Proceedings of the 1987 Winter Simulation Conference
A. Thesen, H. Grant, W. David Kelton (eds.)

ADVANCED FEATURES OF GPSS/H

Robert C. Crain
Daniel T. Brunner
James O. Henriksen
Wolverine Software Corporation
7630 Little River Turnpike, Suite 208
Annandale, VA 22003-2653

ABSTRACT

This tutorial describes many of GPSS/H's powerful additions to, and extensions of, traditional GPSS, and shows how modelers can take advantage of them to build simulations that are more sophisticated in their gathering of statistics, more modeler- or user-friendly, and significantly easier to build, modify and debug. The use of language features is illustrated by the examples in Appendices A and B.

1. INTRODUCTION

A 90-minute tutorial, with accompanying handouts and the use of audiovisual equipment, affords the opportunity to discuss the advanced use of GPSS/H in considerable scope and detail. In a paper of reasonable length for these Proceedings, we can only "hit the highlights", giving an overview of the topics to be covered plus additional detail on a selected few. For those unable to attend, we hope that the material presented below, together with the example models in Appendices A and B, will be sufficient for them to work out for themselves some of the detail presented at the session. For attendees, who will receive handouts and copies of transparencies, we hope that this is a useful summary.

Since the first version of GPSS was introduced over twenty years ago, the language has been the subject of ongoing development and enhancement by multiple vendors. Many features have been added or extended in order to provide greater functionality and ease of use, while retaining the intrinsic strengths of the transaction-flow approach. Unfortunately, modelers often do not have the time to keep up with language development. This tutorial has been designed to provide them with an update on GPSS/H capabilities and how they may be used in normal modeling contexts.

Our focus is on three major areas of concern to the modeler: designing and executing simulation experiments, creating enhanced user interfaces, and building and debugging models. Within each of these areas, the use of advanced features and techniques will be presented. Example GPSS/H models, in which the features and techniques are used, are included as Appendices. Because of space considerations, only relatively small, stylized models have been included.

2. DESIGNING AND EXECUTING SIMULATION EXPERIMENTS

Typically, simulationists would like to separate their models from the *use* of those models in controlled experiments. In many versions of GPSS, such a task has been difficult, if not impossible, because execution of the model is governed by (run-) control statements which are limited in both function and flexibility. Traditional GPSS Control Statements are executed exactly in the sequence in which they appear, with neither forward nor backward branching, whether conditional or unconditional. The ability to perform user-specified input or output is also missing in traditional GPSS Control Statements. As a consequence, experiments must be performed manually by editing values and making new runs, or by embedding the desired control structures (*e.g.*, run length) in the model itself.

In contrast, the Control Statements of GPSS/H comprise a complete run-control programming language, in which execution of the model proper may be viewed in the same fashion as calling a (rather large) subroutine. The run-control language can be used to initialize or modify data values, to read from or write to files, to conduct a dialogue with the modeler or user, to conditionally run the model zero or more times, to collect data and statistics, to provide customized output, and -- if the modeler desires -- to call programs written in other languages. Moreover, all this power and flexibility may be used incrementally. When a model is first being developed, it can be run using a very simple set of control statements. Then, as development proceeds, a more powerful run-control environment can be added at the appropriate times.

A second problem that has plagued simulationists, particularly after a model is largely complete and the emphasis is on its validation and use in controlled experiments, is the need to provide multiple independent streams of random numbers for use in different parts of the model (or in the same parts for different runs). Simulation languages ordinarily provide a means for changing the output of random number generators, but typically have no convenient way to ensure that the specified changes produce independent (non-overlapping) streams, unless the user has a detailed understanding of both the algorithm used in a generator and its implementation. Few users have had the time or inclination to gain such an understanding, and rightfully so. Modelers thus were prevented from exercising intelligent control over a very important part of their experiments.

The random number generator currently provided with GPSS/H was implemented with special attention to the above problem. Modelers can straightforwardly control the starting points of an arbitrary number of different random number streams and be guaranteed that the resulting streams will be independent (that they will not be autocorrelated due to overlap). Moreover, GPSS/H will automatically detect any overlap that might occur in a run, providing an extra measure of protection.

In the subsections below, we examine some of the features of GPSS/H that can be used to control execution of a model in a flexible fashion. We also discuss the new GPSS/H random number generator and how it can be used to improve the quality of statistics gathered during a run.

2.1 Using Ampervariables for General-Purpose Storage

Traditionally, GPSS models stored all non-Transaction data in Savevalues or Matrix Savevalues. GPSS/H provides an alternative means of storing such data, via Ampervariables. "Ampervariables" are so called because their names begin with an ampersand ("&"), in order to provide a simple naming scheme and avoid confusion with more traditional GPSS entities. Despite the superficial similarity in names, Ampervariables are not to be confused with traditional GPSS Variables (which actually define expressions), but behave like variables in ordinary programming languages. By that, we mean that they (1) must be declared before their first use, (2) support integer, double-precision floating point, and character data types, (3) are not affected by GPSS CLEAR statements, and (4) may be referred to by name, using simple syntax.

Ampervariables can be used for a variety of purposes. They are the obvious choice for local variables needed by the control language (*e.g.*, loop-control indices). They are also a good choice for containing experimental parameters, since they can be used in Blocks but are not affected by the CLEAR statement. In addition, Ampervariables provide character-string variables and one-dimensional arrays, both of which are missing from traditional GPSS, and which can be appropriate for both Control Statements and Blocks. For example, constructing an interactive dialogue is much easier if character variables are available, and data that is input to a manufacturing model may contain names as well as numeric information.

2.2 Using Advanced Control Statements to Automate Experiments

Good simulation results require multiple runs and careful statistical housekeeping. For example, consider simulating a hypothetical assembly line for automotive doors. One statistic of obvious interest is the number of doors produced per hour, and since the simulation results will typically be used to make management decisions, it is highly desirable for them to specify confidence intervals where possible.

A simulationist might approach this problem by first developing a GPSS model of the assembly line that gathers the doors-per-hour statistic each hour for the normal 40-hour work week, and calculates the mean doors-per-hour value for the week. Using GPSS/H advanced control statements such as DO and ENDDO, the simulationist could then employ the technique of batch means, running the model automatically for a multiple-week period, and collecting data on the mean value of doors per hour for each week. The weekly means of doors per hour *will* be normally distributed (thanks to the Central Limit Theorem), even though the hour-by-hour distribution of doors per hour is not. Having a normally distributed statistic facilitates the construction of the needed confidence intervals, and the GPSS/H control language facilitates gathering the normally

The principal statements that support general purpose programmability in the GPSS/H control language are:

DO ENDDO	DO-loop Iteration

IF
ELSEIF
ELSE
ENDIF
	IF-THEN-ELSE Conditional Branching

GOTO HERE	Unconditional Branching Dummy Branch Target
INITIAL	Assignment Statement for Traditional GPSS Entities
LET	Assignment Statement for Ampervariables
GETLIST GETSTRING	Input Statement (List-Directed Read) Input Statement (Unformatted String Read)
PUTPIC	Output Statement (Picture-Directed Formatted Write)
PUTSTRING	Output Statement (Unformatted String Write)
CALL	Call an External (User-Supplied) Routine

2.3 Using the GPSS/H Random Number Generator

Generating high-quality random variates is crucial in many simulation experiments. The GPSS/H random number generator has been improved over the years, and now uses an implementation of Lehmer's multiplicative congruential algorithm, with parameters selected on the basis of work by Fishman and Moore (1986). This generator has the following properties:

The "next" random sample is produced by applying the algorithm to the most recent sample. In other words, the *n*th sample is produced solely as a function of the *n-1*st sample. The 32-bit string from which the very first sample is produced is known as the *seed* value.

The stream of samples produced repeats after a fixed number of samples. This number is called the *period* of the generator, and in the case of the Lehmer algorithm is 2**31-2, so a supply of over two billion unique random numbers is available.

Values specified as operands of the RMULT statement (or BRMULT Block) are no longer used as seed values. Instead, these values are interpreted as offsets from the starting point of the generator. Thus, an RMULT operand of 50000 means "start with the 50,000th number in the period of the generator". When a model needs multiple, independent random-number streams, a different starting offset value should be used for each stream, so that each "taps in" at different points within the period of the

When the simulationist does not specify starting offsets, a different default starting point is used for each random number stream. The default starting offset for stream number *n* is (100000 * *n*).

Because the generator's seed value (*not* its starting offset) is known, the *n*th sample can be calculated directly from *n* itself. This *very* important property means that one can specify that the generator start with "the one-millionth value in its period" without having to generate and discard 999,999 intervening

The above properties allow the simulationist to provide independent streams simply by knowing approximately how many samples will be needed for each stream in the model, and ensuring that the values supplied in the RMULT statement are correspondingly far apart. If a model needed two streams, with approximately 500,000 samples to be drawn from each, the simulationist could specify RMULT operands of "100000,700000", which would start the second stream 600,000 samples "downstream" from the first,

leaving a comfortable margin before overlap would occur. In addition, the default offsets have been chosen so that simulationists who run models using multiple streams, but who provide neither an RMULT Statement nor a BRMULT Block, are still guaranteed independent (non-overlapping) streams unless their models draw more than 100,000 samples from a given stream. To aid in the determination of appropriate offset values, GPSS/H provides standard output on each stream used in a model, showing the stream's starting position (offset), ending position, number of samples drawn, and chi-square measure of uniformity in the unit interval for the samples drawn.

3. CREATING AN ENHANCED USER INTERFACE

More and more, people want to employ simulation in the day-to-day management of systems, instead of just in their analysis and design. Simulationists are thus faced with the need to provide models that are designed to be used by non-simulationists. This requires, among other things, the ability to provide custom-tailored output, the ability to have the model read in some or all of the data to be used in a run from an external source, and the ability to have the user control the model interactively. GPSS/H has features that enable the simulationist to accomplish all these objectives simply and

3.1 Generating Custom Output

One of the principal reasons we hear for people to use programs outside the simulation language, written in FORTRAN or other languages, is the need to generate customized output. This is completely unnecessary in GPSS/H, which already contains a very flexible, easy-to-use formatted output capability.

For customized output to appear as a standard part of the model itself (while Blocks are being executed), the simulationist need only use the BPUTPIC or BPUTSTRING Blocks. Custom output is also available within the control language, for use before or after execution of the model, via the PUTPIC and PUTSTRING Statements. Both PUTPIC and BPUTPIC use a "picture" type of format specification, which works in such a manner that "what you see is what you get".

3.2 Creating a Data-Driven Model

Another common reason we hear for people to use programs outside the simulation language, written in FORTRAN or other languages, is the need to read input data from external files. This also is completely unnecessary in GPSS/H, which already contains a very flexible, easy-to-use input capability.

Input data that are used as part of the experiment specification, or for purposes of run control, can be read in via the GETLIST Statement before or after execution of the model, while data that are intended as a standard part of the model can be read by the model itself, via the BGETLIST Block. GETLIST and BGETLIST can read from any reasonably well-formatted data file with a minimum of trouble. Both can also handle error conditions and end-of-file

With forethought and (B)GETLIST, the modeler can place any amount of specific model and/or run-control data into external files. This is the same idea as the "experimental frame" notion employed in some other languages, except that it is much more flexible.

3.3 Building a Model for Interactive Use by Non-Simulationists

(B)GETLIST and (B)PUTPIC can be used for reading from and writing to devices such as terminals, as well as files, and character-type Ampervariables facilitate both reading and writing text. Consequently, it is a relatively straightforward task to have a model, or its experimental control program, or both, run with interactive control. When properly designed, such interactively defined and/or controlled models can be used by someone who knows nothing about simulation or programming. Under such circumstances, the interactive definition/control is usually implemented in a data-driven model with custom-tailored output, as described in sections 3.1 and

4. BUILDING AND DEBUGGING MODELS

Building models takes time, and time is expensive. GPSS/H contains numerous features which make the simulationist's task easier, even when dealing with large models.

4.1 Using Improved Syntax

Many, if not most, simulation models are written by one person. A difficult (but all too frequent) problem that arises from this "lone ranger" syndrome is what to do when a model is to be changed, after the modeler is no longer available to modify it. In fact, lack of attention to readability and ease of understanding can cause major problems even for the original modeler, especially when the project is long or the model itself is large.

The improved syntax of GPSS/H offers several opportunities for simulationists to improve the readability, understandability, and maintainability of their models. Three that are worth special attention, because of their broad applicability, are the use of symbolically named Transaction Parameters, the use of expressions as Block (or Control Statement) operands, and the use of Parenthetical ("subscript") Notation.

Symbolically Named Parameters. The attributes of moving objects in a model are stored in a Transaction's Parameters. In traditional GPSS, these Parameters can only be referred to by number, which is a particularly vexing restriction because of the high frequency with which Parameter references tend to occur in a model. In GPSS/H, enormous gains in readability and understandability can be obtained by using symbolic names (rather than numbers) to refer to individual Parameters. Moreover, the modeler can let the GPSS/H compiler automatically assign numeric values to the names used, or can make the assignments manually, using EQU Statements.

Use of Expressions as Block Operands. In traditional GPSS, it was necessary to use GPSS Variables (each defined with a VARIABLE statement), and/or to insert additional Blocks, when a Block operand was to take on a calculated value. That was true even if the calculation was needed for only one Block in the model. Accordingly, the simulationist often was forced to leaf back and forth repeatedly through pages of model source code while modifying and verifying the model.

In GPSS/H, the modeler can code expressions of arbitrary complexity directly in Block operand positions. This simplifies the modeling process and aids readability in the case of "local" calculations, which are specific to a single place in a model, and for which the appropriate documentation is a comment next to the

expression. At the same time, the modeler retains the ability to use a GPSS Variable to define a "global" calculation, which is used widely throughout a model, and which should be documented (and modified, if necessary) in a single place.

A notable special case of real power and convenience arising from the use of expressions as operands occurs in the TABLE definition Control Statement. The A-operand of this Statement defines the statistic that is to be observed and recorded when a TABULATE Block naming that Table is executed during the course of a simulation. In GPSS/H, the statistic can be coded as an arbitrary expression, thus enabling the modeler to quickly collect and tabulate almost anything of interest during a run.

Parenthetical Notation. Unlike most implementations of GPSS, GPSS/H allows indices used in references to Standard Numerical Attributes (SNAs) and Standard Logical Attributes (SLAs) to be coded using parentheses, like subscripts in standard programming languages. For example, PF(NAME) may be used in place of PF$NAME, and MX(PH(MAT),3,3) in place of MX*PH$MAT(3,3). Particularly where indirect addressing is used to specify an entity, parenthetical notation provides a more natural and intuitively understandable syntax. In addition, the basic syntax of parenthetical notation is the same for all SNAs, SLAs and entities, and there are no restrictions on what may appear within the parentheses.

4.2 Interactive Debugging with Source Code Display

The GPSS/H Interactive Debugger is central to rapid model development, verification, and modification. Several simple commands are provided by the debugger for controlling a model's execution and examining its status. Among the more frequently used

STEP	Execute the next Block
STEP n	Execute the next n Blocks
DISPLAY xxx	Display statistics on one or more entities or SNAs
BREAKPOINT yyy	Set a Block Breakpoint (stop when any Transaction reaches Block yyy)
AT yyy	Set a Block Breakpoint with an attached Debugger procedure to be run
CONTINUE	Execute to completion, or to the next Breakpoint
CONTINUE yyy	Execute until Block yyy is reached by any Transaction
TRAP XACT n	Set a Transaction Trap (stop when Transaction n tries to move)
TRAP CLOCK n	Set a Clock Trap (stop when the Clock reaches or exceeds n)
CHECKPOINT	Save the complete state of the model
RESTORE	Return to the CHECKPOINTed state
QQ	Quit Quickly (return to the operating system)

The Debugger can be invoked at the beginning of a run, or (except for batch runs) by interrupting a long-running model to be sure everything is OK before continuing. Usually it is invoked at the beginning of a run; in fact, its execution-speed penalty is so small (less than 5 percent) that many simulationists use it exclusively as their runtime environment for GPSS/H.

The GPSS/H Debugger also supports a "windowing" mode on many of the machines and operating systems on which it runs. This windowing mode, known as "TV" (test video), provides for the display of GPSS/H source code and selected model status information as the model is run. Unfortunately, there is no way to adequately portray TV mode in this paper, but tutorial attendees will have the opportunity to see it demonstrated.

4.3 Using Improved External Subroutine Interface Features

Traditional GPSS required the use of somewhat unwieldy HELP Blocks for calling external subroutines (written in FORTRAN or other languages). Under GPSS/H, external subroutines can be called via HELP (or its variants) for compatibility, but they also can be called far more conveniently via the CALL Statement, the BCALL Block, or in the case of a function that returns a value, by simply including the function by name in any expression.

Although GPSS/H has dramatically improved the GPSS external interface, external subroutines are rarely needed for traditional purposes. Only in cases where GPSS/H's math function library falls short (as it occasionally does), where a modeling algorithm requires a separate program to implement (such as scheduling optimization), or where needed for device-specific I/O does the simulationist need to use an external subroutine.

4.4 Using the GPSS/H Double Precision Floating-Point Clock

The use of a floating-point clock distinguishes GPSS/H from other versions of GPSS. The use of a *double precision*\ floating-point clock distinguishes GPSS/H from most if not all other general-purpose simulation languages.

In general, the GPSS/H double precision floating-point clock offers the following advantages:

"Natural" time units can be used. For example, if a model requires resolution of simulated time to millisecond accuracy, the use of an integer clock requires a time unit of one millisecond (or smaller) per "tick". If one millisecond is used as the time unit, a time value of 3000 represents 3 seconds. Scaled values, such as 3000, are hard to read. With a floating-point clock, a time unit of seconds can be used, so that a time value of 3 represents 3 seconds, and a time value of .001 represents 1 millisecond. Such times are much easier to read.

Much larger times can be realized than with a 32-bit integer clock. For example, if the time unit is one microsecond, the maximum realizable value of a 32-bit integer clock is approximately 0.6 hours.

Much higher precision is provided -- approximately 16 decimal digits, as opposed to 9 digits for a 32-bit integer clock and approximately 7 digits for single precision floating-point. With a time unit of one microsecond, the GPSS/H clock would not suffer loss of precision until after more than 30 ***years*** of simulated time, as opposed to approximately 10 seconds of simulated time for single precision floating-point. For unusually high-resolution models, the GPSS/H clock is capable of simulating 11 days with a time unit of one ***nanosecond***!

Uniform distributions of the form A±B, computed at ADVANCE and GENERATE Blocks, yield an effectively infinite number of sample values. With an integer clock, only 2B+1 sample values are realizable, and the A- and B-operands must be

specified as integer values. This granularity can cause modeling problems, such as being unable to represent a time advance distributed uniformly over the interval from 10 to 15 (inclusive), because the mean value of 12.5 is non-integral.

A double precision floating-point clock also has a few disadvantages:

Double precision floating-point arithmetic is slower than 32-bit integer arithmetic on most computers. The resultant performance degradation in typical GPSS/H runs varies from "hardly noticeable" to about 15 percent, depending upon the particular model and particular computer on which the runs are

Cumulative roundoff errors may arise when using fractions, depending upon whether the fraction has an exact representation in binary floating-point. For example, after a sequence of 10000 "ADVANCE 0.1" Block executions, the clock will not register a value of exactly 1000. This occurs because the fraction 1/10 does not have an exact representation in binary floating-point, just as 1/3 does not have an exact representation in decimal floating-point. Anyone who has performed floating-point arithmetic on a computer has become acquainted with this

In the extremely unlikely case that more than 16 digits (!) of precision are needed, "small" values will behave like zero values. Adding 0.00001 to a value having 12 significant digits to the left of the decimal point requires 17 digits of precision, and hence will produce a result that is unchanged from the original value.

5. SUMMARY

Many of the features being touted as new in discrete-event simulation have been available for years to GPSS/H users. These include sophisticated experimental design capabilities, built-in facilities for creating "no programming" user interfaces, and model-building tools such as a full-screen interactive debugger and high-speed execution. For those unaware of these facilities, or those who have not had the opportunity to explore how they can be used, we hope that this tutorial has been informative. For more detail on the features described in this paper, see Henriksen and Crain (1983).

ACKNOWLEDGEMENTS

Thanks go to Elizabeth Tucker for her thoughtful suggestions and assistance in the preparation of this paper.

APPENDIX A: PROGRAMMING IN THE GPSS/H CONTROL LANGUAGE

The following GPSS "model" shows the general-purpose programming flexibility of GPSS/H Control Statements. Most readers will recognize it as the "Sieve of Eratosthenes" program, which often is used as a computational benchmark. Please note that this perfectly satisfactory GPSS/H program contains no Blocks whatsoever.

```
        OPERCOL         45      ALLOW OPERANDS OUT TO COL 45, FOR INDENTING
        SIMULATE
*****************************************************************************
*       SIEVE OF ERATOSTHENES
*****************************************************************************
*
TRUE    SYN     1
FALSE   SYN     0
SIZE    SYN     8190
*
        INTEGER         &ITER,&I,&K,&COUNT,&PRIME,&FLAGS(8190)
*
*
        DO              &ITER=1,100                 OUTER LOOP
            LET             &COUNT=0                    INIT COUNTER
*
            DO              &I=1,SIZE                   INIT FLAGS
                LET             &FLAGS(&I)=TRUE
            ENDDO
*
            DO              &I=1,SIZE                   INNER LOOP 1
                IF              (&FLAGS(&I)=TRUE)           FIND PRIME
                    LET             &PRIME=&I+&I+3              NEXT PRIME
                    DO              &K=(&I+&PRIME),SIZE,&PRIME  INNER LOOP 2
                        LET             &FLAGS(&K)=FALSE            FLAG N*PRIME
                    ENDDO                                       END INNER LOOP 2
                    LET             &COUNT=&COUNT+1             COUNT PRIMES
                ENDIF                                       END IF PRIME
            ENDDO                                       END INNER LOOP 1
*
        ENDDO                                       END OUTER LOOP
*
*
        PUTPIC          &COUNT                      PRINT OUT RESULTS
 * PRIMES FOUND
        END
```

APPENDIX B: SEMI-COMPREHENSIVE EXAMPLE

The following simple GPSS/H model illustrates a broad variety of the points discussed in the body of the paper. A data-driven model is conditionally executed from an interactive run-control program with custom-tailored output.

```
         SIMULATE
         CHAR*1          &REPLY                  YES/NO RESPONSE GOES INTO &REPLY
*****************************************************************************
*        ONE-LINE, SINGLE-SERVER QUEUEING MODEL
*****************************************************************************
*
         GENERATE        X(AMEAN),X(ASPREAD)     ARRIVALS
         QUEUE           JOEQ                    GET IN LINE
         SEIZE           JOE                     GRAB THE BARBER
         DEPART          JOEQ                    EXIT LINE
         ADVANCE         X(SMEAN),X(SSPREAD)     HAIRCUT TAKES PLACE
         RELEASE         JOE                     FREE THE BARBER
         TERMINATE       0                       EXIT THE SHOP
****
*        TIMER SEGMENT
****
         GENERATE        ,,480,1                 RUN FOR 8 HRS (IN MINUTES)
         TERMINATE       1                       SHUT DOWN
*****************************************************************************
*        INTERACTIVE DIALOGUE FOR INPUT DATA AND RUN CONTROL
*****************************************************************************
****
*        GET IAT DISTRIBUTION FROM THE USER
****
NEXTRUN  PUTPIC
 PLEASE ENTER MEAN INTERARRIVAL TIME AND SPREAD
         GETLIST         END=DONE,ERR=NEXTRUN,(X(AMEAN),X(ASPREAD))
         IF              X(ASPREAD)>X(AMEAN)
             PUTPIC      X(ASPREAD),X(AMEAN)
 ###   SPREAD (*) CANNOT EXCEED MEAN (*).  TRY AGAIN.
             GOTO        NEXTRUN
         ENDIF
****
*        GET SERVICE TIME DISTRIBUTION FROM THE USER
****
GETSTIM  PUTPIC
 PLEASE ENTER MEAN SERVICE TIME AND SPREAD
         GETLIST         END=DONE,ERR=GETSTIM,(X(SMEAN),X(SSPREAD))
         IF              X(SSPREAD)>X(SMEAN)
                 PUTPIC  X(SSPREAD),X(SMEAN)
 ###  SPREAD (*) CANNOT EXCEED MEAN (*).  TRY AGAIN.
                 GOTO    GETSTIM
         ENDIF
         START           1,NP            WE DISPLAY RESULTS OURSELVES
****
*        DISPLAY RESULTS OF INTEREST
****
         PUTPIC          LINES=4,FC(JOE),FR(JOE)/10.0,QT(JOEQ)

   JOE GAVE * HAIRCUTS AND WAS BUSY **.* PERCENT OF THE DAY
   THE AVERAGE TIME IN QUEUE (INCLUDING ZERO-TIME ENTRIES) WAS *.** MIN.

****
*        SEE IF THE USER WANTS ANOTHER RUN
****
ASK4MOR  PUTPIC
 DO YOU WANT TO MAKE ANOTHER RUN?
         GETLIST         END=DONE,&REPLY
         IF              (&REPLY='Y')OR(&REPLY='y')
                 CLEAR
                 GOTO    NEXTRUN
         ELSEIF          (&REPLY!='N')AND(&REPLY!='n')
                 GOTO    ASK4MOR         RE-PROMPT
         ENDIF
DONE     HERE
         END
```

REFERENCES

Fishman, G.S. and Moore III, L.S. (1986). An Exhaustive Analysis of Multiplicative Congruential Random Number Generators with Modulus 2**31-1. *SIAM Journal on Scientific and Statistical Computing* **7**, 1, 24-45.

Henriksen, J. O. and Crain, R. C. (1983). *GPSS/H User's Manual*, Second Edition. Wolverine Software Corporation, Annandale, Virginia.

AUTHORS' BIOGRAPHIES

ROBERT C. CRAIN has been with Wolverine Software since 1981. He received a B.S. in Political Science from Arizona State University in 1971, and an M.A. in Political Science from The Ohio State University in 1975. Mr. Crain is a member of SCS, SIGSIM, and ACM, and served as Business Chairman of the 1986 Winter Simulation Conference.

DANIEL T. BRUNNER received his B.S. in Electrical Engineering from Purdue University in 1980, and his M.B.A. from the University of Michigan in 1986. He has been with Wolverine Software since 1986, and is a member of SCS.

JAMES O. HENRIKSEN is the president of Wolverine Software Corporation, which he founded in 1976 to develop and market GPSS/H, a state-of-the-art version of the GPSS language. Since its introduction in 1977, GPSS/H has gained wide acceptance in both industry and academia. From 1980-1985, Mr. Henriksen served as an Adjunct Professor in the Computer Science Department of the Virginia Polytechnic Institute and State University, where he taught courses in simulation and compiler construction at the university's Northern Virginia Graduate Center. Mr. Henriksen is a member of ACM, SIGSIM, SCS, the IEEE Computer Society, ORSA, and SME. A frequent contributor to the literature on simulation, Mr. Henriksen served as the Business Chairman of the 1981 Winter Simulation Conference and as the General Chairman of the 1986 Winter Simulation Conference.

Robert C. Crain
Daniel T. Brunner
James O. Henriksen
Wolverine Software Corporation
7630 Little River Turnpike, Suite 208
Annandale, VA 22003-2653
(703) 750-3910