# AN INTERACTIVE DEBUGGING FACILITY FOR GPSS

James O. Henriksen

## ABSTRACT

This paper describes the interactive debugging feature of GPSS/H (1), a new implementation of GPSS for IBM 360/370 computers. In GPSS/H, interactive debugging of a simulation model is carried out by means of a simple, but powerful command language. Commands are provided to selectively display model data on a terminal, to set breakpoints at arbitrary points in a model, and to step through a model one or more blocks at a time. An overview of the command language and an illustrative example of its use are presented in this paper.

## THE NEED FOR INTERACTIVE DEBUGGING FACILITIES

The debugging of GPSS models has traditionally been a batch-oriented activity. In a batch environment, when an error is discovered in the execution of a model, the courses of action available to the programmer are limited. If the programmer is lucky, the cause of the problem may be obvious enough to be inferred directly from "standard" program output. If the problem cannot be located by examining standard output and proofreading the program, additional output must be obtained. The first type of additional output requested is often "snap" output, providing snapshots of the state of the model at a more frequent interval than standard output. Generally easily incorporated into a model, snap output may help to narrow down the time at which the error occurs and (possibly) the location within the program. If snap output is insufficient to solve the problem, more detailed output may be required, to provide an audit trail of actions taken by the model. This type of output can be very difficult to properly target. If the scope of the trace output is too large, insufficient information will be produced to locate the problem, necessitating preparation of another run. If the scope of the trace output is too small, much time will be spent examining the output to find the critical pieces of evidence. Finally, the lowest level of debugging technique available to the batch-mode programmer is modification of the model to trap highly specific conditions and print appropriate information. A well-designed model, if it is to be debugged in a batch environment, should be programmed to contain built-in debugging aids, to facilitate the production of trace output and to minimize the number of changes that must be made purely for debugging purposes.

There are two major difficulties associated with batch-mode debugging as outlined in the preceding paragraph. First, the selection of what output to print must be completely specified prior to making a run. Second, no matter how generalized the built-in debugging aids are in a model, errors can occur which necessitate modification of the model to isolate the problem. When the problem is found and fixed, the changes must be removed. Interactive debugging can do a great deal to alleviate these two difficulties. First, because the programmer can dynamically decide what output to display as the program is being run, targetting output is much easier. As model execution approaches the time and place of an error, the programmer can request increasingly detailed output. If necessary, the model can be executed one block at a time, with the programmer observing the effects of the execution of every block. Second, when debugging is done by means of an interactive command language, exogenous to the model, the number of changes that must be made purely for debugging purposes is minimized. The DISPLAY command, for example, may obviate the need for insertion of PRINT blocks.

To be perfectly fair, it should be pointed out that there are some advantages to batch-mode debugging. The ability to produce bulk output and the absence of connect-time charges come readily to mind. The best approach to debugging simulation models, then, may be a mixture of batch and interactive techniques. In an ideal work environment, the programmer should have both techniques at his disposal, choosing between them as he sees fit.

## THE HISTORY OF GPSS INTERACTIVE DEBUGGING

Prior to the implementation of GPSS/H, interactive debugging aids for GPSS models were relatively limited. Probably the most advanced of the pre-GPSS/H packages is GPSS/Norden (2). GPSS/Norden provides excellent tools for selective display of model data, aids for maintaining external libraries of matrix SAVEVALUE data, HELP blocks for communicating with terminals, and a SIMSCRIPT-like (3) report generator. However, GPSS/Norden has no provision for setting breakpoints in a model, nor does it allow execution of one or more blocks at a time (step mode).

The interactive debugging feature of GPSS/H was originally implemented as a tool for verifying the correctness of the internal operation of the compiler and simulator. The original debugging feature was a small (about 600-700 lines of PL/I) appendage to the logic of the simulator. Considerable effort was put into its design and implementation, to insure that it provided an accurate and independent view of actions taken by the simulator. After initial use proved to be very helpful, examples of use of the package were shown to users of GPSS. At their encouragement, the package was "cleaned up" and documented, making it usable by GPSS programmers not necessarily familiar with the internal implementation of GPSS/H. As presently constituted, the package provides the user with a collection of simple, but very powerful commands for interactive debugging, easily mastered in about one hour.

## INITIATING INTERACTIVE DEBUGGING OF A MODEL

Interactive debugging of a GPSS model is requested by use of the TEST keyword on the system command which invokes GPSS/H. The exact format depends, of course, on the operating system being used. For example, under the University of Michigan Terminal System (4) the following command might be used:

```
$RUN UNSP:GPSSH  SCARDS=TESTPROG  PAR=TEST
```

When such a request is made, compilation and loading of the model proceed in the usual fashion; however, at the point at which execution would normally begin, control passes to a command interpreter, which prints a "READY" message on the user's terminal. At this point, the user assumes (and hopefully retains) control of model execution.

## AN OVERVIEW OF THE GPSS/H INTERACTIVE DEBUGGING COMMAND LANGUAGE

### COMMAND SYNTAX

The command scanner is relatively tolerant of minor errors. For example, extra blanks and missing ")" delimiters are generally accepted without complaint. Commands may be abbreviated, with ambiguities resolved in favor of the more frequently used command. Thus "STEP" can be abbreviated "STE", "ST", or "S", while the less frequently used "STOP" command can be abbreviated only as "STO". Command lines which begin with a dollar sign ("$") are passed to the command interpreter of the host operating system (only in systems which allow dynamic command interpretation). For example, the Michigan Terminal System (MTS) user could enter the following command to request that MTS print the estimated cost of the current terminal session:

```
$DISPLAY COST
```

Command lines which begin with an asterisk ("*") are treated as comments, i.e., are ignored.

In the descriptions which follow, items which appear in upper case must be typed exactly as shown, with the exception of commands, which may be abbreviated as described above. Items in lower case represent symbols to be substituted by the user. Ellipsis ("...") is used to denote optional repetition of the proceeding item.

### DISPLAYING PROGRAM DATA

Three commands are available for selective display of program data. The DISPLAY and PRINT commands display data by invoking the output module of GPSS/H, in much the same manner as the PRINT block. The DX command displays data in hexadecimal form and may be useful in tracking down certain nasty bugs. The DISPLAY and PRINT commands differ only in the destination of their output: DISPLAY output is always printed on the terminal, while PRINT output goes to the file/printer used for all "normal" program output. The syntax of the DISPLAY, PRINT, and DX commands is as follows:

```
DISPLAY  entity ...
         info

PRINT    entity ...
         info

DX       entity ...
         regs
         entity2
```

The allowable values for "entity", "entity2", "info", and "regs" are as follows:

| "entity" | Interpretation |
|---|---|
| BLO | Blocks |
| FAC | Facilities |
| STO | Storages |
| QUE | Queues |
| CHA | User Chain Summary Statistics |
| UCH | User Chain Dump (all xacts on the chain) |
| GRP | Groups |
| LOG | Logic Switches |
| TAB | Tables |
| FSV | Fullword Savevalues |
| HSV | Halfword Savevalues |
| BSV | Byte Savevalues |
| LSV | Float Savevalues |
| FMS | Fullword Msavevalues |
| HMS | Halfword Matrix Savevalues |
| BMS | Byte Msavevalues |
| LMS | Float Msavevalues |

| "entity2" | Interpretation |
|---|---|
| FUN | Functions |
| VAR | Variables |
| BVR | Bvariables |
| RNO | Random Number Streams |

| "info" | Interpretation |
|---|---|
| OUTPUT | Standard Output (everything) |
| STATUS | Current transaction, clock values, termination counter |
| CLOCKS | Absolute and Relative Clocks |
| CEC | Current Events Chain |
| FEC | Future Events Chain |
| INT | Interrupt Chains |
| MAT | Matching Chains |

| "regs" | Interpretation |
|---|---|
| GRS | General Registers of the computer |
| FRS | Floating Point Registers of the computer |

The use of the mnemonics shown above allows printing entire classes of output. Output can be requested for a single entity or a range of entities by using subscript notation. Here are some examples:

| | |
|---|---|
| DISPLAY FAC STO | (Displays all Facilities and Storages) |
| PRINT QUE(SAM) | (Prints Queue statistics for SAM) |
| DX GRS FRS | (Displays the General and Floating Point Registers) |
| P TAB(1...7) | (Prints Tables 1 thru 7) |

Finally, note that any output can be terminated by pressing the attention interrupt button on the terminal.

SETTING BREAKPOINTS

Local or global breakpoints can be established at any block in a program, with no limit on the number in effect at any time. Global breakpoints remain in effect until they are explicitly removed. Local breakpoints remain in effect only for the duration of a single command. Global breakpoints can be established on the BREAK or RUN commands. Local breakpoints can currently only be specified on the CONTINUE command.

When a transaction attempts to enter a block at which a breakpoint is set, a message is printed, identifying the transaction and block, and control returns to the command interpreter. Note that if a breakpoint is established at a GENERATE block, it will be recognized twice per transaction: when the transaction "arrives" at the GENERATE block, and when the successor arrival (if any) is scheduled.

Associated with each global breakpoint is an "ignore" count, which defaults to "infinity" and may be set by the IGNORE command, allowing the breakpoint to be ignored a specified number of times.

In addition to block-oriented breakpoints, two special pseudo-breakpoints are provided, in order to give the user greater control. The "NEXT" breakpoint is recognized by the simulator when it "picks up" an active transaction in its scan of the current events chain. The "SYSTEM" breakpoint is recognized whenever the transaction currently moving through the program relinquishes control (e.g., is denied entry into a block). The SYSTEM breakpoint is particularly useful as a "fence," when a transaction is being tracked through a program in highly detailed fashion. If the transaction being tracked unexpectedly loses control of the simulation (i.e., is "dropped"), control is retained by the programmer, because (s)he is informed how and why the transaction was dropped, if the SYSTEM breakpoint is set. The NEXT breakpoint can provide great insight into the current events chain scan by identifying transactions as they are "picked up" during CEC scan. Both the SYSTEM and the NEXT breakpoint may be used in local and global fashion, but do not have associated "ignore" counts.

The RUN and CONTINUE commands, which can set breakpoints, are described below. The syntax of the BREAK, UNBREAK, and IGNORE commands is as follows:

```
BREAK     block-number ...
          NEXT
          SYSTEM

UNBREAK   block-number ...
          NEXT
          SYSTEM

IGNORE    block-number  optional-count
```

OVERALL RUN CONTROL

A program can run either in normal mode or in step mode. In normal mode, execution proceeds at "full speed," stopping only when breakpoints are encountered, errors are discovered, etc. In step mode control returns to the command interpreter after a specified number of blocks have been executed.

Normal execution is initiated by the RUN command. In addition to initiating execution, the RUN command can also establish one or more global breakpoints. Normally, one or more global breakpoints are established with the RUN or BREAK commands, in order to provide for interaction. Normal execution of a program is resumed by use of the CONTINUE command. The CONTINUE command accepts an optional list of local breakpoints, which remain in effect only for the duration of the command. This provides a convenient way to continue execution to one of a number of potential blocks.

Step mode is entered by use of the STEP command, which accepts an optional count of the number of blocks to be executed. If no count is given, a count of one is assumed. Note that the count is interpreted in terms of total block executions, not in terms of attempted block executions. The step count is decremented each time the total count for some block in the program is incremented. Control returns to the command interpreter when the count goes to zero; however, occurrence of some other event may cause control to return to command mode before the count has gone to zero. For example, a breakpoint may be encountered. Whenever such a premature return is made, the remaining step count is printed as a warning. Note that step mode cannot be resumed with a CONTINUE command, which is only for resumption of normal mode. Instead, a new STEP command must be issued, specifying the desired count.

The syntax of the RUN, CONTINUE, and STEP commands is as follows:

```
RUN        block-number ...
           NEXT
           SYSTEM

CONTINUE   block-number ...
           NEXT
           SYSTEM

STEP       optional-count
```

The following sequence of commands establishes global breakpoints at blocks 36 and SAM, runs the program until execution (presumably) reaches one of these blocks, executes five more blocks (unless a breakpoint is encountered), clears the breakpoint at block 36, and resumes execution until block 42 or SAM is reached:

```
BREAK 36
RUN  SAM
ST 5
UNBR 36
C  42
```

When debugging a program in step mode, it is frequently very useful to establish a SYSTEM breakpoint, to determine how and why transactions are "dropped" by the simulator. It is also convenient to "skip ahead" to the next transaction to be moved through the program. The following sequence of commands turns off the global SYSTEM breakpoint, skips ahead to the next "pick up" of a CEC transaction, and reestablishes the SYSTEM breakpoint.

```
UNB   SYSTEM
C  NEXT
BR    SYSTEM
```

The following sequence accomplishes the same thing, but causes a message to be printed when the current transaction is "dropped," assuming that the SYSTEM breakpoint is currently in effect.

```
C
C  NEXT
```

Because it is so frequently desired to skip ahead to the next CEC scan pickup, without the nuisance of an intervening message, the NEXT command is provided. It is equivalent to a "CONTINUE NEXT" command, except that the SYSTEM breakpoint is temporarily inhibited. The NEXT command is specified as follows:

```
NEXT
```

Finally, the execution of a program is terminated by a STOP command. If a STOP command is issued, execution is immediately terminated. Thus if a program returns to command mode with a message of the form "XACT XXX REQUESTING OUTPUT AT BLOCK NNN", possibly meaningful output will be lost if a STOP command is issued. The STOP command should, therefore, be used with caution. The syntax of the STOP command is as follows:

```
STOP
```

## AN EXAMPLE USING THE GPSS/H INTERACTIVE DEBUGGING FEATURES

Appendix A contains an example of the use of GPSS/H interactive debugging commands. While the example is essentially self-documenting, the following observations are offered:

1. Source and cross-reference listings for the program are shown so the reader can see the model being tested. Ordinarily these listings would not be typed on the terminal.
2. Input lines are typed in lower case and are preceded by a ">", the input prompting character.
3. Command abbreviations become shorter as the example progresses. Thus "continue" becomes "con" and "c".
4. The abbreviations CEC and FEC stand for "Current Events Chain" and "Future Events Chain," respectively.
5. The number of microseconds per block execution is inordinately high because of the CPU time required to do the interactive debugging. If the model were run in non-interactive fashion, the execution rate would be much higher.

## BIBLIOGRAPHY


1. Henriksen, James O. "Building a Better GPSS: A 3:1 Performance Enhancement," Proceedings of the 1977 Winter Simulation Conference.

2. GPSS/Norden Simulation Language, National CSS, Inc., Norwalk, Connecticut.

3. Kiviat, P. J., Villanueva, R., and Markowitz, H. M., Simscript II.5 Programming Language, CACI, Inc., Los Angeles, California.

4. MTS Users Manual, University of Michigan Computing Center, Ann Arbor, Michigan.

## APPENDIX A

```
GPSS/H  PRELIMINARY RELEASE 0.7B8A (UL187)

 LINE# STMT#  BLOCK#  *LOC   OPERATION  A,B,C,D,E,F,G   COMMENTS

 1.000     1            *
 2.000     2            *      BARB1
 3.000     3            *
 4.000     4            *
 5.000     5            *     CLASSIC ONE-LINE, SINGLE-SERVER QUEUEING MODEL
 6.000     6            *
 7.000     7                  SIMULATE
 8.000     8            *
 9.000     9            *     BARBERSHOP SEGMENT
10.000    10            *
11.000    11       1          GENERATE     18,6           ARRIVALS EVERY 18 +- 6 MIN
12.000    12       2          QUEUE        BARBQ          JOIN WAITING LINE
13.000    13       3          SEIZE        BARBR          ENGAGE THE BARBER
14.000    14       4          DEPART       BARBQ          EXIT THE WAITING LINE
15.000    15       5          ADVANCE      15,3           HAIRCUT TAKES 15 +- 3 MIN
16.000    16       6          RELEASE      BARBR          ALL DONE WITH BARBER
17.000    17       7          TERMINATE                   EXIT THE SHOP
18.000    18            *
19.000    19            *     TIMER SEGMENT
20.000    20            *
21.000    21       8          GENERATE     ,,480        SHUT DOWN AFTER 8 HOURS
22.000    22       9          TERMINATE    1              THAT'S ALL, FOLKS
23.000    23                  START       1,,,1           RUN FOR ONE DAY
24.000    24                  END



SYMBOL    VALUE    EQU DEFNS   CONTEXT       REFERENCES BY STATEMENT NUMBER

BARBQ       1                  QUEUE            12     14
BARBR       1                  FACILITY         13     16


READY!
>* Execute 25 blocks.
>step 25
XACT    1 (0444E8) POISED AT BLOCK 5
>display clocks

RELATIVE CLOCK: 80          ABSOLUTE CLOCK: 80
>dis fac(barbr)  que(barbq)

           --AVG-UTIL-DURING--
FACILITY  TOTAL  AVAIL  UNAVL      ENTRIES    AVERAGE   CURRENT  PERCENT   SEIZING  PREEMPT
           TIME   TIME   TIME                TIME/XACT   STATUS   AVAIL     XACT      XACT
   BARBR   .575                          4     11.500    AVAIL    100.0       1

   QUEUE     MAXIMUM      AVERAGE      TOTAL       ZERO     PERCENT      AVERAGE       $
            CONTENTS     CONTENTS     ENTRIES     ENTRIES    ZEROS      TIME/UNIT     TI
   BARBQ          1        0.037          4           3       75.0        0.750
>dis blo(1...7)

BLOCK CURRENT       TOTAL
    1                  4
    2                  4
    3                  4
    4                  4
    5                  3
    6                  3
    7                  3
```

Appendix A (continued)

```
>* Set up some breakpoints.
>bre 2 7
>continue
XACT    1 (0444E8) HAS REACHED BREAKPOINT AT BLOCK 7
>step
XACT    1 (0444E8) DESTROYED AT BLOCK 7
>con
XACT    2 (044658) HAS REACHED BREAKPOINT AT BLOCK 2
>unbreak 2 7
>* Illustrate the SYSTEM pseudo-breakpoint.
>br system
>c
XACT    2 (044658) PLACED ON FEC AT BLOCK 5
>c
XACT    2 (044658) DESTROYED AT BLOCK 7
>* Pick up next CEC transaction.
>next
XACT    3 (0444E8) POISED AT BLOCK 1
>c
XACT    3 (0444E8) PLACED ON FEC AT BLOCK 5
>* Turn on NEXT pseudo-breakpoint to trap CEC scan "pickups."
>br next
>* Note: both the SYSTEM and NEXT pseudo-breakpoints are now enabled.
>c
XACT    3 (0444E8) POISED AT BLOCK 6
>c
XACT    3 (0444E8) DESTROYED AT BLOCK 7
>c
XACT    4 (044658) POISED AT BLOCK 1
>c
XACT    4 (044658) PLACED ON FEC AT BLOCK 5
>c
XACT    5 (0444E8) POISED AT BLOCK 1
>* This is interesting.  The facility is in use, so we'll get blockage.
>dis fac
```

| FACILITY | --AVG-UTIL-DURING-- TOTAL TIME | AVAIL TIME | UNAVL TIME | ENTRIES | AVERAGE TIME/XACT | CURRENT STATUS | PERCENT AVAIL | SEIZING XACT | PREEMPT XACT |
|---|---|---|---|---|---|---|---|---|---|
| BARBR | .650 | | | 7 | 14.857 | AVAIL | 100.0 | 4 | |

```
>c
XACT    5 (0444E8) UNIQUELY BLOCKED AT BLOCK 3
>c
XACT    4 (044658) POISED AT BLOCK 6
>dis cec

CURRENT EVENTS CHAIN

 XACT  ADDR   CURBLK NXTBLK ASMSET CHAIN(S) SDPGCP** PC   MARK-TIME   MOVE-TIME    PRIORIT

    5 0444E8       2      3          CEC      SD                160        ---

PH   1-12  (ZERO)

    4 044658       5      6          CEC                        146        ---

PH   1-12  (ZERO)
>* Note that the "scan skip" indicator is on for transaction number 5.
>unbreak system next
```

```
>* Illustrate tolerance of minor errors.
>d fac ( barbr

          --AVG-UTIL-DURING--
FACILITY  TOTAL  AVAIL  UNAVL    ENTRIES   AVERAGE   CURRENT  PERCENT  SEIZING  PREEMPT
          TIME   TIME   TIME               TIME/XACT STATUS   AVAIL    XACT     XACT
   BARBR   .654                     7       15.142   AVAIL    100.0       4
>d blo ( 1 .. . .. 5

BLOCK CURRENT     TOTAL
    1                8
    2       1        8
    3                7
    4                7
    5       1        7
>* Illustrate error handlins.
>dis cha
NO "CHA" TO DISPLAY
>br 25
"25" IS OUT-OF-RANGE.
>dis blo(3...2)
"3...2" IS AN INVALID RANGE
>br 2
>unbr 2 3 4
"3" IS NOT SET AS A BREAKPOINT
"4" IS NOT SET AS A BREAKPOINT
>* Comrlete the run.
>c
XACT    6 (0445D0) REQUESTING OUTPUT AT BLOCK 9
>* Note: block 9 is a TERMINATE block.
>* "Continue" to set outrut.
>c

RELATIVE CLOCK: 480          ABSOLUTE CLOCK: 480


BLOCK CURRENT     TOTAL
    1               26
    2




                  END

SIMULATION TERMINATED AT BLOCK 9
>* Note: the above outrut was deliberatelw terminated bw attention interrurt.
>* The outrut can still be disrlawed as follows:
>d outrut

RELATIVE CLOCK: 480          ABSOLUTE CLOCK: 480

BLOCK CURRENT     TOTAL
    1               26
    2               26
    3               26
    4               26
    5       1       26
    6               25
    7               25
    8                1
    9                1
```

```
CURRENT EVENTS CHAIN

 XACT   ADDR   CURBLK NXTBLK ASMSET CHAIN(S) SDPGCP**  PC   MARK-TIME   MOVE-TIME    PRIORIT

    7 044580   BIRTH      8         CEC        G                     0        ---

PH   1-12  (ZERO)

FUTURE EVENTS CHAIN

 XACT   ADDR   CURBLK NXTBLK ASMSET CHAIN(S) SDPGCP**  PC   MARK-TIME   MOVE-TIME    PRIORIT

    8 044658   BIRTH      1         FEC        G                     0         489

PH   1-12  (ZERO)

    9 0444E8      5       6         FEC                            476         491

PH   1-12  (ZERO)

           --AVG-UTIL-DURING--
FACILITY  TOTAL  AVAIL  UNAVL      ENTRIES    AVERAGE   CURRENT  PERCENT  SEIZING   PREEMPTI
           TIME   TIME   TIME                TIME/XACT   STATUS   AVAIL     XACT      XACT
   BARBR  .777                          26     14.346    AVAIL   100.0       9

   QUEUE     MAXIMUM      AVERAGE      TOTAL      ZERO     PERCENT     AVERAGE      $A
             CONTENTS     CONTENTS    ENTRIES    ENTRIES    ZEROS     TIME/UNIT    TIM
   BARBQ           1        0.033        26         19       73.0       0.615
>c
SIMULATION TERMINATED AT BLOCK 9
>c
SIMULATION TERMINATED AT BLOCK 9
>step
SIMULATION TERMINATED AT BLOCK 9
REMAINING STEP COUNT = 1
>* Note: the above commands illustrate that there's no continuation once
>* the SIMULATION TERMINATED message is given.
>stop


TOTAL BLOCK EXECUTIONS:          182

MICROSEC/BLOCK AVG CPU TIME: 5157.3

CPU TIME USED (SEC)

PASS1:         0.288
SYM/XREF:      0.034
PASS2:         0.079
LOAD/CTRL:     1.740
EXECUTION:     4.578
OUTPUT:        0.568
```