All features described below, with the exception of the MAKEOBH option, are available in all versions of GPSS/H, including Student GPSS/H.

Recently Added Features

GPSS/H Release 3.59

Release 3.59 includes fixes for all reported bugs, and a limited set of new features:

- **External Ampervariables (External Routines) are now supported in the form of Windows Dynamic Link Libraries. (See below.)**

Features previously introduced in Release 3.5 include:

- **Source code can be typed in upper and lower case.**
- **User-defined names are case-sensitive; e.g., “JOE”, “joe”, and “Joe” are different names.**
- **Command prompt window support has been improved considerably. (See below.)**

Features previously introduced in Release 3.11 and earlier include:

- RMULT Statement and BRMULT Block now can use "stream=value" notation
- User Chains can now handle up to 2**31-1 Transactions
- Suppressing output from PUTPIC Statement or BPUTPIC Block is much faster
- BLET Block and LET Statement can assign any data type
- Built-in random variates for 23 new distributions
- SCAN and ALTER for User Chains
- Floating-point Parameters OK in SCAN, ALTER, REMOVE
- "ALL" option for selective CLEAR/RESET
- Character-to-numeric data conversion
- CHECKPOINT and RESTORE Statements as alternative to READ/SAVE
- Output files can be opened in APPEND mode
- Output files can be unbuffered or line-buffered
- Character Ampervariables compared to entity names via ENTNUM SCA
- Current date and time available via CURDATE, CURTIME SCAs
- Listing date and time available via LISDATE, LISTIME SCAs
- New "FILE=" names for SYSPRINT, GUSER, and SERCOM
- "YIELD" can be used instead of "BUFFER"
- "LC" can be used instead of "LR"
- New security key
- Project-code security for Runtime GPSS/H
- Command-line options can be abbreviated
- SYSCALL statement/BSYSCALL block
- INSERT compiler-directive
- CHAR SCA
- Extended-ASCII characters in PICTURE output
- QUIET option
- MAKEOBH option
- Change to LINK Block current count
- Less COMMON memory needed for some models
- Improved error messages

COMMAND PROMPT WINDOW SUPPORT IMPROVED

GPSS/H operates as a Windows console-mode application and must be launched from a command line. The installation procedure for GPSS/H creates icons on your desktop for initiating command prompt sessions for the version(s) of GPSS/H you install, and it places a shortcut in the Programs\Wolverine folder. When you click on the icon or invoke the shortcut in the Programs menu, a command prompt session is initiated

The shortcuts created by the installation procedure initiate windowed GPSS/H command prompt sessions. Under Windows 2000 and Windows XP, you can modify the width and height of the command prompt window from its default 80 x 25 size. We strongly recommend that you do so, since a larger window will reduce the need to scroll debugger/output windows.

If you change the size of the command prompt window, set the scrolling region to be the same size as the window itself. Otherwise, confusing "double scrolling" will result. (GPSS/H provides its own scrolling via function keys, described in Table 1. If you make the scrolling region smaller than the window's size, Windows will provide scroll bars.)

Home Key	Scroll output window to the top.
End Key	Scroll output window to the bottom.
Page ↑	Scroll output window up ½ page.
Page ↓	Scroll output window down ½ page.
↑	Scroll output window up one line.
↓	Scroll output window down one line.
→	Scroll output window right 20 columns.
←	Scroll output window left 20 columns.
F3	Enter full-screen output mode.
F4	Enter windowed (debugger) output mode
F5	Scroll source window up.
F6	Scroll source window down.
F7	Scroll source window left.
F8	Scroll source window right.
F10	"Step 1" command

Table 1: GPSS/H TV Mode Predefined Function Keys

RMULT STATEMENT AND BRMULT BLOCK NOW CAN USE "STREAM=VALUE" NOTATION

Release 3.11 provides an improved form of specifying the starting point for Random Number Streams. In earlier releases, the starting points were specified by "counting commas" on an RMULT Statement (or BRMULT Block). So to change the starting point for Stream number 4 from its default value of 400000 to a value of 625000, an RMULT Statement like this would be used:

```
RMULT           ,,,625000  New value for stream 4 in 4th position
```

If the Stream had been given a name, it was necessary to know what number had been assigned to that name in order to "count the right number of commas". Also, if a model used a large number of Random Number Streams, specifying starting positions for them could require great care in coding the RMULT statement. In Release 3.11, starting points can be specified on an RMULT Statement or BRMULT Block using "stream_number=value" notation. So the example above would look like:

```
RMULT           4=625000   New value for stream 4
```

The new notation also accepts names and expressions to designate which stream is being assigned a new starting point:

```
RMULT           JOE=250000
RMULT           (JOE+1)=250000,&STREAMNO=700000
```

The old notation is still completely supported, and both the new and old notation can be used in the same RMULT Statement or BRMULT Block:

```
RMULT           125000,,350000,25=450000
```

But you should be careful if you mix both types of notation. If you accidentally specify two values for the same Stream number, the LAST value specified for the Stream will be used. So in the following example:

```
RMULT           3=350000,,300000,450000  300000 overrides 350000
```

The first value specified for Stream 3 (350000) will be replaced by the 300000 specified in the "third comma" position.

USER CHAINS NOW HANDLE UP TO 231-1 TRANSACTIONS**

User Chains were previously designed to support up to 32,767 Transactions on a Chain at any time. With the abundance of memory now present on even inexpensive computers, it has become possible to simulate systems in which hundreds of thousands of Transactions could be present on a User Chain at any given time. In Release 3.11, User Chains can contain up to 2147483647 Transactions at a time.

SUPPRESSING OUTPUT FROM PUTPIC STATEMENTS OR BPUTPIC BLOCKS IS MUCH FASTER

If a model uses PUTPIC Statements or BPUTPIC Blocks to write to a file, and the file becomes very large, it is sometimes convenient to "turn off" writing to the file in order to save disk space and/or speed up execution. Normally, this is done by using a FILEDEF Statement or a BFILEDEF Block to specify the name of the file as "NUL" (on PC systems) or "/dev/null" (on Unix systems). The operating systems assume that anything written to files named "NUL" or "/dev/null" is meant to be ignored, so no file is created and both space and execution time should be saved.

Unfortunately, it can take longer for the operating system to "throw away" the output than it would take to actually write it to the file. (This is especially true on PCs.) The result is that the disk space is saved but the model runs significantly slower -- enough so that "throwing away" the output is not always a reasonable option.

Release 3.11 of GPSS/H avoids this problem with the operating systems by handling the special filenames itself. If you have a FILEDEF Statement or BFILEDEF Block that specifies either "NUL" or "/dev/null" as the name to be used by the operating system, GPSS/H will not send the operating system any output for that file.

So if you want to make a run WITHOUT creating an output file for (B)PUTPICs that write to FILE=BIGFILE, for example, simply change the FILEDEF for that file so it reads:

```
BIGFILE FILEDEF         'NUL'           on a PC
```

or

```
BIGFILE FILEDEF         '/dev/null'     on a UNIX system
```

The file will not be created, and the model will run even faster than normal.

BLET BLOCK AND LET STATEMENT CAN ASSIGN ANY DATA TYPE

The BLET Block and the LET Statement can now be used to assign a value to ***any*** GPSS/H data item. Unless you need the rarely used range-type assignments, the ASSIGN, SAVEVALUE, and MSAVEVALUE Blocks are no longer necessary. Similarly, the INITIAL Statement is now only necessary

for setting the states of Logic Switches (unless you need range-mode assignment for other data).

For example, it is now permissible (and recommended) to write:

```
     BLET          PF(SAM)=1
     BLET          ML(BOB,1,5)=3.4
     BLET          XH$JANE=31
```

instead of:

```
     ASSIGN        SAM,1,PF
     MSAVEVALUE    BOB,1,5,3.4,ML
     SAVEVALUE     JANE,31,XH
```

Similarly, it is now permissible (and recommended) to write:

```
     LET           ML(BOB,1,5)=3.4
     LET           XH$JANE=31
```

instead of:

```
     INITIAL       ML$BOB(1,5),3.4
     INITIAL       XH$JANE,31
  or
     INITIALML$BOB(1,5),3.4/XH$JANE,31
```

The LET Statement cannot be used to assign values to Transaction Parameters, since there is no active Transaction when Control Statements are executing.

In addition to providing a single, straightforward syntax for assigning values to all GPSS/H data items, the newly-extended BLET Block makes readily apparent the distinction between a Transaction Parameter's name and its value. For example, consider the following two old-style Blocks:

```
   ASSIGN          PF$ALEX,1,PF
   ASSIGN          ALEX,1,PF
```

A modeler might write the first Block (which assigns a value of 1 to the Parameter whose number is given in PF$ALEX), when the intent was to assign a value of 1 to PF$ALEX itself (as is done in the second Block).

Using BLET, assigning a value to PF$ALEX is quite intuitive:

```
     BLET          PF(ALEX)=1  Parenthetical notation preferred
     BLET          PF$ALEX=1   "$" notation also allowed
```

while assigning a value to the Parameter whose number is given in PF$ALEX must be explicitly spelled out:

```
     BLET          PF(PF(ALEX))=1
     BLET          PF(PF$ALEX)=1
```

BUILT-IN RANDOM VARIATES FOR 23 NEW DISTRIBUTIONS:

GPSS/H now supports built-in random-variate generators for 23 additional statistical distributions. All the generators are implemented as SNAs. To use one of the distributions in your model, you need only supply (1) the name or number of a GPSS/H random number stream, and (2) the appropriate parameters to describe the distribution.

Complete information on these "RV" SNAs can be found in Appendix A of the GPSS/H Reference Manual.

The complete set of distributions now supported by GPSS/H Release 3, along with the SNAs that support them, is as follows:

Beta	**RVBETA**(*rno,alpha1,alpha2*)
Binomial	**RVBIN**(*rno,trials,probability*)
Discrete Uniform	**RVDUNI**(*rno,lower_endpoint,upper_endpoint*)
M-Erlang	**RVERL**(*rno,m,beta*)
Extreme Value A	**RVEVA**(*rno,gamma,beta*)
Extreme Value B	**RVEVB**(*rno,gamma,beta*)
Exponential	**RVEXPO**(*rno,mean*)
Gamma	**RVGAMA**(*rno,alpha,beta*)
Geometric	**RVGEO**(*rno,probability*)
Inverse Gaussian	**RVIGAU**(*rno,alpha,beta*)
Inverted Weibull	**RVIWEIB**(*rno,alpha,beta,gamma*)
Bounded-Johnson	**RVJSB**(*rno,alpha1,alpha2*)
Unbounded-Johnson	**RVJSU**(*rno,alpha1,alpha2*)
Laplace	**RVLAP**(*rno,gamma,beta*)
Logistic	**RVLGTC**(*rno,gamma,beta*)
Log-Laplace	**RVLLP**(*rno,alpha,beta*)
Lognormal	**RVLNOR**(*rno,mean,variance*)
Negative Binomial	**RVNBIN**(*rno,successes,probability*)
Normal	**RVNORM**(*rno,mean,standard_deviation*)
Poisson	**RVPSSN**(*rno,mean*)
Pearson Type V	**RVPT5**(*rno,alpha,beta*)
Pearson Type VI	**RVPT6**(*rno,alpha1,alpha2,beta*)
Random-Walk	**RVRWK**(*rno,alpha,beta*)
Triangular	**RVTRI**(*rno,min,mode,max*)
Uniform	**RVUNI**(*rno,mean,spread*)
Weibull	**RVWEIB**(*rno,alpha,beta*)

EXTERNAL AMPERVARIABLES (EXTERNAL ROUTINES)

Beginning with Release 3.59, GPSS/H supports External Ampervariables written in C or C++ and stored in Windows Dynamic Link Libraries (DLLs). Details of how to use External Ampervariables are provided in extamp.pdf. This file is now installed into C:\Program Files\Wolverine\GPSSH\doc. If you are running an old installation of GPSS/H, you can obtain this file and an updated version of GPSS/H from Wolverine.

SCAN AND ALTER FOR USER-CHAINS

Before Release 3, Transactions placed onto User Chains were not directly accessible for purposes of checking or changing their Parameter values. To do so required UNLINKing the Transactions from the Chain and then reLINKing them back on.

To work around this restriction, a Group could be used as a surrogate means of manipulating the members of a User Chain. This would be accomplished by having a Transaction JOIN a Group just before LINKing itself onto the User Chain, and then REMOVEing itself from the Group as soon as it had been UNLINKed. While on the User Chain (and in the Group), a Transaction's Parameters could be checked or changed by having a different Transaction execute SCAN and/or ALTER Blocks to operate on members of the Group as appropriate.

In Release 3, the new SCANUCH and ALTERUCH Blocks provide exactly the same functionality as SCAN and ALTER Blocks, but operate **directly** on User Chains. The SCANUCH and ALTERUCH Blocks are identical to the SCAN and ALTER Blocks, as described in Chapter 5 of the GPSS/H Reference Manual, except that the A-operand for SCANUCH and ALTERUCH must be the name or number of a User Chain.

FLOATING-POINT PARAMETERS OK IN SCAN, ALTER, REMOVE

Release 3 removes all restrictions on the use of Floating-point Parameters in the SCAN, ALTER, and REMOVE Blocks. Floating-point Parameters can now have their values changed, checked for MIN or MAX, or used as comparands. In the entries for the SCAN, ALTER, and REMOVE Blocks in Chapter 5 of the GPSS/H Reference Manual, all references to "Pint" should be replaced by "Pany".

Note: The new SCANUCH and ALTERUCH Blocks also allow the use of Floating-point Parameters as described in the preceding paragraph.

"ALL" OPTION IN SELECTIVE CLEAR/RESET

The format of the exception-list that can be specified in a CLEAR or RESET Statement has been extended. The exception-list contains the names and/or numbers of all entities that should *not* be affected by the CLEAR or RESET. The new extension lets you specify that ALL entities of a given type should be excluded from the CLEAR or RESET without having to specify any of their names. The extension takes the form of the keyword ALL, followed by a parenthesized list of one or more entity types, all of whose members are to be excluded from the CLEAR or RESET:

`CLEAR`	`ALL(XF,ML)`	Exclude all Fullword Savevalues and Matrix Float Savevalues
`CLEAR`	`ALL(XF),ALL(ML)`	Exclude all Fullword Savevalues and Matrix Float Savevalues
`RESET`	`F$JOE,ALL(Q,CH,TB)`	Exclude Facility JOE and all Queues, User Chains, and Tables

CHARACTER-TO-NUMERIC DATA CONVERSION

A GPSS/H character constant or Character Ampervariable that contains an ASCII representation of a number can be converted to an integer or floating-point data item by using the CHARSTOI or CHARSTOF SCA.

CHARSTOI returns its value as an integer. It also requires that the character data item consist only of optional leading blanks or tabs, followed by one or more numeric digits (0-9), followed by optional trailing blanks or tabs. A character representation of a floating-point data item, even if it is integer-valued, will cause a GPSS/H execution error if supplied as input to CHARSTOI.

CHARSTOF returns its value as a floating-point data item. It also requires that the character data item consist only of optional leading blanks or tabs, followed by a valid integer or floating-point number, followed by optional trailing blanks or tabs. A character representation of an invalid integer or floating-point data item will cause a GPSS/H execution error if supplied as input to CHARSTOF.

Examples:

```
INTEGER       &IRESULT
REAL          &FRESULT
CHAR*10       &FLOATNUM
VCHAR*10      &INTNUM

BLET          &INTNUM='11223399'
BLET          &FLOATNUM='-1.6E-15'

BLET          &IRESULT=CHARSTOI(&INTNUM)       OK
BLET          &IRESULT=CHARSTOI(&FLOATNUM)     Error - floating point
BLET          &IRESULT=CHARSTOI(' 123456 ')    OK
BLET          &IRESULT=CHARSTOI('123 456')     Error - embedded blank
BLET          &IRESULT=CHARSTOI('10.0')        Error - floating point

BLET          &FRESULT=CHARSTOF(&INTNUM)       OK
BLET          &FRESULT=CHARSTOF(&FLOATNUM)     OK
BLET          &FRESULT=CHARSTOF(' 123456 ')    OK
BLET          &FRESULT=CHARSTOF('123 456')     Error - embedded blank

BLET          &IRESULT=CHARSTOF('10.0')        OK
BLET          &IRESULT=CHARSTOF(&FLOATNUM)     OK if not too large
```

CHECKPOINT AND RESTORE STATEMENTS AS ALTERNATIVE TO READ/SAVE

Capabilities similar to the checkpoint and restore commands of the GPSS/H interactive debugger are now available via CHECKPOINT and RESTORE Statements. These provide an alternative to READ/SAVE when executing multiple iterations of a model after a warmup period. The two techniques have different strengths and weaknesses, and the differences between them can be summarized as follows:

1. READ and SAVE Statements cannot be used in the same model; READ/SAVE can only be used across completely separate executions of GPSS/H.

   CHECKPOINT and RESTORE Statements ***must*** be used in the same model; CHECKPOINT/RESTORE can only be used within a single execution of GPSS/H.

2. When a SAVEd model is READ in for further processing, entities in the SAVEd model can only be referred to by number (***not*** by name).

   When CHECKPOINT/RESTORE is used, all entities can be referred to by name in the normal fashion.

3. When a SAVEd model is READ in for further processing, ***no*** references can be made to Ampervariables, Pictures, Control-Statement labels, user-defined Files, or External Ampervariables (External Routines).

   When CHECKPOINT/RESTORE is used, ***everything*** can be referred to in the normal fashion.

When a CHECKPOINT Statement is executed by GPSS/H, the state of the model is written out to a temporary disk file known as a checkpoint file. If multiple CHECKPOINT Statements are executed during a run, the model state saved by each one ***replaces*** the state written out by any previous CHECKPOINT Statement(s).

When a RESTORE Statement is executed by GPSS/H, the state of the model ***(except for user-defined files)*** is restored to the state saved by the most recent CHECKPOINT Statement. One or more Statements following the RESTORE can then change model data to prepare for the next iteration. After all model changes have been made, a START Statement resumes model execution.

CHECKPOINT/RESTORE Statements use a different file for saving the model state than is used by the interactive debugger for its checkpoint and restore commands. As a result, the debugger's checkpoint

and restore commands can be used to debug a model that contains CHECKPOINT and RESTORE Statements, provided that reasonable care is taken. Suppose, for example, that a debugger checkpoint command is issued after the model has executed a CHECKPOINT Statement. If the model executes another CHECKPOINT Statement before a debugger restore command is issued, the debugger will restore the original model state but the contents of the CHECKPOINT/RESTORE state file have changed from when the debugger save command was issued. Depending on the model's logic, it is possible for the next RESTORE Statement executed to cause a time jump into the ***future*** instead of the past! If this occurs, GPSS/H will issue an error message and terminate the run. Unfortunately, even though GPSS/H will have detected the out-of-sync checkpoint file, your debugging session is effectively over: you can examine the model's state but cannot continue its execution. Always think through your debugging strategy, and be especially careful when using interactive debugger checkpoint and restore commands that span a model's execution of CHECKPOINT and RESTORE Statements.

Any temporary disk files used for saving the model state, whether created by CHECKPOINT/RESTORE Statements or by the interactive debugger, are automatically deleted when GPSS/H execution terminates.

An example of CHECKPOINT/RESTORE use is given below:

```
     START         1                  Warmup period
     CHECKPOINT                       Save model state at end of warmup
*
     START         1                  Run baseline case
*
     RESTORE                          Restore model state
     LET           &OPTION=1          Change a value to implement case 1
     START         1                  Run case 1
*
     RESTORE                          Restore model state
     LET           &OPTION=2          Change a value to implement case 2
     START         1                  Run case 2
*
     END                              All done
```

OUTPUT FILES CAN BE OPENED IN APPEND MODE

GPSS/H built-in I/O now supports opening an existing file for output so that new data are appended at the end of the file. This is accomplished by adding the keyword APPEND after the external filename on a BFILEDEF Block or FILEDEF Statement for the appropriate file, as follows:

```
     OUTFILE  FILEDEF         'FILENAME.EXT',APPEND
              BFILEDEF       OUTFILE,'FILENAME.EXT',APPEND
```

The following rules apply to use of the APPEND keyword:

1. If used, the APPEND keyword must appear ***after*** the external filename.
2. The APPEND keyword is only valid for output files.
3. The APPEND keyword remains in effect until a subsequent (B)FILEDEF is executed for the same GPSS/H internal filename. If the new (B)FILEDEF does not contain the APPEND keyword, the APPEND attribute will be removed and any subsequent opening of the file for output will take place in standard "overwrite" mode.

OUTPUT FILES CAN BE UNBUFFERED OR LINE-BUFFERED

GPSS/H built-in I/O now supports opening an output file so that data are written using line-buffering, or no buffering at all, instead of the standard operating-system buffering. This is accomplished by adding the keyword LINEBUF or the keyword UNBUF after the external filename on a BFILEDEF Block or FILEDEF Statement for the appropriate file, as follows:

```
OUTFILE  FILEDEF    'FILENAME.EXT',UNBUF
         BFILEDEF   OUTFILE,'FILENAME.EXT',UNBUF
```

The following rules apply to use of the LINEBUF and UNBUF keywords:

1. If used, the LINEBUF or UNBUF keyword must appear ***after*** the external filename.
2. The LINEBUF or UNBUF keyword is only valid for output files.
3. A LINEBUF or UNBUF keyword remains in effect until a subsequent (B)FILEDEF is executed for the same GPSS/H internal filename. If the new (B)FILEDEF does not contain a LINEBUF or UNBUF keyword, the LINEBUF or UNBUF attribute will be removed and any subsequent opening of the file for output will take place using the standard operating-system buffering mode.

Changing the buffering mode for an output file is only needed in unusual circumstances, such as might occur when a GPSS/H model is being used in conjunction with other software and/or with External Ampervariables.

CHARACTER AMPERVARIABLES COMPARED TO ENTITY NAMES VIA ENTNUM SCA

Users who have wanted to read in Character Ampervariables during model execution and compare them to symbolic names used in the model can now do so via the ENTNUM SCA.

The ENTNUM SCA allows character constants or Character Ampervariables to be compared to the symbolic names used in a GPSS/H model. To use ENTNUM, you must specify (1) a character data-item to be compared and (2) the GPSS/H entity type for which names are to be checked. If there is an entity of the specified type with a name matching the character data-item, ENTNUM returns the number of that entity. If no match is found, a value of 0 is returned.

```
          INTEGER    &RESULT
          VCHAR*8    &NAME

                     SEIZE JOE
          BLET       &NAME='JOE'
BNAME1    BLET       &RESULT=ENTNUM(BLO,'BNAME1')  Returns # of this Block
          BLET       &RESULT=ENTNUM(FAC,&NAME)     Returns # of JOE
          BLET       &RESULT=ENTNUM(FAC,'joe')     Returns 0 (joe not caps)
```

CURRENT DATE AND TIME AVAILABLE VIA CURDATE, CURTIME SCAs

The current (instantaneous) date and time may be obtained at any point during a simulation run via the CURDATE and CURTIME SCAs. CURDATE and CURTIME both return their values as GPSS/H varying-length character strings. The format of the date returned by CURDATE is DD MMM YYYY, while the format of the time returned by CURTIME is HH:MM:SS.

Example:

```
          VCHAR*20   &MYDATE,&MYTIME

          BLET       &MYDATE=CURDATE
          BLET       &MYTIME=CURTIME
```

LISTING DATE AND TIME AVAILABLE VIA LISDATE, LISTIME SCAs

The date and time that appear at the top of the standard GPSS/H listing (.LIS) file are now available in a GPSS/H model via the LISDATE and LISTIME SCAs. LISDATE and LISTIME both return their values as GPSS/H varying-length character strings. The format of the date returned by LISDATE is DD MMM YYYY, while the format of the time returned by LISTIME is HH:MM:SS. An example is shown overleaf:

```
VCHAR*20    &MYDATE,&MYTIME
BLET        &MYDATE=LISDATE
BLET        &MYTIME=LISTIME
```

NEW "FILE=" NAMES FOR SYSPRINT, GUSER, AND SERCOM

While GPSS/H built-in I/O has always let a model write output to the screen or to the listing (.LIS) file, the GPSS/H internal file names for these destinations (SERCOM and SYSPRINT, respectively) have hardly been intuitive. Similarly, reading input from the keyboard has required using the internal file name GUSER. In Release 3, new easy-to-remember internal file names have been introduced, as follows:

OUTPUT DESTINATION	NEW NAME	OLD NAME
Screen	`FILE=SCREEN`	`FILE=SERCOM`
Listing (.LIS) File	`FILE=LISTING`	`FILE=SYSPRINT`
INPUT SOURCE	**NEW NAME**	**OLD NAME**
Keyboard	`FILE=KEYBOARD`	`FILE=GUSER`

The new names have been implemented as synonyms, so existing models using the old names will continue to run without modification.

"YIELD" CAN BE USED INSTEAD OF "BUFFER"

The BUFFER Block, and the BUFFER operand of the PRIORITY Block, can now be referred to by a much more intuitive name: YIELD. The new name should help to alleviate confusion sometimes experienced by newer users. Examples of the new and old usage are given below.

New:

```
YIELD                Drop current Xact; rescan Current Events Chain
PRIORITY   10,YIELD  Set Xact's prio to 10; drop it; rescan CEC
```

Old:

```
BUFFER               Drop current Xact; rescan Current Events Chain
PRIORITY   10,BUFFER Set Xact's prio to 10; drop it; rescan CEC
```

The new names have been implemented as synonyms, so existing models using the old names will continue to run without modification.

"LC" CAN BE USED INSTEAD OF "LR"

GPSS/H Logic Switches have two states: SET (true) and RESET (false). On occasion, the use of RESET as a state name can be confused with the use of RESET as a verb: if a Logic Switch is RESET, what value is it reset ***to?*** In Release 3, CLEAR has been provided as a synonym for RESET, so that a Logic Switch can be referred to as SET or CLEAR. Although CLEAR does not appear explicitly in the GPSS/H language, C and LC (for Logic CLEAR) can now be used as Auxiliary Operation Codes in place of R and LR (for Logic RESET). Similarly, the LCj SLA can also be used in place of the LRj SLA.

The following examples show completely equivalent forms:

```
LOGIC R     FRED
LOGIC C     FRED

GATE LR     FRED
GATE LC     FRED

INITIAL     LR$FRED
INITIAL     LC$FRED

TEST E      LR(FRED),1,BLOKNAME
TEST E      LC(FRED),1,BLOKNAME
```

NEW SECURITY KEY

GPSS/H Professional now ships with the same security-key technology used for Proof and Run-time GPSS/H. The new (tan) key allows GPSS/H Professional to run under Windows 95/98, ME, NT4, 2000, and XP.

PROJECT-CODE SECURITY FOR RUNTIME GPSS/H

Run-time GPSS/H now allows model-providers to restrict a pre-compiled GPSS/H model so that it can be run only with a Run-time GPSS/H security key that contains a specific project-code. Contact Wolverine for details.

COMMAND-LINE OPTIONS CAN BE ABBREVIATED

All command-line options used when invoking GPSS/H can now be abbreviated to the maximum extent possible. The minimum number of characters needed to specify a particular option depends on the option itself and how similar it is to other options. The options and their abbreviations are given below. See Table 1 of the GPSS/H Professional System Guide for a description of each option.

Option	Minimum Abbreviation	Comments
DICT	DIC	
DIFFTST	DIF	
ERR	E	
MEMSTATS	ME	
NODICT	NOD	
NOERR	NOE	
NOLIST	NOL	
NOQUIET	NOQ	
NOSOURCE	NOS	"NOS" is special case for NOSOURCE
NOSIM	NOSI	
NOUPLIST	NOU	
NOWARN	NOW	
NOXMSG	NOXM	
NOXREF	NOXR	
NOX	NOX	
MAKEOBH	M	
QUIET	Q	
SIZE=A	SIZE=A	Obsolete
SIZE=B	SIZE=B	Obsolete
SIZE=C	SIZE=C	Obsolete
SOURCE	S	"S" is special case for SOURCE
TEST	TE	
TPS	TP	
TTNW	TTNW	
TTSL	TTSL	
TV	TV	
TVTNW	TVTNW	
TYPE	TY	
UPLIST	UP	
WARN	WA	
XMSG	XM	
XREF	X	"X" is special case for XREF
XSORT=A	XS	
XWF	XW	

SYSCALL/BSYSCALL:

Modelers sometimes need to run other software during the execution of a model. For example, after a model runs for a period of simulated time, it might be desirable to run specialized user-interface software. Such software could present intermediate results to the model user, and then allow various model parameters to be changed before the run resumes. Alternatively, intermediate results could be examined by optimization software, which would then determine new parameter values to be used by the model when it resumed running.

In each of the above cases, it is both simple and natural to have the GPSS/H model communicate with the other software by means of data files. The model can write appropriate data to files during execution, then execute a CLOSE or BCLOSE for each such file before starting up the other software. The other software would open the data files as input, perform whatever work is to be done, and write into output files whatever data is to be passed back to the model. The other software would then close all files and terminate. Upon resuming execution, the GPSS/H model would use GETLIST or BGETLIST to open the files written by the other software and read them, as appropriate, during further execution.

The use of ASCII text files as a means of linking the model to the other software has several advantages:

- Both the GPSS/H model and the other software can be written using the input/output features that are familiar in each. No unusual features or special programming tricks are required.
- Debugging is vastly simplified. The modeler can easily inspect the ASCII text file(s) produced as output by the GPSS/H model to verify their correctness. The output file(s) produced by the other software can also be easily inspected and verified. ***Very importantly***, the GPSS/H interactive debugger can be used to debug the GPSS/H model, while the other software can be debugged separately from the model run, using ***its own*** native debugger.
- Portability is maximized. If both GPSS/H and the other software can be run on a different platform, their interaction via text files will also work. No complicated, operating-system-specific tricks are necessary to accomplish the interaction.
- In order to support the type of interaction between models and other software described above, the SYSCALL statement and BSYSCALL Block have been added to GPSS/H. As described below, they allow other software to be executed from a running GPSS/H model.

The SYSCALL control statement and BSYSCALL Block allow a DOS command to be executed from within a GPSS/H model. Both statements may be labeled, and both take a single operand. The operand must be a character constant (enclosed in *single* quotation marks) or a character ampervariable. The contents of the character constant or ampervariable are passed *unchanged* to DOS as a command to be executed. After the command has finished executing, control returns to GPSS/H, which ***immediately repaints the screen*** to restore its GPSS/H context.

NOTE: Error-codes returned by programs executed via SYSCALL/BSYSCALL are not available to GPSS/H Professional or to your model.

NOTE: When using SYSCALL/BSYSCALL, do not try to access any files used by GPSS/H or your model ***that are still open.*** For example, you could use an editor to examine the model source (.gps) file, because it has been closed by GPSS/H before model execution begins. If a listing (.lis) file is being produced, however, it will be ***open*** throughout execution, and should not be accessed. If accessed for reading, the file may appear to be empty; if accessed for writing, the file may become corrupted. Files that are accessed via GPSS/H extended I/O may be open or closed, depending on model execution. Once a file has been read from or written to via GPSS/H extended I/O during execution, the file ***remains open*** until the end of the run, unless a CLOSE or BCLOSE is explicitly executed for that file.

INSERT COMPILER DIRECTIVE

The INSERT compiler directive provides the capability to import model code from multiple files when compiling a model. It is very similar to the "#include" preprocessor directive found in the C language. Its syntax is as follows:

```
INSERT  filename
INSERT  "filename"

INSERT  <filename>
```

When the filename is given either by itself or inside quotation marks, GPSS/H will *only* look in the current directory for the file to be INSERTed. If the filename is placed inside angle brackets, GPSS/H will first look in the current directory for the file, and then, if the file was not found, will look for the file in the list of directories specified in the `HINSERT` environment variable, if one has been established. Directories are specified in the `HINSERT` environment variable separated by semicolons, just as they would be specified in a `PATH` environment variable:

```
set HINSERT=C:\MYDIR1;C:\MYDIR2
```

INSERTed files can themselves contain other INSERT directives (inserted files can be "nested"). The only restrictions on INSERT operations are that:

- A file already in the process of being INSERTed cannot be referred to by an INSERT directive in a lower-level file.
- The maximum depth to which INSERT directives can be nested is 12 if a ".lis" file is being produced, or 13 if it is not (if the HPRO command specifies the TYPE option).
- An INSERT directive may not be continued onto another line, because the underscore character used by GPSS/H to indicate continuation is valid for use in filenames under many operating systems.

All forms of the filename (filename, "filename", <filename>) on an INSERT directive can be followed, if desired, by either of the optional keywords LIST or NOLIST:

```
INSERT  filename,LIST
INSERT  filename,NOLIST
```

The LIST keyword forces the INSERTed file to be included in the ".lis" file produced by GPSS/H.

The NOLIST keyword suppresses listing of the INSERTed file in the ".lis" file.

If no keyword is specified, the INSERTed file will appear in the ".lis" file if listing is in effect for the current file (the file that contains the INSERT directive). Otherwise, it will not.

Once a LIST or NOLIST keyword has been used, it remains in effect for lower-level files unless a lower-level INSERT directive explicitly specifies the opposite keyword.

After GPSS/H has finished INSERTing a file, it re-establishes (if necessary) the LIST/NOLIST/none keyword that was in effect before the INSERT directive was processed.

CHAR SCA

The CHAR SCA allows any ASCII character having a decimal value from 0 to 255 to be specified as an integer. This allows ***any*** desired ASCII character to be used in GPSS/H extended I/O statements.

The CHAR SCA takes an integer expression valued from 0 to 255 as an argument, and returns the single ASCII character having the same decimal value as the argument. The character is returned as a GPSS/H VCHAR*1 data item.

For example, CHAR(10) returns an ASCII linefeed, CHAR(12) returns an ASCII formfeed, CHAR(42)

returns an ASCII *, CHAR(169) returns the upper-left-corner box-drawing character, and so on.

EXTENDED-ASCII CHARACTERS OK IN PICTURES

Extended-ASCII characters (those having decimal values from 128 to 255)can now be used directly in the pictures associated with (B)PUTPIC output. This allows the use of such niceties as line- and box-drawing characters for better-looking tabular output. No conversion is performed on such characters, though, so ***be sure that your printer handles them properly*** before firing off a print run.

"QUIET" OPTION

A new command-line option ("quiet") causes GPSS/H Professional to run a model without writing any of its normal output to the screen. This allows a "canned" model to be run, under the control of a batch file or user-provided "front-end" program, without the person running the model being aware that GPSS/H is running. See page 14 of the System Guide for further details.

"MAKEOBH" OPTION

A new command-line option ("makeobh") causes GPSS/H Professional to write out to disk a fully-compiled model. The resulting file, which has a file type of ".obh", can be run repeatedly by GPSS/H Professional without re-compilation.

To ***produce*** the pre-compiled model with GPSS/H Professional, you must use a command of the form:

```
hpro fname.gps [options...] makeobh
```

This will cause the model to be compiled and then written to disk as "fname.obh" in the directory containing "fname.gps". The model will ***not*** be run when the "makeobh" option is used.

To ***run*** the pre-compiled model with GPSS/H Professional, you must use a command of the form:

```
hpro fname.obh [options...]
```

where the name of the file to be run ***explicitly contains the ".obh" extension.*** Note that only command-line options that affect model ***execution*** should be specified when running a ".obh" file. Options that affect compilation will not have any effect, since the model has already been compiled.

This capability has been added primarily in support of Runtime GPSS/H—a special version of GPSS/H Professional that can ***only*** execute such pre-compiled models. Runtime GPSS/H provides a cost-effective way to distribute a model to one or more users who will need only to run the model with different inputs—not to change it. In addition, a ".obh" file is a ***binary*** image of the model ***after*** compilation, so a proprietary model may be distributed with no risk of its user seeing the original GPSS/H source.

CHANGE TO LINK BLOCK CURRENT COUNT

Transactions that LINK themselves onto a User Chain now show up in the Current count of the LINK Block that they executed in order to get onto the Chain. They remain included in the Current count until they are UNLINKed from the User Chain by some other Transaction.

LESS "COMMON" MEMORY NEEDED FOR SOME MODELS

The management of COMMON has been changed for models that use large numbers of Transactions of different sizes. Previously, COMMON that was used to create Transactions of a given size was only reusable to create additional Transactions of the same approximate size. Beginning with Release 2.05, any COMMON available from TERMINATEd Transactions is "recycled" when no other COMMON is available to meet a need. This change may allow some models to be run with less COMMON than was previously required.