ENERGY LABORATORY

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

A DESCRIPTION OF THE THERMAL POWER SYSTEM ANALYSER STRUCTURE AND COMMANDS

by

B.I. Margulies and P.K. Sharma

MIT Energy Laboratory Report No. MIT-EL 80-030
September 1980

A DESCRIPTION OF THE THERMAL POWER SYSTEM ANALYSER STRUCTURE AND COMMANDS

by

B.I. Margulies

and

P.K. Sharma

September 1980

Energy Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Prepared for the
U.S. Department of Energy

MIT Energy Laboratory Report No. MIT-EL 80-030

N O T I C E

CONTENTS

ABSTRACT

This report describes a system comprised of a set of interactive commands and a data base which aids in the modeling of thermal power systems with the aid of a computer. This system, named TPSA (Thermal Power System Analyser), is implemented as a subsystem within the Multics interactive system, which runs on a Honeywell 6180 computer. TPSA deals with models and data bases. A TPSA model is a program capable of computing various output parameters of a thermal power system when given the values of necessary input variables. A TPSA data base is a collection of named variables which a model can access to obtain the value of a parameter or record a result. The interactive commands facilitate setting desired values of the input variables, executing the program, and displaying the selected output variables at the end of the program run. Some special commands, generally found useful in modeling work are also included. Procedure to modify the data base or to create a new data base is described. Some typical examples are included that provide a record of the interactive session with TPSA, illustrating model execution and use of the "probe" command for debugging the program.

CHAPTER I

INTRODUCTION TO MULTICS TPSA

TPSA (Thermal Power System Analyses) is designed as a subsystem of the Multics interactive system, which runs on a Honeywell 6180 computer. TPSA shares many of the conventions and facilities of the normal Multics environment while providing the special facilities used for modeling power systems. Briefly, TPSA consists of an implementation of a specific set of interactive commands and a database within the Mutlics interactive system.

TPSA deals with two kinds of objects: models and databases. A TPSA model is a program written in FORTRAN or PL/1 that is capable of computing the various outputs of a power system (FBC) when given the values of some set of input parameters. A TPSA database is a collection of named variables which a model can access to obtain a parameter or record a result. The TPSA commands aid in setting desired values of the input variables, executing the program, and displaying selected output variables at the end of the program run.

The TPSA user may modify his programs from time to time. The interactive commands to carry out such modifications, as well as the wider use of Multics is described in Multics Introductory User's Guide [1] and Multics Fortran User's Guide [2].

Relationship to Multics System

An outline of the TPSA structure within the Multics system is shown in Figure 1. TPSA allows any finite number of users to use the facility. When the user first logs in, he is in the normal Multics environment. A 'tpsa' request leads him to the TPSA level where he can use the specific

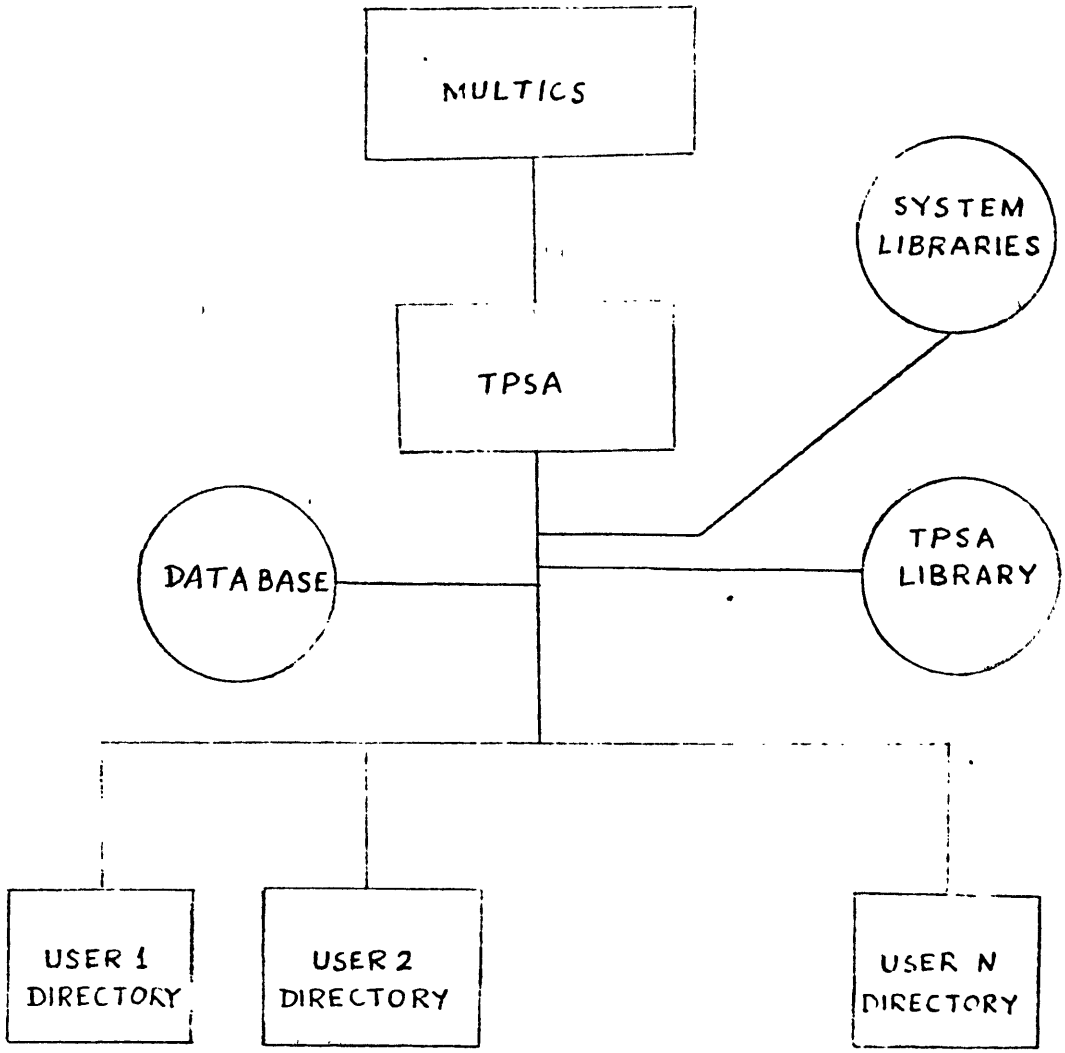

FIG. 1 TPSA STRUCTURE AND RELATIONSHIP TO MULTICS SYSTEM

TPSA commands and requests. At this point, he has access to segments (or programs) contained under his own user I.D., i.e., the segments in this own directory. In addition, the user is connected to the TPSA database and a TPSA library. The TPSA library contains a set of previously tested programs applicable to the particular thermal power system (FBC) being analyzed. The user is also connected to the computer system libraries which contain standard built-in functions and programs. However, to use a particular program from some of the special system libraries (e.g., imsl library), a linkage has to be made between the user's directory and the desired library program.

CHAPTER II

TPSA STRUCTURE AND COMMANDS

Models, variables, and datafiles are important components of TPSA structure. These are described first. This is followed by a description of various TPSA commands that are available.

2.1 Models, Variables, and Datafiles

TPSA Models

This information describes general features of tpsa models. tpsa models are programs that are run under the tpsa subsystem.

tpsa models get data from the active database by calling the tpsa_$read and tpsa_$write subroutines.

TPSA Variables

All information in a tpsa database is stored in variables. Each variable has a name, a long descriptive name, units, and one or more values. Optionally, a variable can be associated with a group. In general, variables are referenced by a specification of the form:

Name[Element]

Name is the variable's short name, and Element is an integer that specifies the array element that is desired. If you leave off Element then 1 is assumed.

Datafiles

tpsa datafiles consist of groups, which in turn consist of varialbes. Each variable may be an array of values.

There are two types of files. Each invocation of tpsa has associated with it an active file. All requests that deal with groups and variables affect only the active file. The active file is discarded when the invocation ends. The other type is permanent files. Permanent files can only be accessed by reading them into the active file with the read request. From then on you can only change the copy in the active file. To update the real file you must write it out again using the write request.

Databases are protected against inconsistency in several ways. If you try to write out a file when someone else is in the middle of doing the same thing, you will get a warning telling you who the other user was. You should not just retry the write request, destroying their changes if you succeed. Rather, you should send them an interactive message to try to avoid the conflict.

If the user fails to completely write out a file due to an error, reading that file will produce a warning message, and attempts to write to that file will fail. You can still delete it using the Multics delete command.

If someone else is writing the file while you are reading it tpsa will try several additional times and advise you if it fails. This means that someone else is repeatedly writing to the file as a write operation takes comparatively little time.

Permanent files have suffixes of "tpsa"; they will appear in Multics directory listings as <name>.tpsa where <name> is the entry name given to the read or write commands.

2.2 TPSA Commands

The following TPSA commands are available:

set
create_group
create_variable
display_group
offline
value
read
write
run
iterate
run_end_command
prec (print_run_end_command)
ump (use_my_program)
ulp (use_library_program)

SET

Syntax: set Var Value

Function: set sets the value of a tpsa variable

Arguments & Notes

Arguments:

Var is the variable to be set.

Value is the value to set it to.

Examples: set bolts 4.5

set tgas 1117.0

CREATE GROUP

create_group,cg

Syntax: cg Group_name LongName

Function: create_group creates new tpsa variable groups.

Arguments & Examples

Arguments:

GroupName

is the name of the group to be created. If may be 8 characters or less.

LongName

is a long descriptive name to be associated with the group.

Example: cg stones properties of the stones.

CREATE VARIABLE

create_variable, cv

Syntax: cv VarName Units LongName

Function: create_variable creates new tpsa variables

Argument & Notes

Arguments:

VarName

is a full variable specification, like Group.Var:size

Units

is the units of the variable, or "" if there are none.

LongName

is a long description of the variable.

Notes: In a variable specification Group is the name of the group, or * for the unnamed group, Var is the name of the variable, and size is the size of the array, size defaults to 1 if none is given.

Examples: cv *.v1 furlongs this is the length of the frog propeller
will create the variable v1 in the unnamed group.
cv colors.green "" this is the color of an unripe apple
will create the variable green in the group colors with no units.

DISPLAY GROUP

display_group,dg

Syntax: dg Group

Function: display_group prints the values and units of all the variables of a group on the terminal.

Arguments:

Group is the name of the group.

OFFLINE

Syntax: off

Function: prints the values of all variables on the offline printer.

Notes: offline creates segments with unique names in your directory, with names like:

!BBBjHzMBBBB.dprint.tpsa

They are normally deleted after they are printed, but some errors may cause them to hand around. To look for them, use the Multics list command:

list *.dprint.tpsa

from Multics command level,

e ls *.dprint.tpsa

from tpsa command level.

VALUE

value, v

Syntax: v Var1 Var2 Var3 .. Varn

Function: prints the values of one or more variables on the terminal.

Arguments:

Varn

is a tpsa variable name.

control arguments:

-brief, -bf

suppresses printing of group name and just prints the numeric value.

READ

Syntax: read Path

Function: reads a saved file into the active file.

Arguments & Notes

Arguments:

Path

is the pathname of a saved tpsa database. The suffix .tpsa is added by default.

Notes: If there is already data in the active file, the user is queried before the existing data is overwritten with the new data.

WRITE

write

Syntax: write Path

Function: saves tpsa databases.

Arguments:

Path

is the pathname of the file where the data is to be saved.

RUN

run

Syntax: run Model

Function: runs a tpsa model

Arguments & Notes & Examples

Arguments:

Model

is the model to run. It may be the pathname of a compiled object segment or an entryname. If it is an entryname, the search rules are used to find the model to run.

Notes: The model is found in the same way Multics commands are found. Users on the TPSA project have the directory >udd>tpsa>lib> searched in their search rules, so they can find the installed model. It is currently called main.

Examples: run main

will run the normal model.

ITERATE

iterate

Syntax: i Model Var1 Sval1 Eval1 Step1 Var2 Sva12 Eva12 Step2...

Function: runs a model repetitively while incrementing the values of one or more variables.

Arguments:

Model

is the name of the model. See the information on run.

Varn

is the name of a variable to iterate.

Svaln

is the starting value for the iteration. If it is just ".". then the current value is used.

Evaln

is the ending value for the iteration.

Stepn

is the increment to be added each iteration.

Notes & Examples

Notes: There can be any number of iteration sets specified. The variable is restored to its original value after the run completes.

Examples: i hardware nuts 1 5 2 bolts 3.4 20.0 .2 screws 10 5 -1 jars . 6 1

RUN END COMMAND

run_end_command, rec

Syntax: rec Comline

Function: sets a tpsa command line to be executed after each run of a model.

Arguments:

Comline

is a tpsa command line. It should be enclosed in quotes.

Notes: Giving a null ("") command line removes an existing one.

Examples: rec "value springs"

PREC

print_run_end_command, prec

Syntax: prec

Function: prints the run end command if one is set.

USE MY PROGRAM

use_my_program, ump

Syntax: ump ProgName

Function: Compels the system to execute the segment 'ProgName' in user's directory even though a different segment named 'ProgName' may exist in TPSA library or system libraries.

Arguments & Notes

Arguments:

ProgName

is the name of a compiled program, to be run by itself or as part of a larger program.

Notes: This command is helpful when comparing different versions of a program; one version exists in the library, another in user's own directory.

USE LIBRARY PROGRAM

use_library_program, ulp

Syntax: ulp ProgName

Function: Compels the system to execute the segment 'ProgName' in TPSA library even though a different segment named 'ProgName' may exist in user's own directory.

Arguments & Notes

Arguments:

ProgName

is the name of a compiled program, to be run by itself or as part of a larger program.

Notes: This command is helpful when comparing different versions of a program; one version exists in the TPSA library, another elsewhere.

CHAPTER III

COMMON USAGE OF TPSA COMMANDS AND REQUESTS

To access TPSA it is first necessary to access the Multics system. Information on accessing the system is given in Chapter V (Example 1). (For additional details the user may read the Honeywell "Multics Introductory Users' Guide" (Reference 1) or the "Notes on Using Multics" available from the MIT Student Information Processing Board.) When the users first log into Multics, they get a Multics ready message, which Multics types to inform the users that it is listening. The ready message looks like:

```
r 11:18 3.779 297 $1.62
```

The first number is the time of day, the second the cpu time, the third is the memory usage, and the last number is the cost of the current session. To access the TPSA subsystem, the user must give the "tpsa" command by typing:

```
tpsa
```

TPSA responds by printing a message like:

```
Multics tpsa version 3.1a
type help for help, ? for a list of requests.
tpsa:
```

Typing 'help' at this point will prompt the system to print out information on selected TPSA components, whereas typing '?' will generate a list of TPSA requests. You may skip these if you are testing model execution alone. To run the model, one first needs the "read" request to obtain input variables from the data base. TPSA database is stored in Multics files; to use it you must read it into your "active file". This is accomplished by typing:

```
read system
```

in response to the "tpsa" prompt. If you have been working with some other database you will be queried as to whether you want to discard the stuff in the active file to replace it with the database you are reading in.

Once you have read a database in, you can examine and change the values of the variables. These changes have no effect on the permanent copy that was read in; it only changes the "active file".

To look at the value of a variable use the "value" request, which has the short name "v". For example, to see the value of a variable named "beda" (bed area) you would type:

```
v beda
```

TPSA will respond by typing out its value. To make it easier to organize, data variables have been organized into groups as described earlier. The "display group" command (short name "dg") will print on your terminal the values of all the variables in a group. To see the values of all of the variables in the group "bedprop" you would type:

```
dg bedprop
```

Finally, it is possible to get an offline printout of all the variables in the database by given the "offline" request.

The next step is to set the values of some variables to see what effect it has on the model. The "set" request provides this feature:

```
set beda 3.14158
```

would set the value of beda to pi.

Once you have set the variables to your satisfaction, you are ready to run the model. The normal model is called "main". To run it type:

```
run main
```

The model will obtain its parameters from the database and leave its output in the output variables where they can be examined with the value request.

There are other requests that can be used to run the model repeatedly while iterating over the value of variables or automatically print the values of some selected variables after a run.

Once the database has been modified, the user might want to save modified values to avoid typing them all over again. The "write" request is provided for that purpose. Typing:

```
write mydata
```

would create a TPSA permanent file in the user's directory called "my data". The next time around the user would use it by giving a command like:

```
read mydata
```

instead of "read system".

The next thing the user might like to do is to modify an existing model. Before you can try to write a new piece of a model you must know how to write a Multics Fortran program. Multics Fortran differs slightly from regular Fortran. The Fortran User's Guide (Reference 2) is recommended reading. There are two specific issues that are important. The first is how to access the database. There are two subroutines that are provided. tpsa_$read gets the values from the database, and tpsa_$write puts values back in. To use them you must first declare them to the fortran compiler with a statement like:

```
external tpsa_$read (descriptors)
```

Then to read a variable called "beda" the call statement would be:

```
call tpsa_$read ("beda", beda)
```

and beda would be set to the value of the variable beda. tpsa_$write is called the same way. These subroutines cannot be called from outside the TPSA environment. If one of them is called from outside of TPSA, an error message is printed.

The second consideration for the model-modifier is the library. On Multics, the various subroutines of a fortran program do not have to be prelinked together. Instead, the system maintains a list of places to look for executable programs. When a fortran program executes:

```
call calculate_so2 (density, temperature)
```

the system looks first in the user's working directory, then in the TPSA library, and then in the system libraries. The same thing happens when you type a Multics command, which is how the system finds the TPSA subsystem even though it is not an installed, official program. However, note that before Multics looks in your directory, it checks to see if you have called that program before in the current process. If so, it will terminate further search. Thus, if you run the installed model, and then make a private version of something in it, and try to run again, you will still get the library copy.

To allow the user to be certain of what they are getting, tpsa requests "use_my_program" (short name ump) and "use_library_program" (short name ulp) are useful. The "use_my_program" request takes as an argument the pathname of the program to use. Thus typing:

```
ump calc_nox
```

will force TPSA to use the version of the "calc_nox" subroutine in the current working directory instead of the library.

```
ump >udd>TPSA>DLee>calc_nox
```

will use DLee's version. Finally:

```
ulp calc_nox
```

will revert to using the library version.

One final note: the source to the current model can be found in the directory >udd>TPSA>lib>s. Thus to make a copy of the routine "fluidm" to

modify the user would type the Multics command:

copy >udd>TPSA>lib>s>fluidm.fortran

or copy <lib>s>fluidm.fortran

If the user has made significant changes to the model, he may well need to add or delete variables from the database. To add a varaibale to an existing group use the "create variable" (short name cv) request. A create variable request looks like: cv bedprop.temp "kelvin" this is temperature of particle surface. This creates a new variable in group bedprop whose name is temp and whose units and long descriptive names are self-evident. To create a variable with no units type "" for them.

To create a new group use the "create_group" (short name cg) request, like:

cg mprops Some long descriptive name

CHAPTER IV

SAMPLE USER SESSIONS

4.1 Model Execution

This example illustrates a typical user session with TPSA where the user logs in, picks the input variables, executes the model and looks at output variables, and finally logs out.

When the user first connects to Multics, the following 'greetings' message is printed on the terminal giving date and time of the day and the current load:

Multics 34.26: MIT, Cambridge, Mass.

Load = 39.0 out ot 95.0 units: users = 39, 09/17/80 1028.8 edt Wed

The user then types in the login request:

login PSharma

Multics responds by asking for the password and blanking out the space below.

Password:

* * * *

When the password is supplied, rest of the login procedure is completed as shown below:

Your project's account has less than $580.79 (1%) of its funds remaining.

You are protected from preemption.

PSharma TPSA logged in 09/17/80 1029.5 edt Wed from ASCII terminal "none".

Last login 09/16/80 1637.1 edt Tue from ASCII terminal "none".

funds used: $1683.36

You have no mail.

r 10:29 3.631 175 $1.03

The user is now at the Multics level. To reach the tpsa level, the types

```
tpsa
```

to which the system responds by printing

```
tpsa
Multics tpsa version 3.1a
Type help for help, ? for a list of requests.
tpsa:
```

The user then reads the database onto an active file by issuing the command

```
read system
```

The system, thereupon, responds

```
Database >udd>TPSA>PSharma>system.tpsa
                created 09/02/80 2035.7 edt Tue by PSharma. TPSA.
tpsa:
```

The user is now ready to execute the model if he is not interested in altering the default values of the input variables. To run the model; he types

```
run main
```

The above command causes the system to run the main program, which in turn, calls member routines, generating the following response:

```
CALL INPUT
OUTER LOOP
CALL PRECAL
MIDDLE LOOP
CALL FLUIDM
DAVIDSONS CORRELATION FOR BUBBLE GROWTH IS BEING USED
```

```
hjet = 0.359084E-01

h1 = 0.501999E-01

ht = 0.35904E-01

dbudo = 0.451363E-01

          CALL MASSEX

          INNER LOOP

          CALL COMB

ICCC = 0

ICC = 1

ICC = 2

     fines by attrition = 0.0000E+00

          CALL DESULF

          CALL MATBAL

          CALL CO

          CALL ENERGY

          CALL NOX

          CALL OUT
```

If the user wants the particle size distribution of carbon in the bed and associated steams printed at the terminal Type 1, and push return --otherwise type 0, and push return

The user may respond by typing 1 if he is interested in looking at the carbon particle size distribution for various streams. Doing so will result in a table being printed which is omitted here.

The user may now like to look at selected output variables. If he is interested in combustion efficiency, carbon load in bed, and bed height, he types the following command in response to the last tpsa prompt:

```
v comef wcoal bedh
```

The system responds by printing

```
comef   combn      9.3195321411e-01
wcoal   combn      2.7079270554e 02 kg
bedh    fluidpr    1.2317059785e 00 meters
tpsa:
```

Let us now assume that the user now wants to input a finite value of the attrition constant 'atrit', and also wants to reset coal feed size and superficial velocity. He issues the neccessary set requests one by one, as shown below:

```
                set atrit 3.0e-08
        tpsa:   set ros2 8.0e-04
        tpsa:   set uo 1.80
        tpsa:
```

The 'run main' command now will generate the following response:

```
        CALL INPUT
        OUTER LOOP
        CALL PRECAL
        MIDDLE LOOP
        CALL FLUIDM
DAVIDSONS CORRELATION FOR BUBBLE GROWTH IS BEING USED
hjet = 0.343481E-01
```

h1 = 0.505421E-01

ht = 0.343481E-01

dbubo = 0.432736E-01

CALL MASSEX

INNER LOOP

CALL COMB

ICCC = 0

ICC = 1

ICC = 2

ICC = 3

fines by attrition = 0.1017E-01

CALL DESULF

CALL MATBAL

CALL CO

CALL ENERGY

CALL NOX

CALL OUT

If the user wants the particle size distribution of carbon in the bed and associated steams printed at the terminal

printed at the terminal

type 1, and push return

--otherwise type 0, and push return

Assuming that the user is not interested in carbon size distribution this time, he types in 0, to which system responds by printing tpsa prompt.

The user now rechecks combustion efficiency, carbon loading, and bed height. He issues the request

v comef wcoal bedh

to which the system response is

```
comef   combn     8.9230854064e-01
wcoal   combn     1.0115819931e 02 kg
bedh    fluidpr   1.1821154356e 00 meters
```

Note that these values have changed compared to earlier run.

To leave the tpsa level, the user issues quit (q) request. System responds:

```
tpsa:  Current contents of active file will be lost.  Are you sure you want
to quit?
```

Answering 'yes' to above query will put the user back into Multics level:

```
tpsa (quit):  Good Morning, PSharma.
r 10:43 8.919 465 $3.14
```

The user may now terminate the session by typing

```
logout
```

to which the system responds:

```
PSharma TPSA logged out 09/17/80 1045.0 edt Wed
CPU usage 17 sec, memory usage 61.0 units, cost $3.20.
hangup
```

4.2 Group and Variable Display

This example illustrates the execution of 'display_group' (dg) and 'show_variable' (sv) commands from the tpsa level.

```
tpsa:  dg acptpr
Group acptpr: 6 variables
```

Acceptor Properties

casrt	2.0000000000e 00	Ca/S Molar Feed Ratio
dpart	7.9999999434e-04 meter	Acceptor Mean Particle Size
rhoai	2.2500000000e 03 kg/m**3	Acceptor Feed Density
sacc	8.3529999247e-03	Molar Ration of Mg/Ca in Acc. Feed
size	1.0000000000e 00	Acceptor Size Type - Borgwardt
ston	2.0000000000e 00	Acceptor Stone Type - Borgwardt

tpsa: sv coalf

coalf	combn	0. kg/sec	Coal Feed Rate

tpsa: sv coe wcoal xso2

coe	combn	0. kgmole/m**3	Effective Oxygen Concentration
wcoal	combn	0. kg	Carbon Loading
xso2	miscin	0.ppm	PPM of SO2 in Effluent Gas

tpsa:

4.3 Debugging the Program

This example illustrates the use of the 'probe' command to aid in debugging of a program when it encounters a fatal error.

For a certain set of input condition, the program fails and prints the following message:

```
Error:  Attempt to divide by zero at >udd>TPSA>DStern>freebd:2706 (line
283) system handler for error returns to command level
r 15:38 8.196 2145 level 2,21 $5.58
```

The above message tells that the zero-division error occurred in line 283 of the member named freebd. One may invoke the editor qedx and print this line.

```
* qedx
* r freebd.fortran
* 283p
  497   pco=ak2/ak1+(pco-ak2/ak1)*earg
* q
  r  15:39  0.278 249 level 2,21 $5.77
```

Looking at line 283, immediate suspicion is on the variable ak1. Now, invoke the 'probe' (pb) command and ask for the values of the various variables that appear in line 283.

```
*   pb
    Condition zerodivide raised at line 283 of freebd (level 14).
*   v ak1
    ak1  = 0
*   v ak2
    ak2  = 2.11134317e-3
*   v earg
    earg = 1.000000001

*   q
    r  15:41  0.671  752 level 2,21  $6.30

*   r1
    r  15:41  0.378  244 $6.51
```

The final 'release' (rl) command causes the various values generated during program execution to be erased from the memory and the system returns to original Multics command level.

The value of ak1 obtained above confirmed our initial guess.

NOTE: The lines with preceding '*' are typed by the user. The line that follows is system response.

REFERENCES

1. Multics Introductory Users' Guide (Level 68), Honeywell Information Systems, Waltham, Mass. 02154, Order No. AL 40, REV. 1.

2. Multics Fortran Users' Guide (Level 68), Honeywell Information Systems, Waltham, Mass 02154, Order No. cc70, REV. 0.