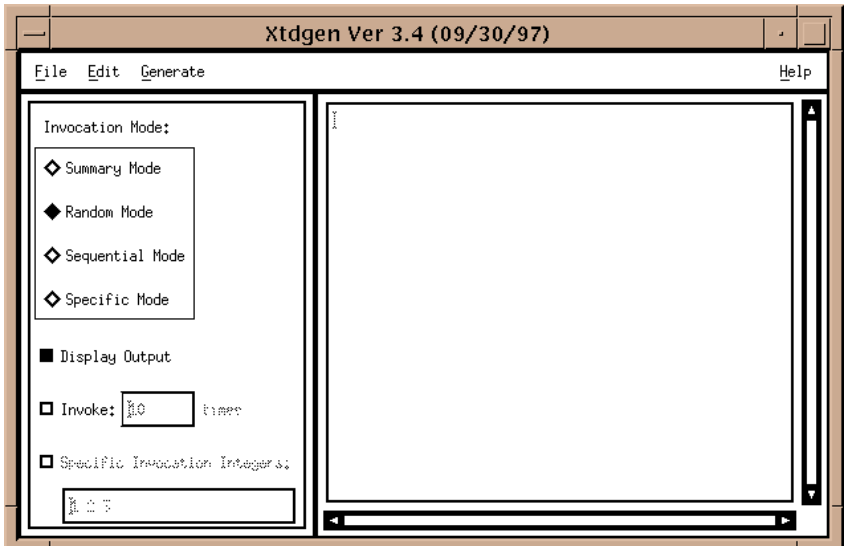

FIGURE 1 Xtdgen Main Menu

To invoke with **STW/Advisor**, click first on **Advisor** and then on **TDGEN**. The **TDGEN** main menu pops up.

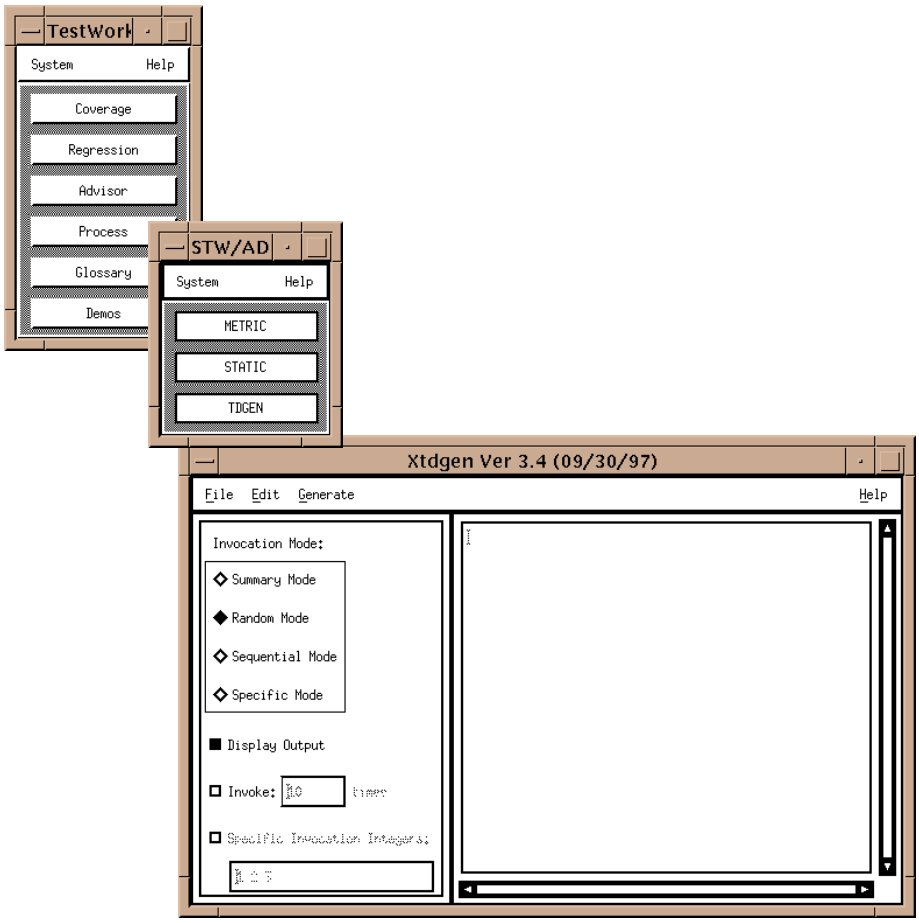

FIGURE 2 STW/Advisor Invocation

4.2 Using TDGEN

4.2.1 Main Menu Options

All ***TDGEN*** operations can be performed from the main menu.

1 For first time use, always read the help frames. Below is main menu's help frame, explaining ***TDGEN***'s main functions.

NOTE: This is the only help frame available.

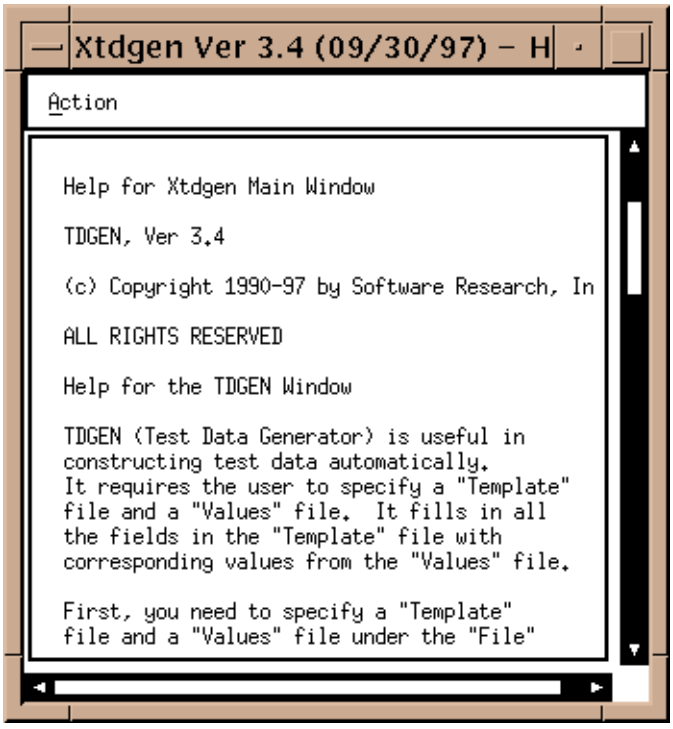

FIGURE 3 Main Menu Help

2. **TDGEN** can be invoked with the following modes:
   - "Summary Mode" provides data that can be generated for statistics. (See Figure 8 "Summary Mode Output" on page 45.)
   - "Random Mode" generates data randomly. (See Figure 9 "Random Mode Output" on page 46.)
   - "Sequential Mode" generates date sequentially. (See Figure 10 "Sequential Mode Output" on page 46.)
   - "Specific Mode" generates date as specified in the "Specific Invocation Integer" option (described below). (See Figure 11 "Specific Mode Output" on page 47.)
3. ***TDGEN*** also offers the following options:
   - "Display Output" displays the file output. You can move up, down, and sideways in the display using the scroll bars. This option is available for all invocation modes.
   - "Invoke" allows you to choose how many sets of data are to be generated. If no number is chosen, then it assumes only **one** set of data is specified. This option is available for only "Random Mode" and "Sequential Mode".
   - "Specific Invocation Integers" generates data according to what integers you specify in this option. This option is available for "Specific Mode" only.

Following is a step-by-step example.

4. Click on the "File" pull down menu and select the appropriate files.
   - "Set Template File" must be set. Type in your new file in the Selection Box or highlight (if you already have the template file in the directory) and then click **OK**. If not selected, then you will be unable to edit later. File filter allows you to specify the filtered keysave files to be displayed. (See Figure 4 "Set Output file window" on page 42.)
   - "Set Values File" must be set. Selection is made the same way as the template file. (See Figure 5 "Set Value File Pop-Up Window" on page 43.) If not selected, then you will be unable to edit later.
   - "Set Sequential File" is only necessary for "Sequential Mode". If not selected, then you will be unable to edit later. The default is set to sfile. (See Figure 16 "Edit Sequential File" on page 50.)

- “Set Output File” is the file where everything is written to, so it can be viewed later. This file can be set at any time during your program. (See Figure 12 "Set Editor Command Message Window" on page 48.)

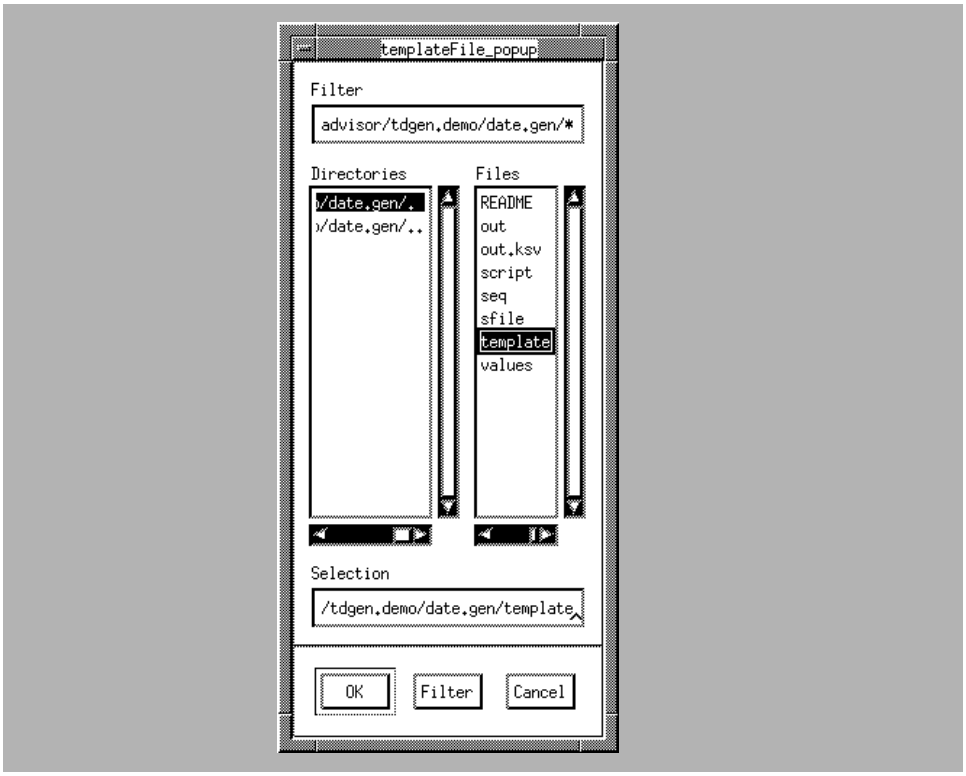

FIGURE 4 Set Output file window

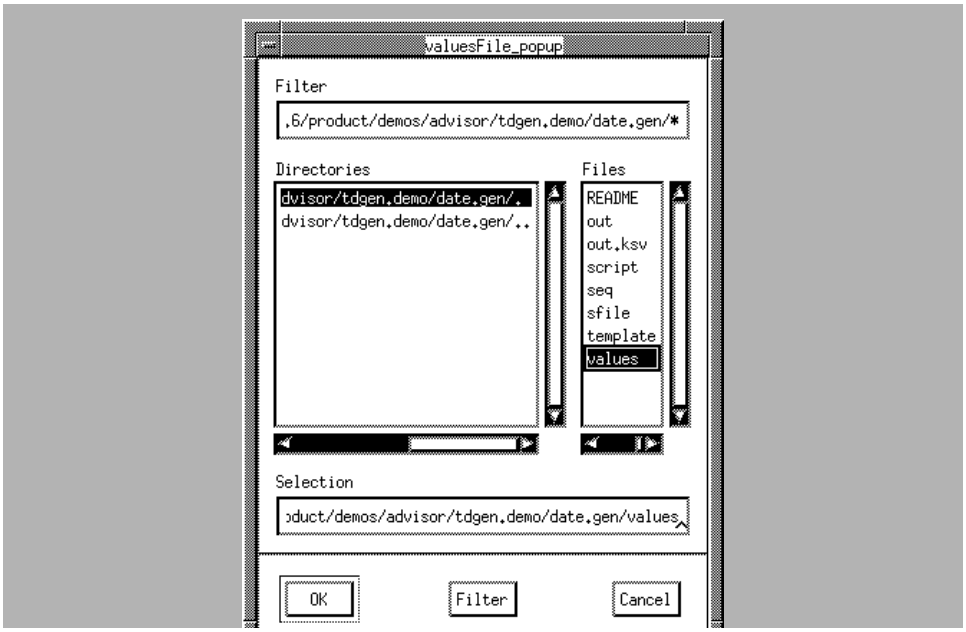

FIGURE 5 Set Value File Pop-Up Window

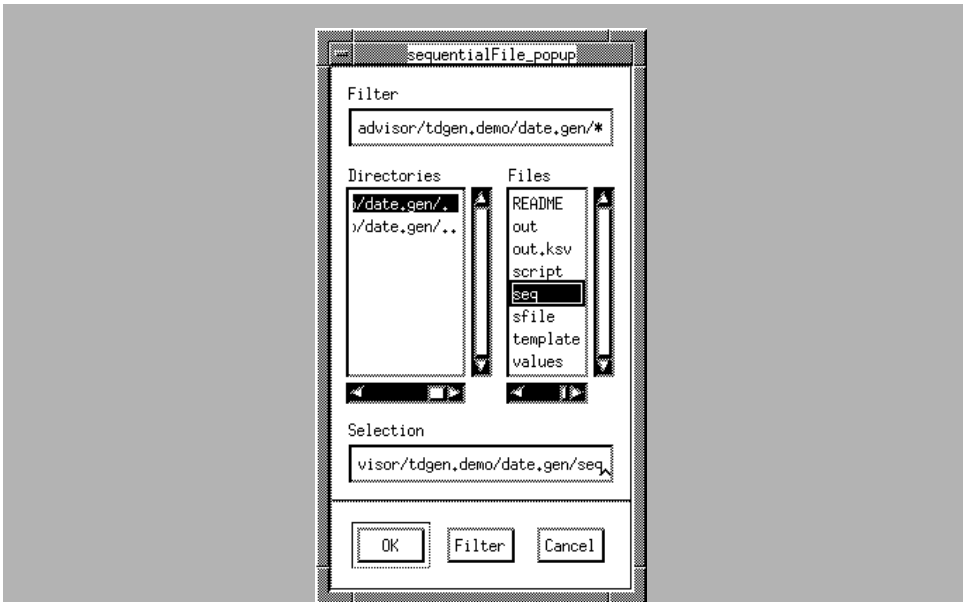

FIGURE 6 Set Sequential File Pop-Up Window

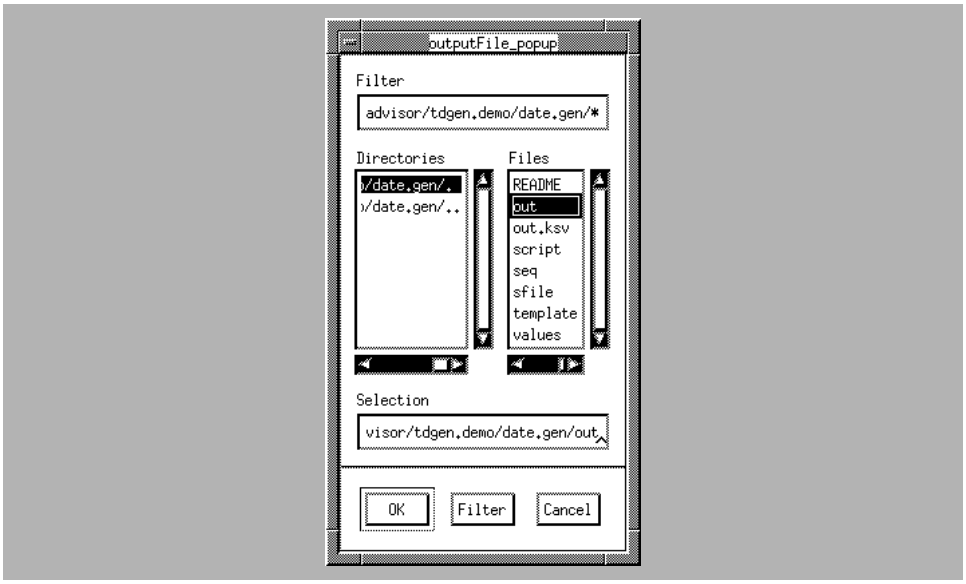

FIGURE 7 Set Output File Pop-Up Window

5. Select the invocation mode. Only one at a time may be selected.
6. Select the appropriate options:
   - Select “Display Output” if you want to display the output in the window.
   - “Invoke” can only be used for “Random Mode” and “Sequential Mode”. If it is not selected, then it assumes you want to generate only one set of tests.
   - “Specific Invocation Integers” can only be used for “Specific Mode”.
7. After selecting the mode and its corresponding options, click on the **Generate** pull-down menu and click on ‘**Generate Now**. The output should be displayed in the window.

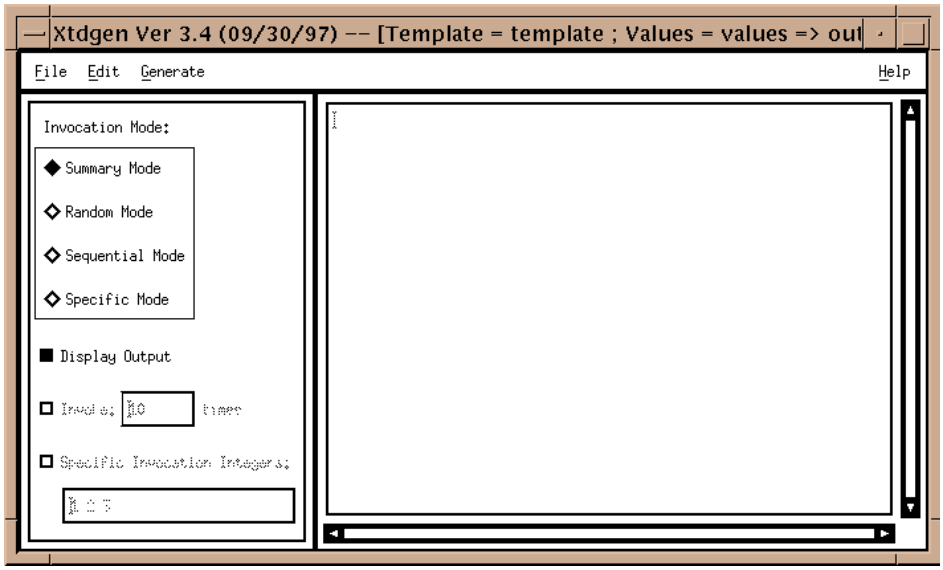

FIGURE 8 Summary Mode Output

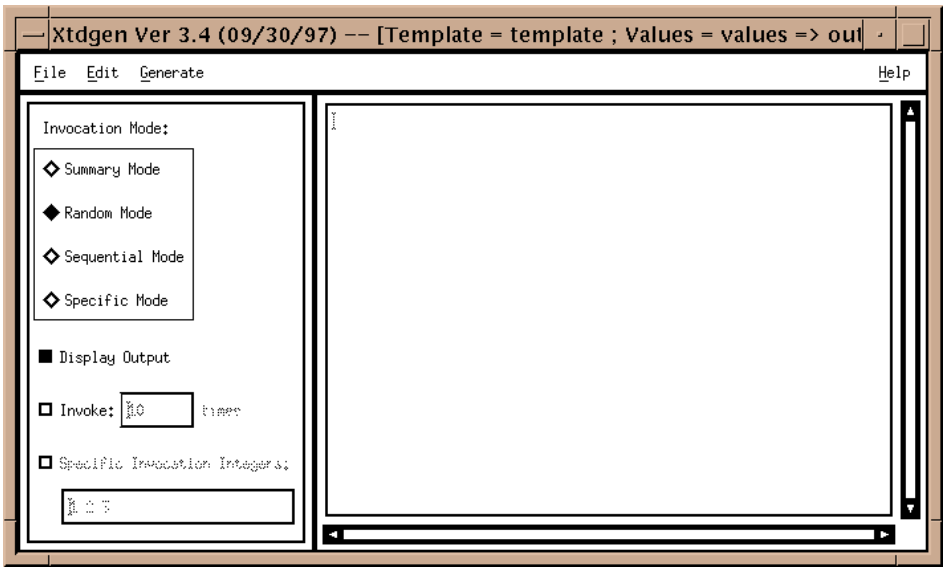

FIGURE 9 Random Mode Output

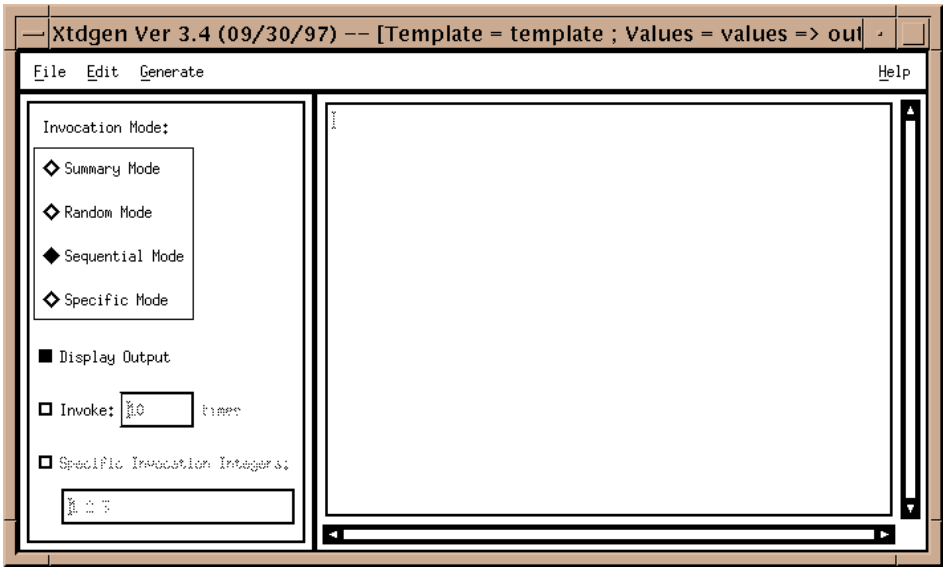

FIGURE 10 Sequential Mode Output

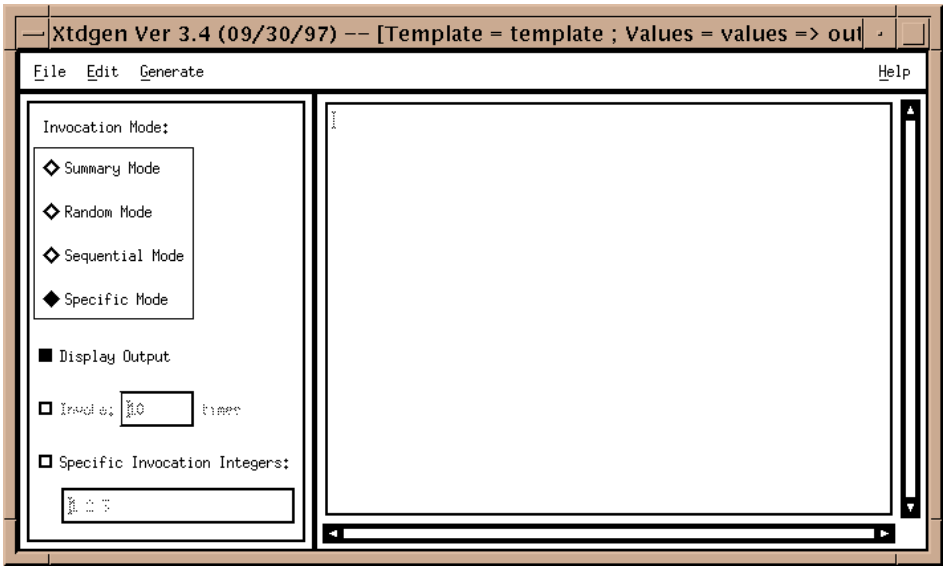

FIGURE 11 Specific Mode Output

8. If you want to edit the template file, values file or sequential file, then click on the **Edit** pull-down menu.
9. If you want to change the default vi editor to another editor, then click on **Set Editor Command**. The message window below pops up.

When finished changing the editor, click **OK**.

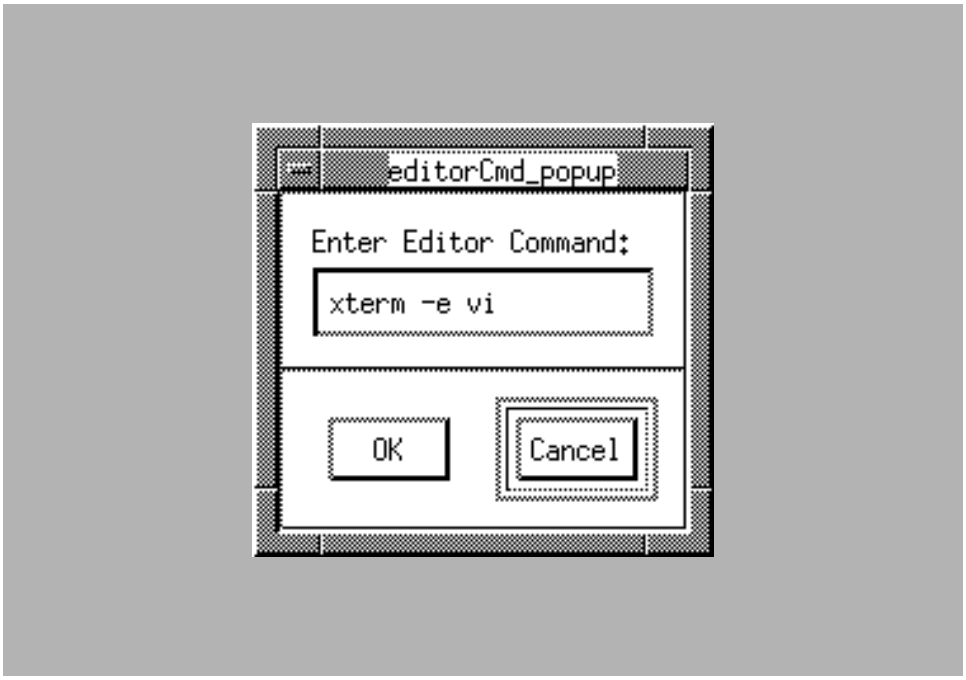

FIGURE 12 Set Editor Command Message Window

10. Click on the file you want to edit: **Edit Template File**, **Edit Values File**, or **Sequential File**.

Note: All files must be set before editing under the **File** pull-down menu.

11. A window such as the one below pops up. You can edit just like you normally would in **vi**.

12. In order to exit, type `:wq` (for save) or `:q` (for quit) or `w` (for write only). Do not use the window's menu button to close, as you may possibly exit the entire program.

13. When finished, click on the **File** pull-down menu's **Exit**.

At this point, you have successfully used ***TDGEN***.

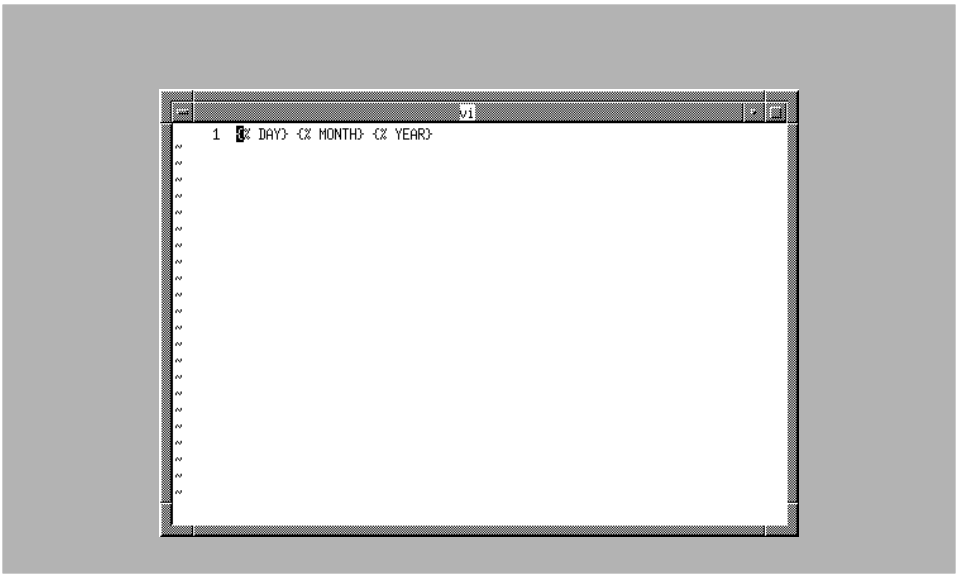

FIGURE 13 Edit Template File

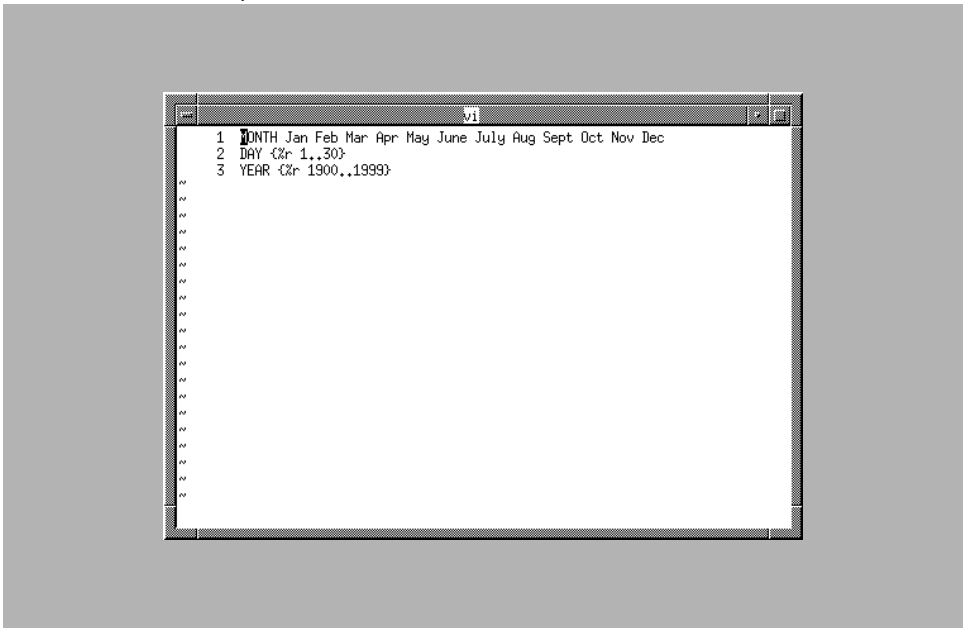

FIGURE 14 Exiting Out of Template File

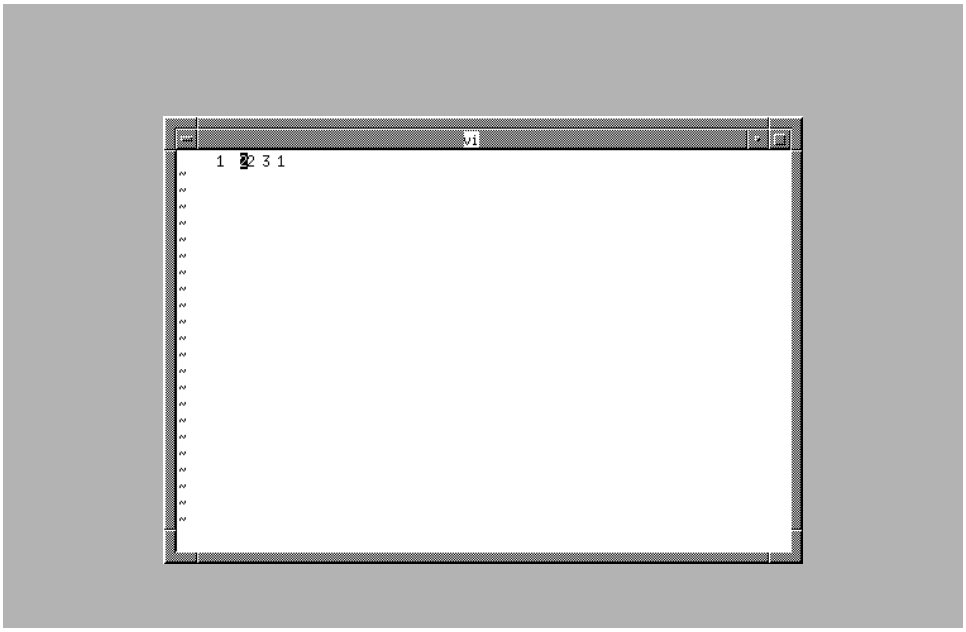

FIGURE 15 Edit Values File

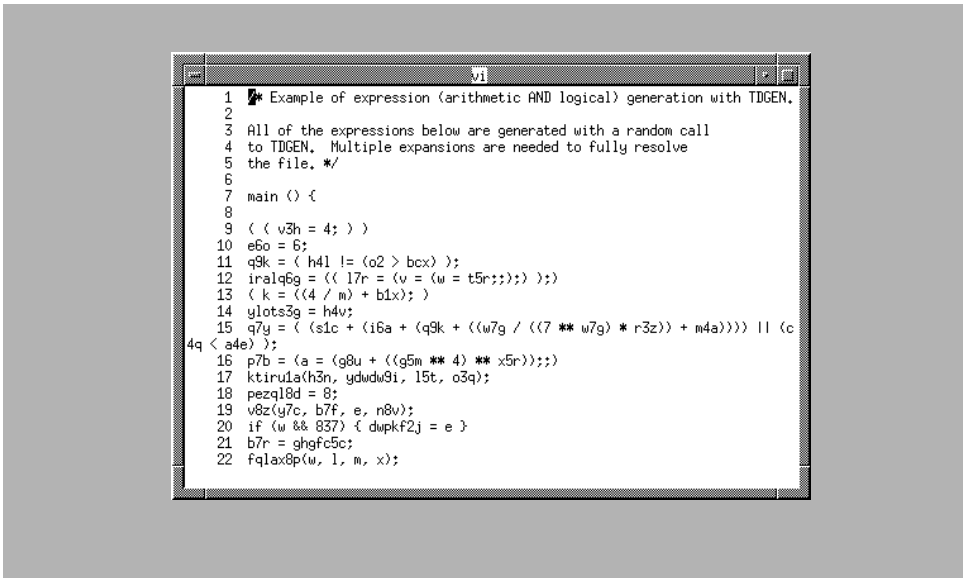

FIGURE 16 Edit Sequential File

CHAPTER 5

Template, Values File Syntax Summary

This chapter shows the proper syntax for both *TDGEN*'s template and values files.

5.1 Template File Syntax

NOTE: In the definitions below, square brackets, [ ], signify optional fields, and quotes, " ", signify literals.

Format	{%[<format spec.>] <field name> [<field size>]}
<format spec.>	[<field length>][<conversion>] <command> <comment>
<field length>	[-]d | [-]dd
<conversion>	"d" for decimal "o" for octal "x" for hexadecimal
<field name>	Any string of characters (no embedded white space) or in the case of <command>, any legitimate system call.
<command>	! <system call's output>
<comment>	"c" followed by comment text
<field size>	For pre-defined descriptors only. n for "ascii", "alpha" and "decimal" where (0 < d <

255).
n.m for "real" where ((n>=0) and (m > 0) and ((n+m) < 254).

5.2 Template File Definitions

-	If used, the - must precede field length. Signifies left justification within the specified field length. Absence defaults toright justification.
<field length>	The value specified will be the minimum field length used. If the string or integer exceeds this value in length, it will be written out in its entirety without truncation.
<conversion>	When one of these options is specified, the input value, i.e. the value read from the values file, must be a decimal integer. The value will be converted and output in the specified mode. For octal and hexadecimal conversions, the integer must be unsigned. The "d" option can be used to check if the input value is a proper decimal integer.
<field name>	Must always be present except when a system call takes its place.
"\ \ n"	Escapes carriage return. If a carriage return is preceded by a backslash, both characters will be ignored.

5.3 Values File Syntax

Format {<field name> <list of values>}

<field name> Any string of characters beginning with a non-white character that follows a new line and terminating with blanks or tab characters. Embedded white space is not allowed. String must be unique.

<list of values> <ranges of values><blank values><values>

List of values for the particular field name. Values can be any combination of the three types of values listed. Values must appear on one line. If characters do not fit on one line, use a backslash followed by a new line.

<ranges of values>{%r n..m}

Denotes range specification; n and m must be integers with n < m. Spaces after r are optional. If a particular value is specified by an integer i, then the value returned will be (n + i - 1); i must not exceed (m - n + 1).

<blank values> {%n}
value is n blanks.

<values> Any string of characters. Special characters must be escaped with a backslash if intended to be part of the values.

\ Escapes blank, i.e., allows blanks and tabs inside strings.

\ \ n Escapes carriage return, i.e., ignores both the backslash
and the carriage return.

\ \ Escape escape. For a backslash

character to appear in the output, it must be preceded by another backslash.

Comment {%c...}

Denotes comment between curly braces; carriage returns within comments must be preceded by a backslash.

CHAPTER 6

TDGEN Samples

Test data / file generators are:

- Useful in constructing test data automatically.
- A step in the right direction towards “fully tested software”.
- Useful in stress tests of compilers (example to follow).
- Versatile; they can generate data in a variety of formats.
- Useful additions to the software tool set.

The examples given next show several different kinds of *TDGEN* use.

Example: *TDGEN* fills in the “descriptors” in the template file, i. e., name, surname, etc., with that descriptor’s associated data from the data file. The numbers accompanying the fields specify the field width of the output.

```
<template File>

{%-10 name}{%-15 surname}{%5 age}
{%-10 name}{%-15 surname}{%5 age}

<values File>

name   Art David Saul Lamont John Jack\
Jill Laura Ruth Regina
surname    Nguyen Rodriguez Steinberg Col-
lins\
```

```
McGrath Banducci Dumaine
age   {%r 1..100}

<TDGEN Output>

David     Rodriguez         29
John      Dumaine           50
```

Example: By using a different data file in the above example, the output from *TDGEN* can be another template file of greater complexity. In this case, we will add another field to the original template - the middle name.

```
<template File>

{%-10 name}{%-15 surname}{%5 age}
{%-10 name}{%-15 surname}{%5 age}
<values File>

name{%-10\ name}{%5\ midname}.
surname Wong Nguyen Rodriguez Steinberg
Collins\
McGrath Banducci Dumaine
age{%r 1..100}

<TDGEN Output>

{%-10 name}{%5 midname}.Dumaine
61
{%-10 name}{%5 midname}.Wong
21
```

Example: Now if we take the output from the above example, and process it through *TDGEN* with a different data file, we will have a finalized version.

```
<TDGEN Output and template>

{%-10 name}{%5 midname}.  Dumaine   61
{%-10 name}{%5 midname}.  Wong      21
<values File>

nameArt David Saul Lamont John Jack\
Jill Laura Ruth Regina Janet
midname A B C D E F G H I J K L M N O P\
Q R S T U V W X Y Z
surname Wong Nguyen Rodriguez Steinberg
```

```
Collins\
McGrath Banducci Dumaine
age{%r 1..100}
<Final Output>

Ruth          N.Dumaine           61
John          B.Wong              21
```

As a more elaborate example, use *TDGEN* to select variables and expressions to generate a pseudo-random Fortran program. This would be the template file:

```
    {% symbol } = {% exprA}
    DO 50 {% symbol}={% symbol},{% sym-
bol},{% symbol}
50  CONTINUE
    IF {% exprL} THEN
    IF {% exprL} THEN        {% symbol} =
{% exprA}
    DO 51 {% symbol}={% symbol},{% sym-
bol},{% symbol}
    IF {% exprL} THEN        {% symbol} =
{% exprA}
51  CONTINUE
    ELSE
    {% symbol} = {% exprA}
    {% symbol} = {% exprA}
    ENDIF
    {% symbol} = {% exprA}
    {% symbol} = {% exprA}
    END
```

NOTE: This is a template file. The descriptors are: `symbol`, `exprA`, `exprL`. Descriptors are denoted by {%. *TDGEN* will replace the three descriptors with tokens, arithmetic expressions, and logical expressions respectively. Let's look at the file that contains the data:

```
VALUES FILE
-----------
exprA ({%\ exprA}{%\ op}{%\ exprA}) {%\ d}
\
 {%\ l}{%\ d}{%\ l}
symbol {%\ l}{%\ d}{%\ l}   {%\ l}
exprL   ({%\ l}{%\ d}{%\ l}.{%\ lop}.{%\
l}{%\ d}{%\ l})
```

```
op+ - * / **
l  I J K L M N
d {%r 1..9}
lop  NE EQ LE GE GT LT AND OR
```

NOTE: In this case we have chosen to expand the descriptors into strings that contain more descriptors. Arbitrarily complicated expressions will result after repeated calls to *TDGEN*. Notice that we have chosen the identifiers to consist of a digit sandwiched in between two letters. Also, all variables created will be implicit integers since the letters permitted range from I through N.

After the two files have been combined through *TDGEN*, the output looks like this:

```
OUTPUT
------
 {% l} = ({% exprA} {% op} {% exprA})
    DO 50 {% l}{% d}{% l}={% l}{% d}{%
l},{% l},{% l}{% d}{% l}
50  CONTINUE
    IF ({% l}{% d}{% l} .{% lop}. {% l}{%
d}{% l}) THEN
    IF ({% l}{% d}{% l} .{% lop}. {% l}{%
d}{% l})\
    THEN        {% l}{% d}{% l} = ({%
exprA} {% op} {% exprA})
    DO 51 {% l}{% d}{% l}={% l}{% d}{%
l},{% l}{% d}{% l},\
{% l}{% d}{% l}
    IF ({% l}{% d}{% l} .{% lop}. {% l}{%
d}{% l})\
    THEN        {% l}{% d}{% l} = {% d}
51  ELSE
    {% l} = {% d}
    {% l}{% d}{% l} = {% l}{% d}{% l}
    ENDIF
    {% l} = ({% exprA} {% op} {% exprA})
    {% l}{% d}{% l} = ({% exprA} {% op} {%
exprA})
    END
```

NOTE: The original template file has been expanded to include extra descriptors. The output itself is another template file.

Consistent with Fortran syntax for integers, `{% l}` can be replaced only by I through N, and *{% d}* can be replaced by digits 1 through 9. `exprA` has been replaced with `exprA op exprA`.

To obtain more complicated arithmetic expressions, the output from the first call to *TDGEN* is fed back to *TDGEN*. Here are the results:

```
    I = ({% d} / {% l}{% d}{% l})
    DO 50 J5N=M5L,L,K9I
50  CONTINUE
    IF (N9M .LE. K1L) THEN
    IF (M4N .EQ. L7I) THEN       K5I = ({%
l}{% d}{% l} / {% d})
```

```
    DO 51 J9N=J8M,N6L,M2K
    IF (M1J .GE. M3N) THEN        M1K = 1
51  CONTINUE
    ELSE
    L = 2
    I4M = J9M
    ENDIF
   K = (({% exprA} {% op} {% exprA}) * ({%
exprA} {% op} {% exprA}))
    K9J = (({% exprA} {% op} {% exprA}) **
{% d})
    END
```

NOTE: {% lop} will be replaced by Fortran logical operators. Notice that some descriptors have already been filled in. The file is processed a third time through *TDGEN*.

```
OUTPUT
------
    I = (3 / I2K)
    DO 50 J5N=M5L,L,K9I
50  CONTINUE
    IF (N9M .LE. K1L) THEN
    IF (M4N .EQ. L7I) THEN        K5I =
(M6I / 5)
    DO 51 J9N=J8M,N6L,M2K
    IF (M1J .GE. M3N) THEN        M1K = 1
51  CONTINUE
    ELSE
    L = 2
    I4M = J9M
    ENDIF
    K = (({% d} + ({% exprA} {% op} {%
exprA})) *\
    ({% d} / ({% exprA} {% op} {% exprA})))
    K9J = (({% d} / {% l}{% d}{% l}) ** 8)
    END
```

NOTE: Since there are still descriptors in this output, it will be processed by *TDGEN* with a different data file next.

After the expressions have been sufficiently expanded, *TDGEN* will now be invoked with a different data file to fill in the last descriptors. Here's the second data file:

```
VALUES FILE
-----------

exprA{%r 1..9}

op+ - * / **

l  I J K L M N

d {%r 1..9}

lop  NE EQ LE GE GT LT AND  OR
```

NOTE: This data file contains only terminals, i. e., descriptors from the template file will not be filled in with other descriptors. Therefore, after an invocation with this data file, the Fortran file will be essentially complete.

After *TDGEN* has been called with the second data file, the final output looks like this:

```
-----------------------------------------
----------
OUTPUT
------

 I = (3 / I2K)
 DO 50 J5N=M5L,L,K9I
50 CONTINUE
 IF (N9M .LE. K1L) THEN
 IF (M4N .EQ. L7I) THEN K5I = (M6I / 5)
 DO 51 J9N=J8M,N6L,M2K
 IF (M1J .GE. M3N) THEN M1K = 1
51 CONTINUE
 ELSE
 L = 2
 I4M = J9M
 ENDIF
```

```
 K = ((2 + (6 ** 4)) * (7 / (1 ** 9)))
 K9J = ((3 / K9M) ** 8)
 END

-----------------------------------------
----------
```

Index

Symbols

A

B

C

D

E

F

G

H

I

M

N

O

P

R

S

T

U

V

X