Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

EXTENDED FEATURES OF GPSS/H

Robert C. Crain
Daniel T. Brunner
Wolverine Software Corporation
4115 Annandale Road
Annandale, VA 22003-2500

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- and user-friendly, and significantly easier to build, modify, and debug.

1. INTRODUCTION

Since the first version of GPSS/H was introduced over ten years ago, the language has been the subject of ongoing development and enhancement by Wolverine Software. 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 the latest developments in simulation languages. This tutorial will provide simulationists with an update on GPSS/H capabilities and how they may be used to solve modeling problems with less effort.

We will focus on five major areas of concern to the modeler: designing and executing simulation experiments, building models, debugging models, building interactive models, and animating models. Within each of these areas, the use of advanced features and techniques will be covered.

All of the advanced features discussed are described in detail in the third edition of the *GPSS/H Reference Manual,* released this year (and previously called the *GPSS/H User's Manual*). Similarly, GPSS/H Release 2.0 makes the features discussed available on all hardware on which GPSS/H is supported. The features are also available in the newly released student/demonstration version of GPSS/H; many, but not all of them are also discussed in its accompanying tutorial text, *Getting Started with GPSS/H.*

During the conference presentation, numerous examples will be presented and discussed, but they are not included here because of space limitations. Copies of the examples and other material will be provided to attendees, and will be available upon request to those unable to attend. We hope that this paper provides a useful overview of the topics of interest.

2. DESIGNING AND EXECUTING SIMULATION EXPERIMENTS

Simulationists often would like to separate their models from the *use* of those models in controlled experiments. Unfortunately, such a task can be difficult, if not impossible, if model execution is governed by a run-control mechanism that is limited in both functionality and flexibility.

The Control Statements of GPSS/H comprise a complete run-control programming *language*, in which execution of the model proper is akin to calling a rather large subroutine. The run-control language can be used to initialize or modify data values, to read or write to files, to perform calculations, to conduct a dialogue with the modeler or user, to conditionally execute the model zero or more times, and to provide customized output. But perhaps most importantly, all this power and flexibility can be used *incrementally*. A model can be run with a very simple set of control statements when it is first being developed. Then, as development proceeds, a more powerful run-control environment can be added if and as needed.

Another problem important to modelers, particularly after a model is largely complete and the modeler is concentrating on validation and the running of 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 allow their users to alter the output of random number generators, but typically have no convenient way to ensure that the specified changes produce independent, non-overlapping streams of numbers. A modeler seeking to ensure independence is often required to gain a detailed understanding of both the algorithm and implementation used in a particular generator. Few users have the time or inclination to gain such an understanding, and rightfully so. The end result too often has been that modelers were unable to exercise intelligent control over a critical part of their experiments.

The ***indexed Lehmer*** random number generator currently provided with GPSS/H was designed and implemented with special attention to the problem just discussed. Modelers can simply and straightforwardly control any number of random number streams in a model and easily guarantee that they will be independent (that they will not be autocorrelated due to overlap). GPSS/H also provides automatic detection of any overlap that might accidentally occur during a run, providing an extra measure of protection to users.

A third problem that modelers often confront is the need to record a custom-tailored measurement or statistic during model execution. In GPSS/H, the modeler can code expressions of arbitrary complexity directly for most Block and Control Statement operands. A notable special case of real power and convenience arises from the ability to code such expressions as the A-operand of the TABLE definition Control Statement. This operand defines the statistic to be calculated and recorded whenever a TABULATE Block naming that TABLE is executed during the course of a run, or at specified intervals during a run. The modeler is thus able to quickly collect and tabulate almost anything of interest during a run.

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

2.1 Using the Control Statement Language to Automate Experiments

Getting good results from simulations requires multiple runs and careful statistical housekeeping. For example, consider simulating a hypothetical assembly line. One statistic of interest will be the average number of widgets produced per hour, and since the simulation results will probably be used to make management decisions, it is highly desirable to specify confidence limits on the statistics where possible.

A simulationist might approach this problem by first developing a model of the assembly line that records the widgets-per-hour produced for each hour of a 40-hour work week, then calculates the average widgets-per-hour value for each week. With only this basic model, but using GPSS/H Control Statements such as DO and ENDDO, the simulationist could straightforwardly apply the technique of batch means, running the model automatically for a multiple-week period, and collecting data on the average value of widgets-per-hour for each week. The weekly means of widgets-per-hour will be normally distributed (thanks to the Central Limit Theorem), even though the hour-by-hour production figures are not. Having a normally distributed statistic greatly eases the construction of the needed confidence intervals.

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

DO ENDDO	DO-loop iteration

IF
ELSEIF
ELSE
ENDIF
	IF-THEN-ELSE conditional branching

GOTO HERE	Unconditional branching Dummy branch target ("CONTINUE")
INITIAL LET	Assignment of values to data items
GETLIST GETSTRING	Input (list-directed read) Input (unformatted string read)
PUTPIC PUTSTRING	Output (picture-directed formatted write) Output (unformatted string write)
CALL	Call an external (user-supplied) routine

2.2 The Indexed-Lehmer Random Number Generator

GPSS/H now uses an *indexed* implementation of Lehmer's multiplicative congruential algorithm, with parameters selected on the basis of work by Fishman and Moore. This generator behaves as follows:

The "next" random sample is produced by applying the algorithm to the most recent sample. In other words, the *n*th sample is produced purely as a function of the *n-1*st sample. The 32-bit value 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. In the case of the Lehmer algorithm, the period is 2**31-2, so a supply of over two billion unique random numbers is available.

Values supplied by the user (as operands of the RMULT statement or BRMULT Block) are interpreted as offsets from the starting point of the generator. Thus, an RMULT operand value of 50000 means "start with the 50,000th number in the period of the generator." A different starting offset can be supplied for each independent stream used in a model, so that the streams "tap in" at different points in the period of the generator.

Because the generator's true seed value is known, the *n*th sample can be calculated from *n* itself. GPSS/H takes advantage of this *very* special property so that a modeler can specify that a stream start with the one-millionth value in its period without GPSS/H having to generate and discard the first 999,999 values.

GPSS/H's indexed Lehmer implementation thus lets the modeler provide independent streams simply by knowing approximately how many samples will be needed for each stream in the model. To get two streams, with approximately 300,000 samples to be drawn from each, the simulationist might specify RMULT operands of "100000,500000". The second stream would start 400,000 samples downstream from the first, leaving a comfortable margin before overlap would occur. To aid in the determination of appropriate offset values, GPSS/H provides standard output for each stream used in a model, showing its starting offset, ending position, and number of samples drawn during a run.

Note that the RMULT statement is part of the GPSS/H Control Statement language, and allows random number streams to be controlled very flexibly outside of the model proper, as part of the run controls defining an experiment. GPSS/H also provides the ability to save the state of random number streams in a model, so that they can be restarted from the same point in a subsequent run.

3. BUILDING MODELS

Building models takes time, and time is expensive. GPSS/H contains numerous features that make the simulationist's job easier, even (perhaps especially) when dealing with large models.

3.1 Math and Random Variate SNAs

Until recently, doing complicated mathematics and calling mathematically defined random variate (probability distribution) generators in GPSS/H required the use of external routines. In 1988 the following capabilities were built into the language itself:

Math Functions	Distributions
ACOS	RVEXPO (Exponential)
ASIN	RVNORM (Normal)
ATAN	RVTRI (Triangular)
COS	

EXP
LOG
SIN
SQRT
TAN

Other closed-form distributions can be built from the ones provided. Although these features are classified and implemented as Standard Numerical Attributes, each one behaves syntactically like a built-in function that returns a value.

3.2 Using the Double Precision Floating-Point Clock

The use of a *double precision* floating-point simulator Clock distinguishes GPSS/H from most if not all general-purpose simulation languages. In general, the GPSS/H clock offers the following advantages:

"Natural" time units can be used. For example, if a model needs simulated time to be measured with millisecond resolution, the use of an integer clock would require a time unit of one millisecond (or smaller) per "tick". If the time unit is milliseconds, a time value of 3000 represents 3 seconds. Such scaled values are hard to read. With the GPSS/H clock, a time unit of seconds can be used, so that a time value of 3 means 3 seconds and a time value of .001 means 1 millisecond. Readability is vastly improved.

Much larger times can be represented with the GPSS/H clock than with a 32-bit integer clock. If the time unit is microseconds, the maximum representable value with a 32-bit integer clock is only approximately 0.6 hours.

Much higher resolution is provided: approximately 16 decimal digits, as opposed to 9 digits for a 32-bit integer clock and only approximately 7 decimal digits for a single precision floating-point clock. With a time unit of microseconds, 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 a single precision floating-point clock. For unusually high-resolution models, the GPSS/H clock is capable of simulating 11 days with a time unit of ***nanoseconds!***

Uniform distributions of the form A±B, computed at ADVANCE and GENERATE Blocks, yield an effectively infinite number of sample values under the GPSS/H clock.. Contrast this with an integer clock, where only 2B+1 sample values are obtainable, 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. Such problems are completely avoided with the GPSS/H clock.

One caveat associated with a floating point clock is worth noting, however. The use of fractional time units can produce roundoff discrepancies, depending on whether or not the fraction has an exact representation in binary floating-point. For example, a sequence of 10000 executions of an "ADVANCE 0.1" Block will not produce a time advance of exactly 1000. This is 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.

3.3 Using Symbolically Named Transaction Parameters

The attributes of moving or transient objects in a model are stored in a Transaction's Parameters. In traditional GPSS, these Parameters can only be referred to by number. This can be an especially maddening restriction because of the high frequency with which Parameters are referenced in a model.

GPSS/H lets the modeler use symbolic names to refer to individual Parameters, producing tremendous gains in model readability and understandability (and thus in modeler productivity). These names appear on all Transaction output, and different names can be applied to the same Parameter to allow for different contexts. Moreover, the modeler can let GPSS/H automatically assign numeric values to the names used, or can use EQU statements to custom-tailor the assignments.

3.4 Creating Data-Driven Models

Input data that are used for run control, or as part of the experiment specification for a run, can be read in from files via the GETLIST Statement before or after the model proper executes each time. Data that are a standard part of the model itself can be read in during model execution via the BGETLIST Block. GETLIST and BGETLIST read files in which data values are separated by blanks, and can handle integer, double precision floating point, and character data. Both also allow user-specified actions to be taken for error and end-of-file conditions.

3.5 Generating Custom Output

Customized output can be produced while Blocks are being executed by means of the BPUTPIC or BPUTSTRING Blocks. Custom output is also available within the Control Statement 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".

4. DEBUGGING MODELS

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 commands are:

STEP	Execute 1 Block
STEP *n*	Execute *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
CONTINUE	Execute to completion, or to the next Breakpoint
CONTINUE *yyy*	Execute until Block *yyy* is about to be executed by any Transaction

TRAP XACT *n*	Stop when Transaction *n* tries to move
TRAP CLOCK *n*	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 (exit immediately to the operating system)

The Debugger can be invoked at the beginning of a run, or (except for runs in batch) by interrupting a long-running model to be sure all is OK before continuing. Usually the Debugger is invoked at the beginning of a run. In fact, its execution-speed penalty is so small (less than 5 percent) that many modelers 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. The windowing mode, known as TV *(test video)*, displays source code and model status information as the model is run.

5. BUILDING INTERACTIVE MODELS

The GETLIST and PUTPIC Control Statements, and their Block counterparts BGETLIST and BPUTPIC, can be used to read from and write to devices such as terminals (or PC screens) as well as files. Character-type Ampervariables make it easy to manipulate text. Consequently, it is not difficult to have a model or its experimental control program (or both) run with interactive control. No programming in an outside language is necessary.

When properly designed, such interactively defined and/or controlled models can be used by someone who knows nothing about simulation or programming. The interactive definition and control are usually implemented via a data-driven model, as described earlier in section 3.4, in conjunction with custom-tailored output and the appropriate amount of user dialogue to define and control the run. Note that a user dialogue does *not* need to prompt a user for voluminous input data, but rather uses just enough interaction to define unique aspects of the run.

6. BUILDING ANIMATED MODELS

Animation has become a powerful tool in simulation. It is frequently used not only to present simulation results to non-simulationists, but also to aid in model development and debugging.

The built-in I/O features of GPSS/H are well-suited to providing the kind of model trace information that is needed to produce animated output. Figure 1 shows a segment from a GPSS/H model of a Kanban system . The model was developed as part of a comparative simulation modeling exercise conducted by WATMIMS for presentation at the 1989 Winter Simulation Conference, and the segment shown is used to model the movement of AGVs throughout the model. An examination of the model segment will show that only 6 Blocks (annotated with "*ZAP*" in their comment fields) are needed to provide the trace output necessary to allow animation of the AGVs. Obviously, the type of information required by specific animation software will determine the quantity and format of the data to be written out, but the example illustrates how unobtrusively animation capabilities can be added to even sophisticated, data-driven, generic model segments.

The model segment presented in Figure 1 is intended to be substantially self-documenting. Although the segment is included to illustrate the use of GPSS/H features to support animation, it also illustrates a number of the features discussed elsewhere in this paper, and should be read carefully.

7. SUMMARY

GPSS/H has emerged as a leading tool for people who regularly model complex systems. This acceptance comes despite the fact that many of the advanced features of GPSS/H are not well known or (in some cases) well understood. Tutorials such as this one provide GPSS/H users (as well as other observers) with the opportunity to keep up with the latest developments in the language, and to see how easily they can be brought to bear on everyday modeling tasks.

REFERENCES

Banks, J., Carson, J. S., and Sy, J. N. *Getting Started With GPSS/H*, First Edition. Wolverine Software Corporation, Annandale, Virginia, 1989.

Brunner, D. T., and Henriksen, J. O. "A General Purpose Animator,". *Proceedings of the 1989 Winter Simulation Conference* (E. A. Mac Nair, K. Musselman, and P. Heidelberger, eds.).

Crain, R. C., Brunner, D. T., and Henriksen, J. O. "Advanced Features of GPSS/H," *Proceedings of the 1987 Winter Simulation Conference*, 269-275.

Crain, R. C. and Brunner, D. T. "New Advanced Features of GPSS/H," *Proceedings of the 1988 Winter Simulation Conference*, 269-275.

Fishman, G.S. and Moore III, L.S. "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. *GPSS/H Reference Manual*, Third Edition. Wolverine Software Corporation, Annandale, Virginia, 1989.

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 a B.S. in Electrical Engineering from Purdue University in 1980, and an M.B.A. from The University of Michigan in 1986. He has been with Wolverine Software since 1986. Mr. Brunner is a member of SCS, and served as Publicity Chair for the 1988 Winter Simulation Conference.

```
*    MUCH SETUP CODE, INCLUDING INITIALIZATIONS AND FUNCTION DEFINITIONS, HAS BEEN OMITTED FROM THIS LISTING
*    THE KANBAN CODE, WHICH CONTAINS A FEW ANIMATION TRACE FILE WRITES (AGV COLOR CHANGES), IS ALSO OMITTED
*
*    WHEN AN AGV XACT TRANSFERS TO AGVBEGIN, IT HAS ALREADY BEEN (1) IDENTIFIED BY NUMBER AND
          (2) COLOR-CODED BY MISSION BY THE ORIGINATING KANBAN
*    ORIGINATING KANBAN ALSO SETS PH(STNNOW), THE STATION THE AGV IS AT NOW, AND PH(STNDEST), THE DESTINATION
*    THERE ARE 33 CONTROL POINTS AND 10 STATIONS
*    ALL AGVs AT ALL POINTS IN THE SYSTEM SHARE THIS PIECE OF CODE FOR ALL THEIR MOVEMENT
*
*                   PH(CPNOW) IS FUNCTION OF PH(STNNOW)
*                   PH(CPNEXT) IS FUNCTION OF PH(CPNOW) AND PH(STNDEST)
*                   SIMULATED DISTANCE AND XING NUMBER ARE FUNCTIONS OF PH(CPNOW) AND PH(CPNEXT)
*                   ANIMATED PATH NUMBER IS FUNCTION OF PH(CPNOW) AND PH(CPNEXT)
*
AGVBEGIN ASSIGN     CPNOW,FN(CPCALC),PH                      CPNOW IS FUNCTION OF STNNOW (CURRENT STATION)
NEXTCP   SEIZE      PH(CPNOW)+AGVOFFST                       DUMMY END-OF-SEGMENT FACILITY PREVENTS LEAPFROG
         ASSIGN     CPNEXT,MH(ROUTING,PH(CPNOW),PH(STNDEST)),PH    NEXT CP = FUNCTION OF CURRENT CP & NEXT STOP
CHKINT   GATE LR    MH(XING,PH(CPNOW),PH(CPNEXT))+AGVOFFST   WAIT FOR CLEAR INTERSECTION
         GATE SNF   PH(CPNEXT)+AGVOFFST                      WAIT FOR NEXT SEGMENT TO CLEAR
         TRANSFER   SIM,,CHKINT                              BE SURE INTERSECTION IS STILL CLEAR!
         ENTER      PH(CPNEXT)+AGVOFFST                      GRAB NEXT SEGMENT
*
         TEST G     AC1,&ATIME,SAMETYM                       *ZAP* TO SHRINK TRACE FILE, ONLY IF CLOCK CHANGED
         BPUTPIC    FILE=TRACE,(AC1)                         *ZAP* ...DO WE TELL ANIMATOR CURRENT (NEW) TIME
 TIME *.**
         BLET       &ATIME=AC1                               *ZAP* TRACK THE TIME ONLY FOR TEST ABOVE
SAMETYM  BPUTPIC    FILE=TRACE,(PH(AGVNUM),MH(PATH,PH(CPNOW),PH(CPNEXT)))   *ZAP* PLACE ANIMATED AGV ON NEW PATH
 PLACE * ON CP*
*
         RELEASE    PH(CPNOW)+AGVOFFST                       RELINQUISH DUMMY
         LEAVE      PH(CPNOW)+AGVOFFST                       RELINQUISH PREVIOUS SEGMENT
         TEST L     MH(XING,PH(CPNOW),PH(CPNEXT)),LASTXING,NOINT   IF THERE'S AN INTERSECTION...
         LOGIC S    MH(XING,PH(CPNOW),PH(CPNEXT))+AGVOFFST         ...SIGNAL THAT THE INTERSECTION IS BUSY
NOINT    ADVANCE    ML(DISTANCE,PH(CPNOW),PH(CPNEXT))*AGVSPEED     TRAVEL TIME ELAPSES
         LOGIC R    MH(XING,PH(CPNOW),PH(CPNEXT))+AGVOFFST   SIGNAL THAT INTERSECTION IS NOW CLEAR
         ASSIGN     CPNOW,PH(CPNEXT),PH                      MADE IT TO MY NEXT CP!
*
         TEST E     PH(CPNOW),21,NOTBACK                     *ZAP* IF AT CP21 ('WAITING SPUR'), CHANGE AGV COLOR
         BPUTPIC    FILE=TRACE,LINES=2,(AC1,PH(AGVNUM))      *ZAP* CHANGE COLOR TO SHOW COMPLETION OF AGV MISSION
 TIME *.**
 SET * COLOR WHITE
*
NOTBACK  TEST E     FN(CPDEST),PH(CPNOW),NEXTCP              IF AT DEST, THEN DONE, ELSE LOOP TO NEXT CP
         TRANSFER   ,PH(RETURN)                              AGV HAS REACHED DESTINATION; RETURN TO KANBAN CODE
```

Figure 1. AGV Subroutine Code Called by the Kanban Code