

# GPSS World Reference Manual

## Table of Contents

### Preface xvii

**Product Description**

**About This Manual**

### Chapter 1 - Introduction

1.1. Highlights

1.2. GPSS World Concepts

1.3. Architecture

1.4. The Modeling Language

1.5. Compatibility

### Chapter 2 - Operating GPSS World

2.1. Installation

2.2. The GPSS World Environment

2.3. Controlling a Session

### Chapter 3 - Model Statements

3.1. Using Model Statements

3.2. GPSS Statements

3.3. Fields

3.4. Expressions

3.5. Names

3.6. Numbers

3.7. Using Strings

### Chapter 4 - GPSS Entities

4.1. Transaction Entities

4.2. Block Entities

4.3. Facility Entities

4.4. Function Entities

4.5. Logicswitch Entities

4.6. Matrix Entities

4.7. Queue Entities

4.8. Storage Entities

4.9. Savevalue Entities

4.10. Table Entities

4.11. Userchain Entities

- [4.12. Variable Entities](#)
- [4.13. Numeric Group Entities](#)
- [4.14. Transaction Group Entities](#)
- [4.15. Random Number Generators](#)
- [4.16. Data Streams](#)
- [4.17. Continuous Simulation](#)

## [Chapter 5 - GPSS World Windows](#)

## [Chapter 6 - Commands](#)

[BVARIABLE](#)

[CLEAR](#)

[CONTINUE](#)

[EQU](#)

[EXIT](#)

[FUNCTION](#)

[FVARIABLE](#)

[HALT](#)

[INCLUDE](#)

[INITIAL](#)

[INTEGRATE](#)

[MATRIX](#)

[QTABLE](#)

[REPORT](#)

[RESET](#)

[RMULT](#)

[SHOW](#)

[START](#)

[STEP](#)

[STOP](#)

[STORAGE](#)

[TABLE](#)

[VARIABLE](#)

## [Chapter 7 - Block Statements](#)

[ADOPT](#)

[ADVANCE](#)

[ALTER](#)

[ASSEMBLE](#)

[ASSIGN](#)

[BUFFER](#)

[CLOSE](#)

COUNT  
DEPART  
DISPLACE  
ENTER  
EXAMINE  
EXECUTE  
FAVAIL  
FUNAVAIL  
GATE  
GATHER  
GENERATE  
INDEX  
INTEGRATION  
JOIN  
LEAVE  
LINK  
LOGIC  
LOOP  
MARK  
MATCH  
MSAVEVALUE  
OPEN  
PLUS  
PREEMPT  
PRIORITY  
QUEUE  
READ  
RELEASE  
REMOVE  
RETURN  
SAVAIL  
SAVEVALUE  
SCAN  
SEEK  
SEIZE  
SELECT  
SPLIT  
SUNAVAIL  
TABULATE

TERMINATE

TEST

TRACE

TRANSFER

UNLINK

UNTRACE

WRITE

## Chapter 8 - PLUS

8.1. Defining PLUS Procedures

8.2. The Language

8.3. The Procedure Library

    8.3.1. Utility Procedures

    8.3.2. Math Procedures

    8.3.3. Query Procedures

    8.3.4. String Procedures

    8.3.5. Probability Distributions

## Chapter 9 - Advanced Topics

9.1 Transaction Chains

9.2 The Transaction Scheduler

9.3 Synchronization

9.4 Preemption and Displacement

## Chapter 10 - Performance Tips

10.1 Memory Allocations

10.2 Identifying Congestion Points

10.3 Operating Tips

10.4 Modeling Tips

## Chapter 11 - Standard Reports

11.1 Report Management

11.2. Sample Report

11.3 Standard Report Items

## Chapter 12 - Statistics

12.1 Introduction

12.2 ANOVA

12.3 RESET

12.4 Space-Time Products

## Chapter 13 - Experimentation

13.1 Introduction

13.2 Experimentation and the Analysis of Variance

13.3 GPSS World Features

## [13.4 The Automatic Experiment Generators](#)

### [13.5 Operating Tips](#)

## [Chapter 14. Error Messages](#)

### [14.1 Introduction](#)

### [14.2 Messages and Explanations](#)

## [Appendix](#)

### [1.1. GPSS Grammar](#)

### [1.2. PLUS Grammar](#)

### [Glossary](#)



[Home](#)



[Up](#)

# **GPSS WORLD REFERENCE MANUAL**

**◆ Copyright 2001 Minuteman Software.**

**Holly Springs, NC, U.S.A.**

**All Rights Reserved.**

The software described in this manual is furnished under license and may be used or copied only in accordance with the terms of the license agreement.

**Fourth Edition 2001**

Minuteman Software

P.O. Box 131

Holly Springs, NC 27540-0131 U.S.A.

[www.minutemansoftware.com](http://www.minutemansoftware.com)

## **Product Description**

GPSS World™ is a high powered general purpose computer simulation environment, designed for simulation professionals. It is a comprehensive modeling tool covering both discrete and continuous computer simulation, with an extremely high level of interactivity and visualizability.

Using GPSS World, it is possible to predict the effects of design decisions on extremely complex real world systems.

## **About this Manual**

This manual contains information on the use of the GPSS World application program, and is the primary reference for the version of the GPSS language implemented by GPSS World. The first part of the manual shows you all you need to know in order to make full use of the features of the GPSS World application program. The last part of the manual contains a detailed description of each GPSS Statement and a variety of information that may be useful as you become experienced with the use of GPSS World. Most of this manual is accessible via the online Help facility of GPSS World.

A companion manual, entitled *GPSS World Tutorial*, is included in the documentation set. It contains an introduction to simulation with GPSS, followed by suggested sessions involving many different sample models. You may find one or more examples that correspond closely to your own simulation problems.

## **How To Get Started**

If you are familiar with GPSS and do not wish to go through all the features offered by GPSS World, after you have installed GPSS World according to the instructions in Chapter 2, you should work through the manual entitled *GPSS World™ Tutorial*.

Then, as questions arise, refer to the Statement descriptions in Chapters 6, 7, and 8 of this manual. The Online Help feature can save you the trouble of looking up the details of individual Statements. You can start the Online Help system by pressing [F1] , by pushing a Help button, or by choosing

from the Help menu. You can reach context sensitive help by placing the menu selection cursor over the menu item, and then pressing [F1].

To get the maximum benefit of GPSS World, you need to be aware of all it has to offer. This is best done by working through the *GPSS World Tutorial Manual*, and then the *GPSS World Reference Manual*. We suggest that you experiment with new features until you are comfortable with them.

## The Parts of This Manual

This manual begins with general information. It then proceeds to material of increasing detail.

- ◆ Chapters 1 and 2 contain introductory material about GPSS World, the manual, and some concepts you will need.
  - ◆ Chapters 3 and 4 describe the most important aspects of the GPSS language.
  - ◆ Chapter 5 contains the detailed descriptions of the windows of GPSS World.
  - ◆ Chapters 6, 7, and 8 comprise the reference section. They contain detailed descriptions of each of the Commands, Block Statements, and PLUS Facilities, respectively.
  - ◆ Chapter 9 contains many details of the programming of GPSS World. Generally only experienced GPSS modelers will use the material in this chapter.
  - ◆ Chapter 10 contains information on how to get the most from your simulations. It contains performance tips and other information you may find useful.
  - ◆ Chapter 11 describes the Report Management Facilities, and the Standard Reports produced by GPSS World.
  - ◆ Chapter 12 discusses the statistics generated automatically by GPSS World.
  - ◆ Chapter 13 contains the experiment support in GPSS World. In addition to the multiway ANOVA procedure for user designed experiments, it describes the powerful Screening and Optimizing Experiment generators.
- Chapter 14 contains the error messages, explanations, and remedial actions.
- ◆ Finally, the Appendix contains a description of the formal grammar, and a short glossary.

Some readers will appreciate the formal grammar description in the Appendix. Most of the elements of the GPSS dialect used with GPSS World are defined formally there. The forms of an operand which are acceptable are given under the appropriate Statement description in Chapter 6, 7, or 8.

## How This Manual Describes Actions

Concepts with meaning specific to GPSS World are capitalized. For example, the Current Events Chain is a specific construct with a special meaning. When words are used generically, and do not refer to a particular item, they are not capitalized.

Actions you are asked to make immediately, are represented by verbs in uppercase. For example, when you see PRESS, CHOOSE, CLICK, DOUBLE CLICK, SELECT, or TYPE, you are to perform the action, yourself. When actions are referred to, but you are not asked to do them immediately, the verbs are shown in lower case.

Words representing objects or denoting concepts specific to GPSS World, are often shown in italics for emphasis, when first encountered, or capitalized when used frequently. For example, SHOW is an Immediate Command.

Keystrokes you are required to make are indicated by a special font which encloses the name of the key in an outline. For example, "PRESS [Enter]" means that you are to press the key labeled "Enter" on your keyboard.

When more than one key must be used, the key symbols are separated by a plus. For example, [Ctrl] + [Alt] + [H] means that you are to press and hold the left key, while you press and release the right ones.

Mouse actions are required when the capitalized phrases CLICK or DOUBLE CLICK appear. This means that you are to use mouse button 1 to perform the action. When mouse button 2 is needed, it will be called for explicitly. By default, mouse button 1 is the left mouse button, and 2 the right. You can use the OS/2 control panel to switch these assignments.

Menu items are indicated in a bold **Times New Roman** font. When more than one level of selection is necessary, they are shown in the order: menu bar item / pull down menu item / cascade menu item. For example

### CHOOSE FILE / OPEN

means that you should use either the mouse or the keyboard to choose the "File" item in the menu bar. Then you should choose the "Open" item in the pull down menu. In this example, there is no cascade menu item. For purposes of nomenclature in this manual, "CHOOSE" is used for menu items, "SELECT" is used for other choices.

Dialog window controls are also shown in a bold **Times New Roman** font. For example,

### SELECT OK

means that you should use either the mouse or the keyboard to select the OK button in the dialog window.

When you see SELECT **OK**, you can accomplish the same thing pressing [Enter] on your keyboard. There is one exception, however. Since **Command / Custom** provides for multiline command lists and PLUS Procedure definitions, the enter key is used only to break text lines, instead of to SELECT **OK**.

Characters you are to type into a Text Window are indicated in a bold **Courier New** font. For example,

### TYPE GENERATE 100

means that you are to type the characters "GENERATE 100", making sure the text window has the focus, and that the text cursor has been positioned to the right place in the window.

Finally, objects in the operating system shell are described with underlining, in order to distinguish them from items within GPSS World, itself. For example,

### DOUBLE CLICK on The GPSS World Icon

means that you are to view the desktop or a folder of the Workplace shell containing the icon which represents the GPSS World Session Object, and you are to position the mouse pointer over it and double click mouse button 1.

## Menu Operation

You can use either the mouse or the keyboard to choose menu items. In multistep selections, a selection cursor appears to show you which item is about to be chosen. A selection cursor is a dotted line or a "pressed" appearance around a menu item. The [Alt] key, when pressed, will select the first item in the main menu give it a "pressed" appearance. The arrow keys can then be used to move back and forth over the menu items. The [Enter] key makes the final selection.

## Using the Mouse

There are two ways to use a mouse to choose a menu item. You can position the mouse pointer over the item to be chosen, and then press and release mouse button 1. You can do this in the menu bar, a pull down menu, or a cascade menu.

Alternately, you can press mouse button 1 and not let up immediately. Instead, you can drag over the item in the pull down menu, and over the item in a cascade menu to be chosen, and then release the button. You may find this combined action quicker, although there are keyboard shortcuts, as well.

## Using the Keyboard

The keyboard alone can be used to make menu choices. If any menu actions are in process, press **^**. Then press **a** to position the menu selection cursor on the first item in the menu bar.

Now you can move the selection cursor by using the arrow keys, [Home] or [End]. Press [Enter] to make the final menu item choice.

Alternately, you may be able to simply press a mnemonic key. These keys are denoted by the underlined letters occurring in the menu items. After entering keyboard selection mode by PRESSing [Alt] you can press mnemonic keys to choose succeeding menu items.

Finally, some items have shortcut key combinations listed in the menu, to the right of the item. You can immediately choose a menu item, simply by pressing this key combination. Of course, for you to be able to do this, that window has to have the focus.

## Text Windows

When a GPSS World Text Window has the focus, keystrokes usually cause the replacement or insertion of characters into the window. The [Ins] key can be used to change between insert mode and replace mode. The mode is indicated by the shape of the blinking text cursor, which also indicates the insertion point in the window.

Either the keyboard or the mouse can move the text cursor. Clicking mouse button 1 moves the text cursor to the position under the mouse pointer. The arrow keys, [Home], [End], [PgUp], and [PgDn] all move the text cursor, as does normal text insertion and deletion.

A new line of text can be started by pressing [Enter].

Selected text is shown in inverse, and can take part in special operations. You can select text by dragging mouse button 1 over the target text, by pressing j, and clicking mouse button 1 on the end of selected text, or by double clicking on the word to be selected.

You can delete characters, one at a time, by pressing [Del] or backspace. Pressing [Del] deletes the character to the right of the cursor; pressing backspace deletes the character to the left of the cursor. Selected text can be deleted by choosing **Edit / Cut**, or by pressing [Del]. A single level undo is available by choosing **Edit / Undo**.

## Graphics Windows

Graphics Windows may have menu actions based on the selection of one or more objects in the window. For example, a contiguous sequence of Blocks can be copied from the Block Input Window to the clipboard.

Generally, an object can be selected by clicking on its icon. Swipe selection is supported, as well. In addition, some windows allow extended selection. First select an object. Then scroll to the other end of the selection set of objects, hold down [Shift] while clicking on the second object. That object, the first object, and all intervening objects will then be selected.

Finally, if you hold down [Ctrl] while clicking on an icon, the object is toggled into or out of the selection set. A selected object becomes unselected, or an unselected object becomes selected.

## Dialog Windows

Although the GPSS World dialog windows are all different, they do have some things in common.

Mouse selection is normally used to set the focus in entry fields, and make selections in each dialog. Usually, the [Enter] key will select **OK**, the [Esc] key will select **Cancel**, and the [F1] key will select **Help**.

## Getting Help

Online Help is available when you

CHOOSE **Help**

in a dialog window, or when you

PRESS [F1]

Once the Online Help system is activated, you can utilize a variety of search techniques for information. If you have selected a word in a Text Window, the Help Window will open directly in the topic in the Online Reference Manual.

## **Universal Keys**

Several keys, and key combinations are effective no matter which GPSS World window has the focus. They are generally used to control a translated simulation by quickly passing Commands to the active Simulation Server.

### **Hot Keys**

The Hot Keys initiate an Interactive Command from a single keystroke combination. These are described in Chapter 2.

### **Function Keys**

Function keys can be given ad hoc assignments. You can do this in the Settings Notebook by typing in the new assignment beside the appropriate function key label. Thereafter, when you press the associated function key, the assigned Command is Translated and sent to the Server. You can assign a complex Command List and/or PLUS Procedure redefinition by assigning an INCLUDE Statement to a function key.

## **Intended Audience**

GPSS World is a full strength simulation environment designed for simulation professionals. A significant amount of preparation is required before you can expect to use it to its full potential. This manual is intended for users who are already familiar with the basic elements of the GPSS language. In addition, you are expected to know how to operate your personal computer system. You are expected to know about the file system, disk drives, directories, diskettes, desktop objects, and how to use a mouse. Also, you are expected to know how to operate menus and dialogs by mouse and/or keyboard. This information is available in the documentation of your operating system.

If you are a GPSS/PC user, be sure to read the section entitled GPSS World Features in Chapter One. It goes over a detailed list of features and changes from the old GPSS/PC environment. More information for each item exists in later chapters. Be sure to refer through the index when you need to know more about any individual item. Online Help and the online Reference Manual are available, as well.

If you are new to GPSS, the General Purpose Simulation System, you should do some preliminary work before you continue with this manual. First, you should read one of the excellent books on the GPSS language. Then, you should carefully work through the companion user manual, entitled *GPSS World Tutorial Manual*.

Afterward, you should read the first 4 chapters of this manual, using the remaining chapters as reference material. This course of study will place all the interactive power of GPSS World at your fingertips. It is best to restrict yourself to a subset of the GPSS language until you gain some actual model building experience. Most GPSS modelers are productive without mastering all the Blocks available in the language. However, you will be most effective by considering simulation methodology, and the GPSS language, to be a topic of continuing study.

You should become familiar with both the Online Help facilities and the online reference manual. Both are handy for quick references, and for providing answers to specific questions.

## **Acknowledgments**

Minuteman Software gratefully acknowledges the contributions by Drs. Averill Law and Stephen Vincent, of implementation of many of the random variate generation algorithms used in GPSS World's built-in probability distributions.

# Chapter 1. Introduction

Predicting the behavior of complex real world systems -- that's what GPSS World is all about.

Many expensive projects in the past have failed because the end result was not characterized accurately. From maximum capacity to cost of operation, it is essential for detailed knowledge of the behavior of the system under construction to be known as soon as possible. Although purely mathematical models are extremely valuable, and should be used where feasible, the complexity of most real world systems requires the use of computer simulation to get the necessary answers. That's where GPSS World comes in.

GPSS World is based on the seminal language of computer simulation, GPSS, which stands for General Purpose Simulation System. This language was developed primarily by Geoffrey Gordon at IBM around 1960, and has contributed important concepts to every commercial discrete event Computer Simulation Language developed ever since. GPSS World is a direct descendent of GPSS/PC, an early implementation of GPSS for personal computers. Since its introduction in 1984, GPSS/PC and its successors have saved thousands of users millions of dollars. Now, the Windows implementation of GPSS World extends these capabilities into an Internet aware environment.

GPSS World is designed to deliver answers quickly and reliably, with the least effort, achieving the highest reliability of results. Consistent with these objectives, visualization of running simulations is highly stylized and a default statistical treatment is built in. This approach means that animations are "free" requiring no additional effort to produce, but are not photo-realistic. GPSS World's forte is transparency, not photo-realism. Third party animation systems are available which can provide pictorial animations based on GPSS World simulations.

Transparency is valuable for three reasons. First, it is dangerous to rely on an opaque "Black-Box" simulation whose internal mechanisms cannot be observed. Not only can you not be sure it fits your situation, but it is difficult to be assured that it even works as intended. Second, successful simulations are valuable and have a surprisingly long lifetime. It is possible that new staff members will be required to become familiar with the internal workings of the simulation -- a near-impossible task unless provision has already been made for a high level of transparency. Third, one of the most effective but least mentioned benefits of computer simulation is the insight on system behavior achieved when an experienced simulation professional can see the internal dynamics at crucial times in the simulation.

GPSS World was designed to address these issues. Its visualizable nature allows the internal mechanisms of models to be revealed and captured. Its interactivity allows one to explore and manipulate simulations. Its built-in data analysis facility can calculate confidence intervals and an Analysis of Variance easily. And now, it can even create and run sophisticated screening and optimization experiments automatically, with relatively little effort from you.

Most systems can be modeled in any of several ways using GPSS World. Usually only a small subset of the features available need to be used. However, the greatest proficiency requires familiarity with all that GPSS World has to offer. This manual is the primary source of that information.

This chapter consists of 5 sections. The first is a brief overview of the most important aspects of GPSS World. The second explores a few basic concepts that are necessary for moving on to later chapters. The third section discusses the advanced architecture of GPSS World and what you must know to take advantage of it. The fourth section explores the modeling language of GPSS World, and the fifth section is for GPSS/PC users who want to take advantage of the Compatibility Mode of GPSS World. This last feature is available only in the Commercial Version of GPSS World.

Installation and operating procedures are discussed in Chapter 2.

## 1.1. Highlights

GPSS World is object oriented. Its inhabitants include Model Objects which are used to create Simulation Objects. Simulation Objects, in turn, are used to play out simulations and create Report Objects. Finally, Text Objects can be used as Include-files to support code sharing and a user source code library and they are often used as files which can be read from or written to by the simulation.

Simulation projects require several steps. They normally include model building and data collection, testing and verification, simulation, experimentation, and the analysis of results. GPSS World has a large number of capabilities addressing each of these steps. In GPSS World, you will create and modify a model using the full-screen text editor. If you prefer, you can insert GPSS Block Statements

using special Block Input Dialogs where you just fill in the blanks. The resulting Block Statements are placed sequentially at the insertion point in your Model Object. You then Create a Simulation Object by selecting **Command / Create Simulation** in the Main Window Menu. Thereafter, you have a powerful set of Commands at your disposal to control the running of the simulation. You can enter Commands interactively, or you can include them in the original model. During the testing and verification phase, a large number of window types are available for online viewing and to take snapshots of the simulation. Hot Keys and Point and Shoot breakpoint control makes verification and debugging easy. After the simulation is tested, you can use the Automatic Experiment Generators or you can design your own experiments. GPSS World will help you every step of the way.

GPSS World was designed to exploit your computing environment. The use of virtual memory allows your models to grow to a billion bytes. Preemptive multitasking and multithreading mean higher responsiveness and permit GPSS World to be doing many things at the same time. It also means that the simulation environment can utilize the computing capacity of Symmetric Multi-Processing architectures.

The structure of the modeling language has been simplified in GPSS World. A Model is now defined as a sequence of Model Statements. Each can be a GPSS Statement or a PLUS Procedure definition (possibly a PLUS Experiment definition). A Model Object is used to create the corresponding Simulation Object, which can then be activated. Any Model Statement can be sent to an existing Simulation Object. Such statements are called Interactive Statements. These concepts are discussed in more detail in the next section.

New in GPSS World are the GPSS Block creation dialogs. These are great in a teaching environment. Using the **Edit / Insert Block** menu command, you can open the Block Menu Window which has a button for each GPSS Block. Passing your mouse over each button is all you need to do to see a description of that Block. When you click on a button, a Dialog opens with all the details you need to create a GPSS Block. Just fill in the blanks. The Help button takes you to the appropriate place in the Reference Manual describing the block. When you click OK the resulting block is inserted in the Model Object. Don't worry, any syntax errors are reported immediately so you can correct them on the spot. For formatting purposes, the inserted GPSS Block Statements contain tabs. You can alter the tabstops to your liking in the Settings of the Model Object. In any event, since inserted Block Statements and PLUS Experiments are in the form of source code, you can always edit them before Creating the Simulation Object.

The simulation language within GPSS World has been extended by PLUS, the Programming Language Under Simulation. This simple but powerful programming language removes the restrictions that existed in the older GPSS implementations. Data within this environment are untyped, with conversions occurring automatically as needed. In addition, powerful function and probability distribution libraries are directly available to be used in PLUS Expressions. The Procedure Library supports string manipulations, numeric calculations and probability distributions. User written PLUS Procedures can be used in the same way. An INCLUDE Command can bring in existing User Procedure Libraries containing tested PLUS Procedures for use throughout the simulation.

User-defined PLUS Procedures and can be accessed anywhere in the Model. Expressions, defined in PLUS, can incorporate data elements and System Numeric Attributes. When parenthesized, PLUS Expressions can appear outside PLUS Procedures, in GPSS Statements. In fact, most arguments in the GPSS Statements can now be in the form of a parenthesized expression.

In Version 4 of GPSS World, the PLUS Language has been extended for definition of Experiments. This powerful feature permits programmable control and can even be based on simulation results. Thus, completely automatic operation is possible including the exploration of response surfaces. PLUS Experiments, which are invoked by the CONDUCT Command, can be used to control the running of simulations over a parameter space. Experiments can be created with minimal involvement on your part. GPSS World now includes two new powerful experiment generators that can be accessed through the Edit Menu of the Main Window. The Screening Experiment Generator will create a fractional factorial experiment under your direction, and insert its PLUS source code into your model. Similarly, the Optimizing experiment generator will insert a sophisticated response surface exploration experiment that searches for minimum or maximum yields. The CONDUCT Command necessary to initiate the experiments is by default loaded into a Function Key. To execute a Screening or Optimizing Experiment and analyze the results, all you have to do is press a button.

GPSS World has comprehensive discrete and continuous modeling capabilities. The tightly knit continuous modeling feature allows for easy transition between the continuous and the discrete phases. From the continuous phase thresholds can be established that trigger the creation of Transactions for the discrete phase. Conversely, the INTEGRATION Block and INTEGRATE Command, control the processing in the continuous phase.

Several new GPSS Blocks have been added to cover the control of integrations, Transaction rescheduling, changes to Assembly Sets, user-defined PLUS Blocks, and Data Streams. Data types now include integer and real numeric values, as well as strings. Each type is automatically coerced to the required form, as needed. The new data type "UNSPECIFIED" now is available to indicate the absence of valid data, such as that from a missing run of an experiment.

Even matrix structures have been improved. They can now incorporate up to 6 dimensions. The new multiway ANOVA Library Procedure analyzes data from a "Result Matrix", which is nothing more than a GPSS Matrix Entity where the yields of an experiment have been stored in a conventional way.

GPSS World is easy to operate. A full screen text editor can be used in any of the Text Windows. Even the Journal Window and Reports can be customized and annotated. You can use bookmarks as placeholders in any text window. Then since a single keystroke takes you to successive bookmarks, jumping from place to place in a large Model is a trivial matter. When you Create a Simulation, if any errors are detected, they are remembered so that you can correct them one at a time. When you go to each successive error, the cursor moves automatically to the location in the Model where the error was detected to help you locate each problem. After the Simulation is Created, several Hot Keys can be used in place of Interactive Commands. Not only that, using Model and Simulation Settings, you can load your own Commands into one or more function keys in order to get single keystroke response. Windows are easy to open. Generally, only a menu selection is needed. During debugging, the Debug Toolbar allows even breakpoint control to be accomplished in a Point and Shoot fashion.

It's easy to visualize running simulations. GPSS World can create stylized animations of any GPSS entity type with no more than a mouse click or two, from you. These windows are dynamic -- showing the changing state of running simulations. In several different ways, you can view the changing condition of any GPSS entity. In addition to the entity specific views, one or more Expression Windows can be opened to display the changing values of any number of your own Expressions. For visualization of any of the variables in your simulation, any number of scrollable PLOT Windows can be opened to show dynamics graphically. And finally, the Table Window opens a changing view of any frequency distribution you care to see.

GPSS World provides for a set of snapshots, as well. These are advanced features intended for professionals who need detailed knowledge of microstates of the simulation. Static snapshots can be taken of any Transaction, the Future Events Chain, the Current Events Chain, or the membership of the Numeric and Transaction Groups.

GPSS World is highly interactive. All Model Statements can be used interactively. When you send a Command, Block Statement, Procedure Definition, or Experiment Definition to an existing Simulation Object, the interaction is used to redefine or manipulate the state of the simulation. This level of control is useful while the behavior of the simulation is being verified. In essence, you can do almost anything, except Block insertion, to the simulation after it is created.

Simulations can communicate with the outside world. You can now use five new GPSS Blocks and/or PLUS Procedures to manipulate Data Streams. They are OPEN, CLOSE, READ, WRITE, and SEEK. The last of these, SEEK, provides for direct access into a database. Data Streams have many purposes. You can use them to access data files, to create Result Files and custom reports, to access internal data directly. There are now PLUS Library Routines with the same names that perform nearly the same functions. That means that complex I/O operations can now be handled inside User-created PLUS Procedures. GPSS World simulations can even communicate with other products by direct invocation. The PLUS Procedure Library now contains the **Call()**, **Call\_Integer()**, **Call\_String()**, and the **Call\_Real()** procedures which can invoke external functions existing on your system in executable files (i.e. EXE and DLL files). These features are described in Chapter 8.

The analysis of results is easy in GPSS World. It has facilities that support the capture and printing of graphical windows. The Journal Window records the activities associated with the Simulation Object. An automatic Report numbering system ensures the safe keeping of each Standard Report. The new ANOVA Library Procedure can perform a complete multiway Analysis of Variance when passed a Result Matrix. That's all there is to the generation of an ANOVA table and a set of Confidence Intervals. And now, the Automatic Experiment Generators can create sophisticated screening and optimizing experiments based on dialog-window input from you. The basic analyses of these experiments are done automatically and reported in the Journal Window.

A new Batch Mode of operation can be used to run simulations in the background. If you specify a Model File or a Simulation File on the DOS command line followed by the word "BATCH", GPSS World will run in a minimized window. It first opens the file and then passes it a Create Command (if a Model Object) or a CONTINUE Command (if a Simulation Object). In the former case, you would normally append the desired run control Commands in the Model itself. You can use the new EXIT 1 Command (or Exit(1) Library Procedure) to shut down the resulting Session by itself, automatically saving all the newly created or modified Objects. EXIT is discussed in Chapter 6.

GPSS World does inherit some characteristics from GPSS/PC. For example, normally it retains the use of the # character as the multiplication operator, reserving the \* character for GPSS Indirect Addressing. You can switch the function of these characters so that \* denotes multiplication, if you like, by changing a Model Setting. Although GPSS Statements, except Function Followers, must still fit on a single line of text, the maximum line length has been increased to 250 characters. PLUS Statements are not so restricted. They can span any number of lines.

## 1.2. GPSS World Concepts

GPSS World is populated by 4 kinds of objects. Model, Simulation, and Report Objects form the 3 basic types that are used in all GPSS World simulations. Typically, a Model is developed by editing the statements in a Model Object. Then, a Create Simulation Command is issued, thereby creating a Simulation Object structured according to the statements in the Model. Simulations are run by sending them GPSS Commands, or by including commands in the Model Object, itself. Normally, when a simulation completes, a Report Object is created automatically.

Overall control of multiple simulation runs can be handled within this framework by including a PLUS Experiment in the Model Object. (PLUS is the "Programming Language Under Simulation" that can be used to define procedures and experiments.) Then, a CONDUCT Command can be sent to the Simulation Object to control a series of simulation runs, and deal with the results.

The Text Object completes the 4 types of GPSS World Objects. These are named in INCLUDE Statements to aid the development of large simulations, and for a few other handy uses such as the creation of a source code library. In addition, Text Objects are often used by Simulation Objects, when OPEN, CLOSE, READ, WRITE, and SEEK operations are performed.

All GPSS World Objects can be saved at any time, in any state, and reopened in a later session. This includes Simulation Objects, which can be run any number of times from a saved state, thereby facilitating debugging and demonstrations. With some careful planning, Simulation Objects can be saved in the middle of complex Experiments, allowing you to complete the simulation runs at a later time.

All GPSS World Objects except Text Objects contain data in addition to the basic ASCII text. This is why you can use an external text editor to edit a Text Object, but not any of the others. Model, Simulation, and Report Objects contain Bookmarks for easy navigation, and a list of attribute values called Settings, which can be modified and are inherited by child objects. When a Simulation Object is Created it begins with a copy of the Settings of the Model that created it. The same goes for Report Objects which inherit their settings from the parent Simulation Object. In addition, Model Objects remember all the Translation errors encountered the last time you tried to Create a Simulation Object from it. That way you can make some corrections and save the rest for later.

### Model Objects

A Model Object contains primarily a sequence of Model Statements and a set of built-in values called Settings. In addition, a set of Bookmarks and a circular Result List of syntax errors are part of the Model Object. When a Simulation Object is Created by Translating the Statements in a Model, it inherits all the Settings from the parent Model Object. The **Command / Create Simulation** menu command in the Main Window is used to create a new Simulation Object.

#### Model Statements

A Model is the set of all statements contained in a Model Object and all its Include-files. A Model Statement can be either a GPSS Statement, or a PLUS Procedure definition. GPSS Statements, in turn, can be either Block Statements or Commands. Model Statements define a Simulation Object when it is first Translated, and they can be sent as interactions to an existing simulation.

Commands are either *Queued* or *Immediate*. Actually, all Commands except HALT and SHOW are Queued. Each Simulation Object has a special chain called a Command Queue that holds all Queued Commands received by the Simulation Object. Commands are taken off the Command Queue, one after the other, for execution.

The HALT Command is a special case. It is an Immediate Command, and therefore is not placed on the simulation's Command Queue. However, when a HALT is executed, all remaining Commands are removed from the Command Queue and discarded. This ensures that all action on behalf of the simulation ceases when a HALT is received. Normally, a HALTED simulation is resumed by sending it a CONTINUE Command.

#### Multiple Model Files

Usually, a Model is the set of Statements in a single Model Object, and optionally, any number of plain text files, called Text Objects. It is sometimes useful to place a sequence of one or more Statements, PLUS Procedures, and/or PLUS Experiments into one or more separate Text Objects in order to modularize the model.

Text Objects, which are plain text files, can be named in INCLUDE Statements in the Model Object. Doing so causes all Statements in the Text Object to be inserted in place of the INCLUDE Command when the Simulation Object is created. Text Objects may themselves may contain INCLUDE Statements, to a maximum nesting level of 5. INCLUDE Commands can be used interactively, as well

as within Models. They can even be loaded into Function Keys, thereby invoking a complex sequence with a single keystroke. This is discussed in detail in Chapter 2.

A Model is built from the Statements in exactly one Model Object and optionally any (nonnegative) number of Text Objects, which are mentioned in INCLUDE Statements encountered during the Translation of the Model. Only the Model Object can have Settings associated with it, which can be accessed through the Main Window's **Edit / Settings** menu command. These Settings are inherited by all Simulation Objects Created from that Model Object. To prevent the occurrence of multiple Settings, Model Objects may not be used in INCLUDE Statements, only Text Objects.

There are two kinds of numbering associated with files. Model and Include-files are numbered internally so they can be referenced by Status Messages coming from the simulation. The number is the 0-relative ordinal of the file in the order encountered by the Translator when the Simulation Object was created. For example, the primary Model Object is assigned Model File Number 0 and the Include-files are numbered as they are encountered starting with 1. This numbering is for identification in error messages only. A different numbering system is applied to newly created simulation and report files in order to keep the file names distinct. You can set the numbering of the filenames of child objects within the Settings of the parent object. For example, to change the numbering of simulation filenames, you should edit the Settings of the parent Model Object. Similarly, edit the Settings of the Simulation Object to set the numbering of its child Report Objects.

## Model Settings

Each Model Object carries with it a set of choices called Model Settings, or simply "Settings". These are accessible using the **Edit / Settings** menu command in the Main Window's menu. Settings are inherited by the Simulation Object, and also by the resulting Report Objects. Some settings can be changed even after the Simulation Object is created. The details are in Chapter 2.

## Simulation Objects

A Simulation Object is created by translating the statements in a Model Object. The **Command / Create Simulation** menu item is used for this purpose. After a Simulation Object is created successfully, commands are then used to advance the state of the simulation. These Commands can be part of the Model Object, or they can be sent to the existing Simulation Object as interactions.

## The Translator

The high performance Translator is that part of the GPSS World program which Creates the Simulation Objects. All Model Statements are Translated before they are incorporated into the simulation. Similarly, interactions are Translated in global scope before they are sent to an existing Simulation Object.

When errors occur which prevent a Simulation Object from being Created, a list of messages is placed in the Model Object. Then, you can move to the next error message by a mouse click in the **Search / Next Error** menu item or using accelerator keys. Each error is described in the Status Line at the bottom of the Main Window and the mouse cursor jumps to the location in the Model text where you can correct the error. Correcting and retranslating is quick and easy. There is a special command, **Command / Retranslate**, for retranslating the Model into an existing Simulation Object.

## Report Objects

One of the strongest features of GPSS has always been its Standard Reports. With essentially no effort from the model developer, a simulation run automatically reports on all the GPSS Entities defined in the model when the simulation's Termination Count reaches zero. There are a variety of ways of customizing these reports by Editing the Settings in the Simulation Object.

## Text Objects

A Text Object is simply a way to represent a plain text file in GPSS World. Their primary use is as the target of INCLUDE Commands. This allows a shared set of Model Statements to be accessed easily from a Model Object. You can even load an INCLUDE Statement into a function key. In that way, a complex Command list and/or PLUS Procedure in a text file can be sent to a Simulation with a single keystroke.

Text Objects turn up in other places, as well. Data Streams allow a Simulation to read and write data from text files, and to create Result Files for later analysis. As you might expect, these files are all represented as Text Objects.

## 1.3. Architecture

At the interface level, GPSS World is an implementation of the Document/View architecture common to Windows™ applications. Objects can be opened with multiple views, modified, and saved to persistent memory according to highly intuitive action sequences on the part of the user. The familiar main window menu and the disabling of inappropriate menu commands guides the user to his/her objective with minimal distractions. Internally, GPSS World has been designed to take advantage of close interactivity even in a multitasking virtual memory environment. This is discussed next.

### 1.3.1. Multitasking

The Multithreaded-design of GPSS World allows it to run multiple simulations and experiments simultaneously. Not only is screen update, user input, disk Input/Output, printing, and simulation done concurrently, but any number of simulations can be run at the same time, as well.

### 1.3.2. Virtual Memory

Simulations are not directly limited to the size of the physical random access memory (RAM) where the Simulation Object is run. Using virtual memory, simulations can be up a gigabyte in size. No other direct entity limits are imposed, assuming adequate swap space is provided. For optimum performance, however, sufficient real memory must be available. All entity allocation and management is invisible to the user. Entities are created automatically, unless additional information is required. REALLOCATE statements are not used.

### 1.3.3. Interactivity

GPSS World maintains a high level of interactivity with your simulations. You can send any Model Statement to an existing Simulation Object, using the Command item on the main menu of the Model Window, Accelerator Keys, or by using Model Settings to load function keys with your own customized Commands. You can use the Custom Command Dialog for statements not on the drop down menu, and using an INCLUDE command, you can send an interaction of any complexity to a simulation. Even INCLUDE Commands can be loaded into Function Keys, thereby supporting iterations of any complexity invoked by a single keystroke.

Block Statements, when used interactively, cause the Active Transaction to attempt to enter a temporary block, which is then discarded. This mode is called Manual Simulation Mode and it is useful during Experiments, providing all the functions available inside the simulation.

PLUS Procedure and Experiment Statements case a user-defined procedure or experiment to be registered in the simulation's Procedure Library. Any existing Procedure item of the same name and type is replaced.

Commands, except for HALT and SHOW, when received by an existing Simulation Object, are placed on the simulation's Command Queue to be executed in turn. HALT and SHOW are Immediate Commands, and are executed immediately when first received by the simulation. In addition, HALT removes all waiting Commands from the Command Queue. CONDUCT Commands initiate a PLUS Experiment that takes over control of the Simulation Object. Experiments can base decisions on results and invoke a statistical analysis, however interactivity is reduced during an Experiment.

### 1.3.4. Visualization

GPSS World excels in the visualization of running simulations. Consistent with the generality of the GPSS language, twenty different windows are available for observation of and interaction with your simulations. No additional effort beyond operating the windows is necessary to obtain, save, and print a visual representation of the state of the simulation. Whereas, Simulation Snapshots a one-time static picture of a quickly changing entity, Simulation Windows are updated online, dynamically changing to keep current with the changing state of the running simulation. Although the dynamic windows show the changing state of entities, they also cause simulations to run more slowly.

Many windows have a non-detailed view and a detailed view which show alternate sets of information. Others allow you to display a running numerical simulation clock at the bottom of the Main Window. You make the choices using the View item on the Main Window's menu bar.

The details on the appearance and operation of each of the windows is in Chapter 5.

## Snapshots

Snapshots are available for:

- ◆ Current Events Chain
- ◆ Future Events Chain
- ◆ Individual Transactions
- ◆ Numeric Groups
- ◆ Transaction Groups

## Dynamic Windows

Any number of online simulation windows can be opened on the following entity types:

- ◆ Blocks
- ◆ Facilities
- ◆ Logicswitches
- ◆ Matrix
- ◆ Queues
- ◆ Savevalues
- ◆ Storages
- ◆ Table

The Table Window presents a dynamic histogram that is extremely useful for viewing the accumulation of values, looking for outliers, and observing the convergence to a parent probability distribution.

In addition, the Plot Window and the Expression Window can be used to follow values with meaning to your simulation. Any number of multicolored on-line plots can be viewed, saved, or printed by opening one or more Plot Windows. Each window can track the value of up to 8 general Expressions, including integration variables. Plots can be scrolled vertically and horizontally, and can be scaled in either direction, as well.

The Expression Window can be an extremely powerful microscope on your simulation. Using it, you can view the changing values of any number of PLUS Expressions, of your own creation. As with all Simulation Windows, you can open Expression Windows at any point in the simulation.

The Blocks Window shows the entry of Transactions into blocks. It allows you to set and remove breakpoints using the mouse or the keyboard and to visually step through the simulation.

Most Interactive Windows have a Debug Toolbar which allows a "Point and Shoot" debugging session using only the mouse. The Debug Toolbar is discussed more thoroughly in Chapter 5.

## Simulation Clocks

Each Simulation Object has an internal toggle switch that causes it to print the changing value of its System Clock in the rightmost pane of the Status Line at the bottom of the Main Window. This provides an easy verification that the Simulation is progressing normally. When you click on any View of a Simulation Object, it will display its clock if you have set its "Show Clock" toggle switch. This is very useful for monitoring the status of Experiments.

## Animation

A variety of animation possibilities exist in GPSS World. The level of realism varies from abstract visualization requiring no effort, to highly realistic dynamic representations involving complex user-drawn elements.

### Abstract Animation

A comprehensive stylized animation capability is built into GPSS World. This facility is provided by a large set of Simulation Windows which show the GPSS Entities of the simulation dynamically, as they change state during the simulation. No additional effort is required to view such animations, other than merely opening the window. The graphical images may be saved for inclusion into reports, and/or printed, as well.

### Post-processing Animation

GPSS World includes an external interface that can support simulation animation packages that are driven by trace data. Photorealistic animations can be developed in this manner. To use this interface, you construct an output Data Stream obeying the formatting and content rules of the third party animation package. Data Streams are discussed in the next section..

### Online Animation

GPSS World includes set of dynamic call PLUS Procedures that allow you to invoke the functions in third party executable files. This can provide an online link to a third party animation package. The details are in Chapter 8.

## 1.4. The Modeling Language

GPSS World contains an implementation of GPSS, the General Purpose Simulation System, enhanced by an embedded programming language PLUS, Programming Language Under Simulation.

The GPSS variant includes over 50 Block Statements and 25 Commands. It includes over 35 System Numeric Attributes, which provide instant state variables, accessible anywhere in the model.

PLUS is a small but powerful procedural programming language created out of only 12 statement types. Much of its power comes from a large procedure library containing mathematical and string manipulation functions, and a strong set of probability distributions.

In GPSS World, a Model is defined to be a sequence of Model Statements. These are either GPSS Statements, PLUS Procedure Statements, or PLUS Experiment Statements. GPSS Statements, in turn, are either Block Statements or Commands. Except for Function Follower Statements, all GPSS Statements must reside on a single text line of up to 250 characters. Any Model Statement can be part of the model in a Model File, or can be sent interactively to modify an existing simulation.

The inclusion of Expressions in PLUS Statements and Parenthesized Expressions in GPSS Statements is a strong enhancement to the modeling power and ease of use of the language. The use of the [\*] character to denote multiplication has been retained as the default. This clearly distinguishes that arithmetic operation from the GPSS indirect addressing operation which is denoted by the [\*] character. However, you can now switch the role of these operators if you prefer. This is done on Page 1 of the Settings Notebook of the Model Object.

Commands are discussed in Chapter 6, Block Statements in Chapter 7, and the PLUS Language, including the Procedure Library, in Chapter 8.

### 1.4.1. What's New

Many changes have been made to the modeling language. The distinction of Control Statements has been removed. All GPSS Statements that are not Block Statements are now simply called Commands. Line numbers are ignored in GPSS World. If used, they must begin in the first column of the text line.

Automatic truncations have been removed from GPSS World. You must now explicitly use the INT( ) procedure, or some other method, if you wish intermediate numeric results to be truncated. This is true in all Expressions, even in VARIABLE and BVARIABLE Statements. Similarly, the data type returned by a System Numeric Attribute can now be integer, real, or string, depending on the SNA. Even those SNAs returning parts per thousand do so as a real number of double precision between 0

and 1000, inclusively. The old truncations and integer SNAs can be enforced by running a Commercial Version of GPSS World in GPSS/PC Compatibility Mode, which is discussed below.

PLUS Statements can span any number of text lines. Although GPSS Statements, except for Function Followers, must reside on a single text line, the maximum length is now 250 characters.

Parenthesized mathematical expressions can now be used nearly anywhere you can use an SNA. Expressions are an integral part of the PLUS Language, but they are also used to expand the power of operands in GPSS Statements. The permissible forms of each operand are given in the statement descriptions in Chapters 6 and 7. Many new statement types have been added, as well. These are discussed next.

Several new features are available in both Student and Commercial Version. Here's a quick overview:

- ◆ Automatic Optimizing Experiment Generator
- ◆ Automatic Screening Experiment Generator
- ◆ Snapshots available in Student Version
- ◆ Multiway 'ANOVA' Library Procedure
- ◆ Variable Argument count for 'PolyCatenate' Library Procedure
- ◆ 'Call' Library Procedures to invoke external functions
- ◆ PLUS Trace Setting
- ◆ BATCH Mode Sessions
- ◆ EXIT Command with File Save Modes
- ◆ Block Creation Dialogs
- ◆ Adjustable Model Tabstops
- ◆ PLUS Data Stream Procedures

## New GPSS Statements

Several new Blocks have been added.

### New GPSS Blocks

- ◆ **ADOPT** - Change Assembly Set.
- ◆ **DISPLACE** - Change the Next Sequential Block of a given Transaction. Save FEC residual time. Can be used to reschedule events.
- ◆ **PLUS** - Evaluate PLUS Expression and save result in Parameter.
- ◆ **INTEGRATION** - Turn the integration of a User Variable ON or OFF.
- ◆ **OPEN** - Initialize a Data Stream for operation.
- ◆ **CLOSE** - End the Data Stream operation.
- ◆ **READ** - Bring the next line of data from a Data Stream into a parameter of the Current Transaction.
- ◆ **WRITE** - Send a line representing the value of a Transaction Parameter to the next slot in a Data Stream.
- ◆ **SEEK** - Change the Current Line Position in a Data Stream.

### New Commands

Statements have been added to control the integration of continuous state variables, to provide for multiple Model Files and Procedure Libraries, and to invoke PLUS Experiments..

- ◆ **CONDUCT** -Begin a previously registered PLUS Experiment.
- ◆ **EXIT** - Terminate Session, optionally saving objects.
- ◆ **INTEGRATE** - Automatically integrate an Ordinary Differential Equation, with optional Transaction generation trigger values.
- ◆ **INCLUDE** - Insert a Text Object for Translation.

## New SNA

The new System Numeric Attribute **A1** returns the Assembly Set of the Active Transaction. This is useful in conjunction with the new ADOPT Block which lets the Active Transaction change its Assembly Set.

### 1.4.2. Language Elements

GPSS World provides for a range of language elements for convenience in the development of complex simulations.

#### Polymorphic Data Types

Variables can now take on values with one of four types. User controlled variables, such as Savevalues, Matrix Elements, Transaction Parameters, and User Variables can take on integer, real, string, or UNSPECIFIED values. Clock values are integer or real, in double precision.

Conversion between types is automatic. Procedures that require a specific data type as an argument will coerce the argument to the appropriate type. If you pass a string to a Procedure that requires a numeric argument, the numeric equivalent of the string is used. Similarly, if you attempt to WRITE a numeric value, it is converted to a line of text automatically.

String values have many uses. They can be used in Data Streams, to format reports and Result Files, and internally for direct access of data. String constants are denoted by enclosure of the string in a pair of double quotes. You can use two double quotes together to represent a single double quote internal to the string. For example, you would use a total of 6 double quote characters to represent a string within a string. The inner string would be "sandwiched" by two pairs of double quotes, and the overall string would be "sandwiched" by a double quote character on each end. Procedure Library contains many string procedures that can be used to create and manipulate strings.

You can now give the Data Type 'UNSPECIFIED' to Savevalue Entities, and Matrix Entities. The INITIAL Command allows you to assign this type to Savevalues, Matrix elements, and even whole Matrices at once. If an UNSPECIFIED datum is used in an operation that requires a value, an Error Stop will occur. The ANOVA Library Procedure has been upgraded to recognize UNSPECIFIED elements in a Result Matrix as unavailable experimental run results.

#### Initializing Data Structures

Variables and Matrix elements may be given initial values in several ways.

- ◆ The INITIAL Command can be used to give values to Logicswitch, Savevalue, and Matrix entities. PLUS Assignment Statements are used to assign values to higher dimension matrices.
- ◆ SAVEVALUE Block Statements assign values to Savevalue Entities.
- ◆ MSAVEVALUE Block Statements assign values to Matrix Entities.
- ◆ LOGIC Block Statements assign values to Logicswitch Entities.
- ◆ READ and ASSIGN Block Statements assign values to Transaction Parameters.
- ◆ PLUS Assignment Statements can be used to assign values to User Variables and Matrices.
- ◆ The CLEAR Command normally resets the values of Savevalues, Logicswitches, and Matrices to integer zero, This action can be suppressed by specifying OFF in the B Operand of CLEAR.

Normally, you would use the OFF option in CLEAR Statements to be used in Experiments. Otherwise, the Result Matrix would not be preserved.

It is often useful to isolate initialization statements in a Text Object. Then, only an INCLUDE Command need be asserted to perform the complete initialization. This can be done interactively, as well.

PLUS Experiments can control initializations to any level of detail. Any initialization command can be invoked within an Experiment by use of the DoCommand Library Procedure.

## Multidimensional Matrices

Matrices can be defined with up to 6 dimensions. Dynamic Matrix Windows view any cross-section of a Matrix, and there may be any number of Matrix Windows. Temporary matrices can be created for use during procedure invocation. PLUS assignment statements are used to initialize matrices of dimension 3, or more.

## Expressions

GPSS World supports the widespread use of expressions. They can appear in PLUS Procedures or, when parenthesized, in GPSS Statements. This means that a powerful level of computation can be achieved just in the Operands of Blocks and Commands. Expressions can do simple computations, invoke procedures that perform math or string operations, sample probability distributions, or execute user-defined algorithms, including file I/O.

### 1.4.3. Embedded Programming Language

The Programming Language Under Simulation, PLUS™, is a simple, but powerful, embedded programming language that fills the detailed computational needs of users who require a fine level of control of data structures, computational algorithms, and ad hoc Block processing.

PLUS Procedure Statements can be used to define Procedures as part of the original model, or they can be sent to an existing simulation. This applies to PLUS Experiment Statements, as well.

A new GPSS Block, PLUS, allows users to invoke a PLUS Procedure as a Block routine. In effect, Users can create their own blocks, in as complex a manner as desired.

PLUS Procedures are easily created, can reside anywhere in the model, and can be INCLUDED in a Procedure Library. They can also be redefined at any time, even interactively.

PLUS Statement types are discussed in Chapter 8. They include:

- ◆ Assignment◆ BEGIN
- ◆ DO...WHILE
- ◆ END
- ◆ EXPERIMENT
- ◆ GOTO
- ◆ IF ...THEN...ELSE◆ PROCEDURE
- ◆ Procedure Call
- ◆ RETURN
- ◆ TEMPORARY

### 1.4.4. Procedure Libraries

GPSS World has a wide range of functions that can be accessed as procedure calls.

## String Procedure Library

To facilitate manipulation of the new string data types, a built-in String Procedure library is available. It includes:

- ◆ **Align** - Return a copy of one string placed in another, right justified.
- ◆ **Catenate** - Return a copy of two strings combined into one.
- ◆ **Copies** - Create a string from many copies of a string.
- ◆ **Datatype** - Return a string denoting the data type of the argument.
- ◆ **Find** - Return the offset of one string in another.
- ◆ **Left** - Return a copy of a substring starting on the left.
- ◆ **Length** - Return the count of characters in a string.
- ◆ **Lowercase** - Return the lowercase representation of a string.
- ◆ **Place** - Place one string in another. Left justify.
- ◆ **PolyCatenate** - Return a copy of one or more strings combined into one.
- ◆ **Right** - Return a copy of a substring starting on the right.
- ◆ **String** - Convert a data item to its string equivalent.
- ◆ **StringCompare** - Return an integer result if string comparison.
- ◆ **Substring** - Return a copy of a substring of the string argument.
- ◆ **Trim** - Remove leading and trailing white space.
- ◆ **Uppercase** - Return the uppercase equivalent of a string.
- ◆ **Value** - Return the numeric equivalent of a string.
- ◆ **Word** - Return a copy of one of the words in a string.

## Math Procedure Library

Common Math library routines include

- ◆ **ABS** - Absolute Value
- ◆ **ATN** - Arctangent
- ◆ **COS** - Cosine
- ◆ **INT** - Truncate
- ◆ **EXP** - Power of e
- ◆ **LOG** - Natural Logarithm
- ◆ **SIN** - Sine
- ◆ **SQR** - Square Root
- ◆ **TAN** - Tangent

## New Transaction Query Procedures

Transaction state query procedures now include:

- ◆ **QueryXNExist** - determine the existence of a Transaction.
- ◆ **QueryXNParameter** - retrieve the value of a Transaction Parameter.
- ◆ **QueryXNAssemblySet** - retrieve the Assembly Set of a Transaction.
- ◆ **QueryXNPriority** - retrieve the priority of a Transaction.
- ◆ **QueryXNM1** - retrieve the mark time of a Transaction.

## New Utility Procedures

New utility Procedures have been implemented for the support of Experiments:

- ◆ **DoCommand** - Translate a Command String in global context and send it to a Simulation Object
- ◆ **ANOVA** - Perform Analysis of Variance. The new multiway ANOVA Procedure now can handle replicates and limited factor interactions.
- ◆ **Effects** - Calculate Effects from a generated screening experiment.
- ◆ **Exit** - Terminate the GPSS World Session saving all objects, no objects, or inquiring User for each.

A new utility Procedure has been implemented to communicate with programs in external executable modules:

- ◆ **Call** - Invoke a function in an executable module. Pass no arguments. Optional return value is an integer.
- ◆ **Call\_Integer** - Invoke a function in an executable module. Pass an integer argument. Optional return value is an integer.
- ◆ **Call\_Real** - Invoke a function in an executable module. Pass a real argument. Optional return value is an integer.
- ◆ **Call\_String** - Invoke a function in an executable module. Pass a string argument. Optional return value is an integer.

Over 20 built in probability distributions are available:

- |                           |                            |
|---------------------------|----------------------------|
| ◆ <b>Beta</b>             | ◆ <b>LogLaplace</b>        |
| ◆ <b>Binomial</b>         | ◆ <b>LogLogistic</b>       |
| ◆ <b>Discrete Uniform</b> | ◆ <b>LogNormal</b>         |
| ◆ <b>Exponential</b>      | ◆ <b>Negative Binomial</b> |
| ◆ <b>Extreme Value A</b>  | ◆ <b>Normal</b>            |
| ◆ <b>Extreme Value B</b>  | ◆ <b>Pareto</b>            |
| ◆ <b>Gamma</b>            | ◆ <b>Pearson Type V</b>    |
| ◆ <b>Geometric</b>        | ◆ <b>Pearson Type VI</b>   |
| ◆ <b>Inverse Gaussian</b> | ◆ <b>Poisson</b>           |
| ◆ <b>Inverse Weibull</b>  | ◆ <b>Triangular</b>        |
| ◆ <b>Laplace</b>          | ◆ <b>Uniform</b>           |
| ◆ <b>Logistic</b>         | ◆ <b>Weibull</b>           |

### 1.4.5. High Performance Translator

GPSS World features a new high performance model Translator that is hundreds of times faster than its predecessor. The largest models are Translated in no more than a few seconds. Any errors that are detected are placed in a circular error message queue so that they may be corrected quickly using the full screen editor in the Text View of the Model Object. Error correction is aided by automatic placement of the cursor next to the offending lexical element in the model. The error list is easily traversed by mouse or keystroke, and is saved with the Model Object.

### 1.4.6. Block Creation Dialog

The creation of GPSS Blocks in GPSS World can optionally be guided by "fill-in-the-blanks" dialogs. You access these through the **Edit / Insert Block** menu item which opens the GPSS Block Menu Window. You can then open a Creation Dialog for any GPSS Block by clicking on the appropriate button. When you click the OK button after filling in the information needed by the Block, the resulting statement is checked for syntax errors and, if correct, is inserted into the Model just after the line containing the flashing insertion point.

### 1.4.7. Automatic Experiment Generators

GPSS World now has the ability to generate experiments written in the PLUS language and insert them into your Model Object. All you have to do is to fill in the blanks in a few dialog windows, accessible through the Main Window's **Edit** Command. You can create either screening or optimizing experiments automatically. Optionally, a CONDUCT Command will be loaded into a function key so that all you must do to run the experiment is to Create the Simulation Object and press the function key. All this, and more, is discussed in Chapter 13, and in Lessons 19 and 20 of the *GPSS World Tutorial Manual*.

## 1.5. Compatibility

The following section is an extensive comparison between GPSS World, and its predecessor, GPSS/PC. You should skip this section if you have no experience with GPSS/PC.

GPSS World is compatible with GPSS/PC, and normally yields results that are statistically indistinguishable from it. This level of compatibility is available by simply correcting a few differences, and running the simulations.

In addition, a higher level of compatibility is available called GPSS/PC Compatibility Mode. In most cases, you can achieve precise duplication of results. However, GPSS World utilizes a new run time library. Its floating point round off differs slightly from that used in GPSS/PC. Even so, most GPSS/PC models can be modified slightly to achieve identical results when run under the Commercial Version of GPSS World in GPSS/PC Compatibility Mode. The procedure you should follow is detailed below, after the discussion of differences between GPSS World and GPSS/PC.

### 1.5.1. Differences from GPSS/PC

GPSS World has many differences from GPSS/PC. GPSS World is based on the idea that a textual Model Object is built, and/or modified, and then Translated in order to create a Simulation Object. This differs from the GPSS/PC notions of Program File and Simulation.

In GPSS World, the model loading operation of GPSS/PC has been completely replaced. Instead of scanning each line, one at a time, GPSS World has a full-screen editor and a Create Simulation menu Command. Any errors detected during the Translation are saved in an error message queue, so they may be easily found and corrected. The speed of the Translator assures that the detection/correction of errors can proceed quickly.

GPSS World does not perform Keystroke Error Prevention, as did GPSS/PC. Instead, GPSS World uses the model Translator for creating the simulation. This improves the model load-time by a factor of hundreds. However, it changes the error detection mechanism. Now, errors are detected during Translation, and are corrected by Selecting "Next Error" from the Search submenu of the Main

Window. The cursor then automatically moves to the error, and an error message is written in the status line at the bottom of the Main Window.

Perhaps the next most noticeable change from GPSS/PC is that line numbers are no longer necessary, and in fact, they are ignored by GPSS World. This means that the positioning of Blocks is no longer specified by the line number of the Block, only by its relative position in the Model File(s) given to the Translator. Since the new INCLUDE Statement allows Model Objects to contain other plain text files, it is the sequence of Block Statements read by the Translator that dictates the position of Blocks in the simulation. Although Blocks can no longer be inserted during a simulation, it turns out that the Translation time is so fast, for most purposes it will be as easy to retranslate the model when a change to the Block structure is desired. Even so, the high level of interactivity of "Manual Simulation" is retained. This means that any Model Statement can be used during a simulation as an Interactive Command.

The level of interactivity of GPSS World has been maintained. Any Model Statement can be passed to an existing simulation for execution. In fact, PLUS Procedures can be defined, or even redefined, on the fly. In the case of GPSS Commands, this usually redefines an entity, or controls the running of the simulation. In the case of a Block Statement, as in GPSS/PC, a "Manual Simulation" Block entry is attempted by the Active Transaction. All this is done using the Commands submenu, after the model has been Translated.

Automatic truncations have been removed from GPSS World. You must now explicitly use the INT( ) procedure, of some other method, if you wish intermediate numeric results to be truncated. This is true in all Expressions, even in VARIABLE and BVARIABLE Statements. Similarly, the data type returned by a System Numeric Attribute can now be either integer, real, or string, depending on the SNA. Even those SNAs returning parts per thousand do so as a real number of double precision between 0 and 1000, inclusively. The old truncations and integer SNAs can be enforced by running in GPSS/PC Compatibility Mode.

The multitasking architecture of GPSS World has led to other changes. To begin with, messages are now sent to represent Commands or status changes. Therefore, online window update is done via a queue of messages received from the simulation. Similarly, most Commands received by the Simulation Object are placed on the simulation's Command Queue before being executed one after the other. There are two exceptions: HALT and SHOW, which are done immediately when received. In addition, HALT deletes all remaining Commands on the queue.

The remaining changes will be listed here and discussed more fully, elsewhere in this manual.

- ◆ PLUS Statements can span any number of text lines. Although GPSS Statements, except for Function Followers, must reside on a single text line, the maximum length is now 250 characters.
- ◆ Parenthesized mathematical expressions can now be used nearly anywhere you can use an SNA.
- ◆ System Numeric Attributes now may return integer, real, or string values depending on the SNA. In GPSS/PC Compatibility Mode, SNAs return only integers, except as Function Modifiers.
- ◆ A new System Numeric Attribute, A1, has been created to return the Assembly Set of the Active Transaction.
- ◆ The HELP Block has been dropped. It is replaced by the PLUS Block, which supports the inclusion of complete Procedures in a model, and by the Data Stream Blocks that support communication with external files and programs.
- ◆ The MICROWINDOW Control Statement has been replaced by the Expression Window, which is opened from the Window / Simulation Window submenu of the Main Window Menu.
- ◆ The PLOT Control Statement has been replaced by the Plot Window which is opened from the Window / Simulation Window submenu of the Main Window Menu.
- ◆ The END Control Statement has been replaced by EXIT, which can terminate a Session. END is now a keyword in the PLUS Language.
- ◆ The ANOVA Control Statement has been replaced by the ANOVA Library Procedure.
- ◆ The EVENTS Control Statement has been replaced by the FEC and CEC Snapshot Windows which is opened from the Window / Simulation Snapshot submenu of the Main Window Menu.

- ◆ The GROUPS Control Statement has been replaced by the Numeric and Transaction Groups Snapshot Windows which is opened from the Window / Simulation Window submenu of the Main Window Menu.
- ◆ The RESULT Control Statement has been replaced by the Stream IO Blocks which can automatically write to a Result File.
- ◆ The WINDOW Control Statement has been replaced by the Window submenu of the Model Window Menu.
- ◆ The USERCHAINS Control Statement has been replaced by the Userchain Snapshot Window which is opened from the Window / Simulation Snapshot submenu of the Main Window Menu.
- ◆ The ANITRACE Control Statement has been dropped. Animation in GPSS World is provided by Data Steams and third party postprocessors.
- ◆ The MOVE Block has been dropped. Animation in GPSS World is provided by Data Steams and third party postprocessors..
- ◆ The Positions Window and the POSITION.GPS file have been dropped. Animation in GPSS World is provided by Data Steams and third party animation packages.
- ◆ The @ file inclusion character has been replaced by INCLUDE, which takes a double quoted filespec as an operand.
- ◆ The Z1 System Numeric Attribute no longer represents the total amount of physical memory available. It now returns the value of the maximum amount of memory that can be allocated, as returned by the operating system.
- ◆ The following GPSS/PC source management functions have been replaced by the Graphical User Interface with full-screen editor and filing options: DELETE, DISPLAY, DOS, EDIT, RENUMBER, and SAVE.
- ◆ Integer, Real, or String Data types are supported in Savevalues, Matrices, Named Values, and Transaction Parameters. Clock values may be integer or real. The unlimited precision integer arithmetic of GPSS/PC has been replaced.
- ◆ REPORT Commands always operate in NOW mode. Operand A is no longer used, and must be *Null*.
- ◆ A library of PLUS Procedures is now available for use in PLUS Expressions. Procedures for manipulating String data types and over 20 built-in probability distributions are supported.
- ◆ GPSS Matrix entities can now have up to 6 dimensions. Any 2 D Cross Section can be viewed dynamically in a Matrix Window.
- ◆ Operator precedence has changed. The new precedence is discussed in Chapter 3. If there is any question, you should fully parenthesize expressions from GPSS/PC Program Files.

### 1.5.2. Modifying Old GPSS/PC Programs

This section is for GPSS/PC users who wish to migrate existing GPSS/PC models to the GPSS World environment. It contains a list of changes you should make to your GPSS/PC Program Files. This will provide you with results that are not significantly different from those obtained from GPSS/PC simulations. You do not need GPSS/PC Compatibility Mode to achieve this level of correspondence.

However, if you want to obtain precise duplication of results, you must perform the procedure in the next section, as well.

There are many features provided to ensure an easy move to the new system. All GPSS/PC users should note the small number of enhancements that may require changes to your existing Model Files. The other end of the spectrum is those users who wish to precisely duplicate the GPSS/PC results in order to establish the veracity of the model when running under GPSS World. After that, it's an easy matter to switch over to the new mode of operation, including the floating point clock, and the myriad of other features.

Line numbers are now ignored. Do not use old Program Files that replaced GPSS/PC Blocks by line number. If you use them as Include-files they will add to the size of the simulation, if you use them as interactive Commands they will be executed one at a time as Manual Simulation Statements.

You should make the following changes to your Program.

1. If you used Keystroke Error Prevention of GPSS/PC to finish keywords automatically, you must type in the remaining characters before Translating the model under GPSS World.
2. Replace @ with INCLUDE.
3. Filespecs must be in double quotes in INCLUDE.
4. Remove all REPORT Commands.
5. Remove END Commands and any labels, such as BEGIN, COUNT, NORMAL, etc. that now clash with GPSS World keywords.
6. Replace HELP Blocks with PLUS and/or Data Stream Blocks.
7. Remove MOVE Blocks. Positions Windows are no longer supported.
8. Fully parenthesize all Expressions in VARIABLE, FVARIABLE, and BVARIABLE Statements.

It is probably safest to remove all the old run commands from the old GPSS/PC Program File until you have tested your modified model. Do not use line number replacement of Blocks in the Savable Program. This no longer works.

### 1.5.3. Strict Duplication of Results

Most GPSS/PC Program Files can be made to yield precisely the same results when run under GPSS World. However, because of differences in the rounding of real numbers in floating point calculations, you may need to make a few additional modifications to the GPSS/PC Program File, in addition to the ones in the previous section. If you only need results that will be statistically indistinguishable from those obtained from GPSS/PC, you do not need to do the following things using the Commercial Version of GPSS World.

1. This first step is to turn on GPSS/PC Compatibility Mode in the Model Settings. This causes GPSS World to use an integer clock and to truncate results, just as GPSS/PC did.

CHOOSE **File / Open** to open the Model Object

CHOOSE **Edit / Settings**

then on the first page check the checkbox labeled GPSS/PC Compatibility.

2. Now set the Random Number Streams of GENERATE, ADVANCE, and TRANSFER to correspond to the SETTINGS.GPS File you used with GPSS/PC. Next, set the Random Number Stream of Time Ties to 1.

CHOOSE **Edit / Settings**

then select the **Random Numbers** page. Then set the entry fields of the Random Number Streams for GENERATE, ADVANCE, TRANSFER, and Time Ties.

3. Make sure all seeds in any RMULT Commands are less than 100,000,000.
4. Remove all dependencies on an identifier being given a default system value. For example, RN\$IDENTIFIER seeds a random number stream with a distinct system default value. This entity specifier would need to be changed.
5. Do not call a Random Function from a Random Function. Replace any such statements.
6. Replace all fractions, occurring in TRANSFER Blocks, with parts-per-thousand integers.

7. Make sure Random GPSS Function Entities do not allow any Function Follower pair to be associated with a probability of  $1/n$ , where  $n$  is the number of pairs. For example, if a Function is defined by 4 points, do not allow any probability to be .25. You can do this by adding additional points until  $1/n$  is a repeating fraction, or is not associated with any of the Function Follower Pairs. FN\$SNORM and FN\$XPDIS do not have to be changed.

A special DOS program, named PCAID.EXE, has been provided to help you modify your GPSS/PC Program File. It will correct line numbers and remind you if you have GPSS FUNCTION Statements that need to be examined. You can start this program in a DOS Window.

In either case, keep in mind that Model Objects are not simple text files. They contain Settings, Bookmarks, and Result Lists that cannot be read by external word processing programs. For this reason, when you have finished modifying your GPSS/PC Program File, you should use a word processing program to copy it to the Windows clipboard. Then, in GPSS World, open a new Model Object

Select **File / New**

and paste the text into place using

Select **Edit / Paste**

This process creates a valid GPSS World Model Object based on your original model. You should now be able to get corresponding results for simulations run under GPSS/PC and GPSS World.

[\[Table of Contents\]](#)

# Chapter 2 - Operating GPSS World

This chapter contains a discussion of the organization of the features available in GPSS World and how to operate them. We first go through the installation procedure and then discuss the normal operation of an installed GPSS World package. Specific windows are discussed only briefly. An additional treatment of each window type can be found in Chapter 5. Explanations of error messages are in Chapter 14.

## 2.1. Installation

Normal installation of GPSS World places a copy of the executable files, the help file, and the GPSS Sample Models into folders, sometimes called directories, on your computer. Online HTML Documentation, such as this Reference Manual and the *GPSS World Tutorial Manual* are normally included in the installation of the Commercial Version of GPSS World. For Student Version installations the HTML manuals must be downloaded and/or copied separately. Both manuals are available in hardcopy form. All installations include the Reference Manual in the form of online help.

### 2.1.1. Overview

Your installation will begin when you double click on the Microsoft Windows Installer (msi) file downloaded from MinutemanSoftware.com or when you insert, or run the startup program on, an authentic GPSS World Commercial Version Distribution CD. Unfortunately, we are not able to support software obtained in any other way.

There are two important issues to be dealt with right at the beginning. First, you should ensure that if your existing Model Objects have become obsolete that they will be preserved, i.e. usable, after the installation. Second, you should then uninstall any existing installation of the same GPSS World product.

### 2.1.2. Preserving Obsolete Objects

If you are replacing an existing GPSS World installation, you should first determine if your old Model Objects will have to be updated. Although we try not to impose this inconvenience on you, occasionally the internal format of Model Objects must be changed. Before you begin the installation, download the current notices or view the general information page on the MinutemanSoftware.com Internet Web site to determine compatibility. If the Model Objects of the new version you are installing are incompatible with your existing Model Objects, you need to save your the text of each existing Model Object before you continue with the installation. To do so, open each of your Model Objects and use the Windows clipboard to save the text in each Model Object (.gps file) into a newly created Text Object. Then, after the installation, you can again use the clipboard to move the Model Statements into newly created Model Objects of the new kind. You do not have to update existing Text Objects. If you are not using the default Model Settings, you will also need to modify the Settings in each new Model Object. For that purpose, you may want to record all the old Settings before the installation. Settings can be viewed or changed using the **Edit / Settings** menu item in the Main Window.

Usually, the new Model Objects will be compatible with the old so that you won't have to do all this.

Although the internal format of Model Objects usually does not change from one release of GPSS World to the next, this is not true for Simulation Objects. Usually, but not always, you will have to retranslate your Models using the new version of GPSS World, thereby creating new Simulation

Objects. As long as your old Model Objects are compatible with the new ones, this is a simple matter. The only thing to worry about is that if you have made changes to the Settings of the old Simulation Objects. If so, you will have to change those same Settings in the new Simulation Object using the **Edit / Settings** menu command. You may want to record the old values before proceeding.

### 2.1.3. Uninstalling the old GPSS World

Before beginning the installation, we recommend that you remove any existing old installation of the same GPSS World Product. (The Student and Commercial Versions are different products). This will not remove your own Objects, only the ones that were installed originally with GPSS World. The Uninstall operation is done through the Add / Remove folder in the Control Panel of your operating system. Usually, your old Commercial Version software key will be preserved if you install into the same directory. You can install the Student Version of GPSS World even if you already have the Commercial Version installed, and vice versa.

After any old installation has been removed, you are ready to install GPSS World. If you are installing from a CD into a system with the autorun capability enabled, simply inserting the CD will start the Setup program. Otherwise, open the CD icon in your My Computer folder and double click on the Setup icon.

It's even easier if you are installing from a World.exe or Student.exe self-extracting file downloaded from MinutemanSoftware.com. Just double click on its icon and the GPSS World setup program will begin.

### 2.1.4. Installation Requirements

First, you should check that the installation requirements have been met.

#### Configuration Requirements

1. The use of the GPSS World Package requires an IBM compatible computer running Windows 98 or a compatible operating system.
2. We recommend an Intel Pentium III CPU, or better.
3. Floating point math coprocessor capabilities are strongly recommended, but not required. Most CPU chips marketed today have floating point processors.
4. A mouse is strongly recommended. Although simulations can be built and run without a mouse, some optional operations require the use of one.
5. At least 32 megabytes of random access memory (RAM) is recommended.
6. At least 15 megabytes of hard disk space is required for installation of all components of the Commercial Version. More is recommended for holding Model Files. In addition, adequate swap space must be available to the Windows Virtual Memory Manager.
7. CD drive is required only if installation is from CD medium.
8. A 16 digit numeric user key is required for the operation of the software. This numeric key must be obtained from MinutemanSoftware.com after the basic installation process has completed.

#### Version Restrictions

1. The Student Version of GPSS World is limited to simulations containing no more than 180 GPSS Block Entities, and the Personal Version is limited to 2000. The Commercial Version can run simulations of millions of blocks since it is limited only by the size of the virtual memory provided by your operating system. For optimal performance you should have enough RAM on your personal computer so that all simulations to be run at one time can fit in it.
2. The Student Version cannot open simulations saved by the Personal or the Commercial Version of GPSS World. The Commercial Version can open any simulation of the same version number, as well as those with other compatible version numbers.
3. The Student Version of GPSS World does not support the GPSS/PC Compatibility Mode. The Personal and Commercial Versions do.

## 2.1.5. File Folders/Directories

Although you can choose otherwise, the GPSS World setup program normally installs the software in the Minuteman Software folder in your computer's Program Files folder. In the following discussion we will use "folder" and "directory" synonymously. The Commercial Version of GPSS World will be placed in a folder named Commercial GPSS World in the Minuteman Software folder. The Samples folder, which holds all the examples, is then placed in that. Files pertaining to the operation of GPSS World are normally placed in the "module directory", i.e. the one containing the GPSS World executable files. Other sharable executables may be placed in one of your computer's system directories if they are not already present.

The operation of GPSS World results in the creation of several different files on your computer. Files for notices, registration information, and user keys are created in the module directory. You Control the other file placements when you open and save GPSS World Objects. Just as with GPSS World Objects, Data Streams can be directed to specific directories by your GPSS Open Statement. If you specify only a filename and not a path, the directory containing your Simulation Object is used.

Unless you direct otherwise in the **File / Save As** menu item, Simulation Objects will be placed with their parent Model Objects, and Report Objects will be placed with their parent Simulation Objects. Similarly, the search for Include-files begins in the Model Object's folder, or for Data Stream Text Objects in the Simulation Object's folder.

GPSS World can now invoke external programs directly by dynamic **Call()** Procedures. The rules for the location of the executable files containing the target functions are detailed in Chapter 8.

When you begin to build your own models you should create directories which make sense with respect to the project you are working on. For example, if you are investigating two very different inventory control strategies that need two very different models to simulate them, you may find advantages in placing them in separate directories. On the other hand, variants on a theme may best be organized in subfolders with the same parent.

## 2.1.6. The Installation Procedure

The use of all GPSS World products is governed by a license agreement which can be displayed during installation or by selecting **Help / About** in the Main Window of the program. Once you begin the installation, the GPSS World setup program directs you through the process. After a short initialization, the first GPSS World Setup Window appears.

## 2.1.7. Software Activation

After you install a GPSS World Microsoft Windows Installer (msi) file downloaded from [www.minutemansoftware.com](http://www.minutemansoftware.com), you must activate the software. This is done by acquiring a 16 digit numeric key, so that the software can run where you installed it. If you choose to use the Online Activation method, the activation process is almost automatic.

You may activate your GPSS World installation by either an Online Activation or an Email Activation. Either way, you must first prepay for the activation by clicking the ♦Buy Now♦ button on our website, [www.minutemansoftware.com/downloads](http://www.minutemansoftware.com/downloads). Alternately, Users of the Personal Version or the Commercial Version of GPSS World can contact us directly by email ([sales@minutemansoftware.com](mailto:sales@minutemansoftware.com)) for prepayment.

When you activate GPSS World you must indicate where your prepayment can be found. You do this by specifying a ♦Minuteman Invoice Number♦, a ♦PayPal Transaction Number♦ or a ♦Minuteman Authorization Number♦.

The first of these, the ♦Minuteman Invoice Number♦ resides in a cookie and if not disabled on your computer, allows you to activate by simply clicking a single button.

The ♦PayPal Transaction Number♦ can be used, as well. You can find it on the email confirmation sent to you from PayPal.

Finally, sometimes it is convenient to get an authorization number directly from Minuteman Software, the ♦Minuteman Authorization Number♦, which allows you to proceed with your activation.

After you have indicated your payment method, you may choose either an Online Activation or an Email Activation.

An online activation will immediately attempt to get your 16 digit software key from our registration server. If successful, you will be able to run simulations on GPSS World immediately. If not, you may retry the online activation or perhaps you may choose to activate by email, instead.

An email activation will cause an encrypted message to be sent to our registration server. If all is in order, an email message containing the 16 digit software key will be sent to the email address you specify in the registration information. If you have trouble sending email from the computer where GPSS World was installed, there is an option that places the encrypted message on the clipboard of your Windows operating system. You can then use a text editor to move the message to a place where you can generate an email message to be sent to [Activation@MinutemanSoftware.com](mailto:Activation@MinutemanSoftware.com). Your 16 digit software key will be sent to the email address you specified in the registration information.

## 2.2. The GPSS World Environment

We now begin a discussion of the manner in which you can control your new simulation environment.

### 2.2.1. The Main Window

The GPSS World simulation environment is the collection of all the actions available to you when you open a Session. Everything is controlled from the GPSS World Main Window, an example of which is shown next.

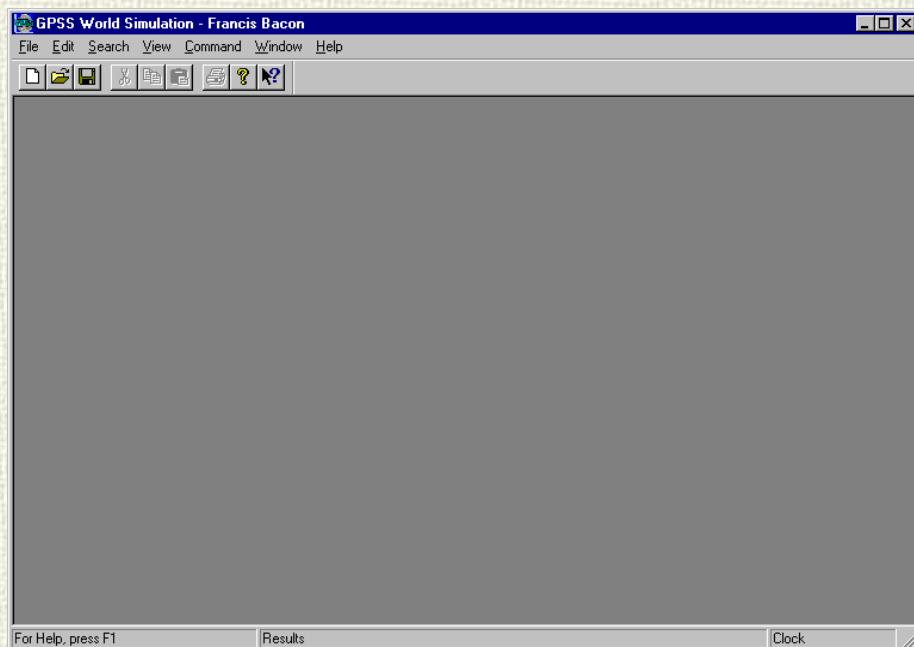


Figure 2\_13. The GPSS World Main Window

The Main Window has several components. The Title Bar is at the very top. Below that is the Menu, and below that the Toolbar (which can be hidden by toggling the **View / Toolbar** menu command). Most of the action occurs in the large empty client area.

At the very bottom of the Main Window is a Status Line that is divided into three parts. The left pane of the Status Line shows Command Prompts that give you more information on the menu items you are about to use. The middle pane of the Status Line shows status and error messages send from the GPSS World Objects you are interacting with. Finally, there is a small pane at the right side of the Status Line that is used to display the changing System Clock of running simulations. You can turn this on or off for each Simulation Object by using the **View / Show Clock** menu command.

### 2.2.2. A Quick Overview

Using the menu of the Main Window, you can create and manipulate the Objects of GPSS World: Model, Simulation, Report, and Text Objects. Each of these will have one or more views that can be opened within the GPSS World Main Window.

A typical project begins with the creation and modification of a Model Object. A Model is just a list of Model Statements, defined in Chapters 6, 7, and 8. After the Model is composed the next step is to create a Simulation Object. Each time you select **Command / Create Simulation** or **Command / Retranslate** from the Main Window's Menu, GPSS World Translates the Model Statements into a runnable Simulation Object. If errors need to be corrected, a single keystroke (representing **Search / Next Error**) can cycle through the circular error list, setting the cursor in the Text View of the Model, just before each error.

Once a Simulation Object is successfully Created, you can send any Model Statement to it to change the structure or state of the simulation. Normally, you will send Commands, such as START, but you can redefine entities or PLUS Procedures, as well.

You can send Model Statements to existing Simulation Objects using one of several methods.

First, some of the most common commands are listed in the **Command** menu of the Main Window. Most have keyboard accelerators as well.

Second, the Custom Command dialog is invoked through **Command / Custom** in the Main Window's menu. Here you can type any Model Statement -- even a PLUS Procedure -- and send it to an existing Simulation Object.

Third, you can load the set of function keys with your own choice of Commands. This is done through the **Edit / Settings** menu item in the Main Window. Thereafter, pressing the function key sends the Command to the currently selected Simulation Object.

Complex PLUS Procedures and lengthy Command lists can be sent using an INCLUDE Statement associated with a previously created Text Object. You can enter INCLUDE as a custom command (**Command / Custom**) or load it into a function key. After an INCLUDE Statement is loaded into a function key (**Edit / Settings**), a single keystroke sends the whole lot to the Simulation Object for processing.

During the debugging phase of your project, be sure to make use of all the visualization tools at your disposal. There are 10 dynamic windows and 7 snapshot windows that let you view the running simulation. Most require no effort on your part except to open them. Some, like the Blocks Window, add point and click debugging facilities for manipulating stop conditions and for controlling the advance of the simulation during debugging.

A new PLUS Trace feature can be turned on in the Settings Notebook. It will place a description of each PLUS Procedure invocation in your Journal Window. Here's a useful tip: If you define a PLUS Procedure that takes arguments but does nothing, you can place invocations of it anywhere in your PLUS code to evaluate and trace your own User Variables -- both Global and Temporary.

Among the dynamic windows are the Plot and Expression Windows which allow you to visualize the evaluation of any PLUS expression as it changes dynamically. The Tables Window can also be very enlightening, visualizing the convergence of frequency distributions. In addition, there are seven snapshot windows that present detailed state information at some instant within the simulation.

GPSS World is strong in the extensiveness of its Standard Reports. Unless you choose otherwise, when a Simulation Object completes a simulation (i.e. the Termination Count reaches zero), it creates a Report Object containing a Standard Report describing the final state of the simulation. You can modify the content of Standard Reports by editing the Settings of the Simulation Object. Interim reports can often be used effectively during the debugging/testing process. While interacting with a Simulation Object you can use the REPORT Command to create a Report Object describing all internal GPSS Entities.

You can extract just about any information you desire from a simulation by using Data Streams. These are associated with a set of GPSS Blocks and PLUS library procedures (OPEN, CLOSE, READ, WRITE, SEEK) that allow you to trace and collect derived information from the simulation and write them to plain text files. Such traces can be used to drive photo-realistic animations, as well. Within GPSS World, text files used in association with Data Streams are represented as Text Objects. The new Call PLUS procedures provide direct online access to external programs such as ad-hoc animators.

After the Model is thoroughly tested, it is time to define an Experiment to quantify results and test their statistical significance. GPSS World excels in this regard. Its built-in programming language, PLUS, supports a special kind of PROCEDURE statement, called, naturally, the EXPERIMENT Statement. Within an experiment, you can run and analyze complex experiments unattended. You can even get an automatic Analysis of Variance of the results. These features are developed in the OneWay.gps Sample Model, and further in Lessons 19 and 20 of the *GPSS World Tutorial Manual*. Chapter 13 of this manual discusses ANOVA and the Automatic Experiment Generators in depth.

## Getting Help

GPSS World provides several Help aids to keep you on track. The most immediate is the Online Help feature. Online Help is available throughout GPSS World. The [F1] Function Key will bring up a Help Panel for the active window or dialog. In addition, if you press the [F1] key when a menu item is highlighted, a context sensitive Help Panel will appear with information pertaining to that menu item. Within the Text View of an object, select a word and press the [F1] key to go to the keyword's description in the GPSS World Language reference section of the Help file. Finally, Online Help can be initiated by a Help button in some dialog boxes.

The Internet offers more help on GPSS World. HTML Online documentation can be found at the Minuteman Software Internet Web site, [MinutemanSoftware.com](http://MinutemanSoftware.com). This site offers many other support options as well. You can get to it by

### SELECT File / Internet / GPSS Web Page

in the Main Window's menu.

The [MinutemanSoftware.com](http://MinutemanSoftware.com) site contains other useful resources and ways to communicate with other users and consultants. The Student Version of GPSS World can be downloaded from the Web site free of charge. The official documentation of GPSS World includes this manual, the *GPSS World Reference Manual*, and the *GPSS World Tutorial Manual*. Both are available on the Web site and in hardcopy, as well.

Don't overlook the extensive set of sample models. Many professionals have found that one of the fastest ways to acquire simulation skills is to examine the work of others. The sample models in every GPSS World installation include a wide variety of applications of simulation to real-world problems.

## The Main Menu

Most of the action begins at the menu of the Main Window. Each of the top-level menus

**File Edit Search View Window Help**

have submenu items, most of which are enabled only when appropriate. To help guide your actions, GPSS World disables irrelevant menu commands. When disabled, menu commands are gray and cannot be activated. When there is ambiguity as to which Object is to receive the Command, you will have to select one of the object's windows with the mouse or keyboard in order to activate the related menu commands.

Now, let's look at each top level menu.

### The File Menu

The leftmost menu is the File Menu. It is used to open and save Objects, print views of Objects, access the Internet to download notices and Internet Web pages, and to register the software for users of the Commercial Version.

### The Edit Menu

The top items in the Edit Menu are for working with text. All GPSS World Objects have a Text View that can be modified using these menu commands. **Undo** is a "one-level backup" facility for removing the effects of the last Text View interaction. **Cut**, **Copy** and **Paste** move selections to and from the Windows clipboard. **Insert Line** and **Delete Line** operate at the cursor location in a Text View. The **Font** command opens a dialog allowing customization of the view.

The remaining five items allow you to modify attributes of GPSS World Objects. **Expression Window** and **Plot Window** open a dialog which allow you to change the contents of an existing dynamic Expression Window or Plot Window. **Insert Block** opens the **Block Input Menu** from which you can open a **Block Creation Dialog**. These Dialogs are available for guiding the creation of GPSS Blocks. When you complete a Block Creation Dialog and click OK, the new Block Statement is inserted after the Insertion Point in the Model. Block Statements created this way contain tabs. You can adjust tabstops in the Report page of the Model Object's Settings. **Insert Experiment** opens a dialog which allows you to automatically generate either an efficient screening experiment or a sophisticated optimizing experiment and insert it into a Model Object. See Chapter 13 of this manual for a discussion of the Automatic Experiment Generators.

**Edit / Settings** allows you to view and modify the Settings of a Model, Simulation, or Report Object. These attributes are discussed in detail below.

## The Search Menu

The Search Menu helps you navigate inside the text view of an object.

The first item, **Find / Replace**, opens a dialog that lets you move to the next occurrence of specific text, and optionally replace it. A Replace All option is available, as well.

The **Go To Line** item will jump to a line number in the text. This is useful at times when an error message from a simulation mentions a line number.

The next set of items is for Bookmarks. They let you place invisible marks that get saved with the object. They form a circular list that you can traverse via the appropriate menu commands or keystrokes. The **Next Bookmark** menu command jumps to that position in the text corresponding to the next Bookmark in the list. **Mark** enters a Bookmark at the current cursor position. **Unmark** removes the current Bookmark. **Unmark All** removes all Bookmarks, **Select to Bookmark** selects text from the current cursor position to the current Bookmark position.

The last two items in the Search menu deal with Translation error messages. When you try to Create a Simulation Object from a Model which contains syntax errors, a circular list of error messages and locations is built into the Model Object, replacing any that was there before. This list becomes part of the Model Object and is saved with it. To traverse the list in either direction use the **Next Error** and **Previous Error** menu commands to display the error message in the Status Line at the bottom of the Main Window. At the same time the cursor is placed just before the error in the Text View so you can correct it. The keyboard accelerators **[Ctrl]+[Alt]+[N]** and **[Ctrl]+[Alt]+[P]** let you navigate the list easily from the keyboard.

## The View Menu

The View menu includes menu commands for controlling what is shown in a few of the windows.

The first item, **Notices**, allows you to view the notices concerning GPSS World. You can update them from minutemansoftware.com if you need to.

With the second item, **Toolbar**, you can hide or show the Main Window's Toolbar by toggling the first item Toolbar.

The third item, **Entity Details**, lets you change the format of some of the dynamic simulation windows to show more or less detail onscreen.

The last menu item, **Clock**, toggles a switch in the current Simulation Object. If this switch is on in the current Simulation Object, changes in the System Clock of the simulation are displayed in the lower right of the Status Line of the Main Window when one of that Simulation Object's views is the active window. This is a handy way to assure yourself that a simulation is indeed running, but it slows down the simulation somewhat.

## The Command Menu

The Command Menu is concerned with creating and manipulating Simulation Objects. The **Create Simulation** menu command invokes the high performance Translator to create a Simulation Object based on the Model Statements in an existing Model Object and its Include-files. Normally you will encounter errors that must be corrected (see **Search / Next Error**, above). The **Retranslate** menu command is available to perform the Translation again when errors have been corrected.

Next, **Repeat Last Command** is an easy way to do some action again on behalf of the same Object.

The remaining menu commands represent formal GPSS Commands. Most have accelerator keys for easy use and most can be loaded into a function key, as well. These menu commands are disabled unless the current Simulation Object is ready for them.

You can click on the last item in the Command submenu, **Custom**, in order to write your own Model Statement, not listed elsewhere.

## The Window Menu

The top two items in the Window Menu can be used to tidy up when many windows are open in the Client Area of the Main Window. **Tile** will cover the client area of the Main Window with a small version of all open windows and **Cascade** will stack them in a handy stack where all title bars can be seen.

The next two items have submenus themselves. They are used to open Windows and Snapshots on existing Simulation Objects. These individual windows are discussed below. They are disabled until the current Simulation Object is ready to oblige.

## The Help Menu

The Help menu can be used to start the Online Help system. In addition, the bottom item **About GPSS World** displays the software version and the license agreement.

### 2.2.3. Settings

Settings are collections of values that form a common thread throughout most of the GPSS World Objects. They are used to control the simulation, the appearance of reports, and the contents of windows and loadable function keys.

Settings are inherited. When you Create a Simulation Object, it inherits its initial Settings from its parent Model Object. So if you want to set values for all the simulations in your project, you should set them in the Model Object.

Similarly, Report Objects inherit their settings from the Simulation Object that created them. In this way, the Report contains a read-only record of the Settings in effect when the simulation was run.

Text Objects are different. They are mere representations of plain text files and do not contain a copy of Settings. This is the reason that they are the only GPSS World Objects that can be opened by external text editors. The other objects contain Settings, Bookmarks, and in the case of Model Objects, Result Lists, that are saved with the object and that confound external word processors. If you need to, you can always transfer just the text to and from the text view of these GPSS World Objects using the Windows clipboard.

To open a Settings Notebook you must first select an open GPSS World Object. Then, its Settings are viewed or modified by selecting **Edit / Settings** after clicking on the GPSS World Object to be changed. This presents the Settings Notebook for that Object, with 5 tabbed pages.

## The Simulation Page

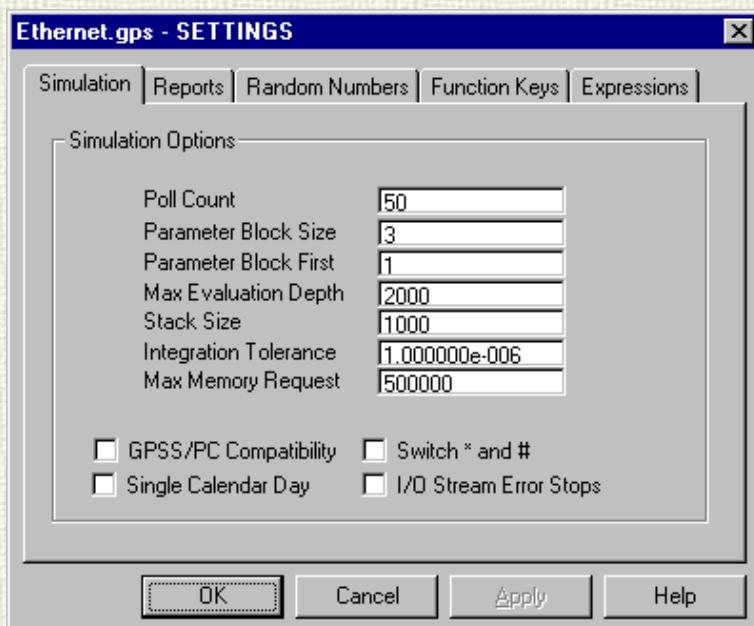


Figure 2◆14. The Simulation Page of the Settings Notebook

The Simulation Page contains several values that you can change which affect the limits and behavior of running simulation.

The **Poll Count** Setting is the number of Block entries to be attempted before checking for any kind of interruption, such as a Command from the user.

If you use a lot of GPSS Transaction Parameters in your model, you can speed up simulations a little by putting them into a Parameter Block. To do this enter the number of Parameters to be blocked into the **Parameter Block Size** field, and the Parameter number of the first Parameter in the block in the **Parameter Block First** field. If you are using Names instead of numbers to refer to these Parameters, you should control the values given to Parameter Names by using EQU Statements.

The **Max Evaluation Depth** is used to protect against circular references in sets of PLUS Procedures. If your simulation is highly nested without circular references, you can increase this value to accommodate any level of nested Procedure calls.

The **Stack Size** is used to allocate stack space for nested PLUS Procedures and library procedures. You can save space by decreasing this value, but if you get a Stack Overflow Error Stop during a simulation, you may have to increase it.

The **Integration Tolerance** field is used by the continuous simulation phase. If you loosen this tolerance by making it larger, your integrations will be faster but may lose some accuracy.

The **Max Memory Request** places a limit on the acquisition of RAM memory for structures such as tables and matrices. Such a limit is useful to catch errors, but you can increase it as necessary. If your RAM memory demands exceed that available in your computer, your operating system's virtual memory manager will have to increase its disk activity and perhaps its disk swap space usage significantly.

The four checkboxes at the bottom of the page control GPSS/PC Compatibility, which is available only in the Commercial Version, and how Data Stream errors are to be handled in the simulation.

CHECK the **GPSS/PC Compatibility** checkbox to operate the simulation in GPSS/PC Compatibility mode as described above in Section 1.5. This option automatically invokes the Single Calendar Day option, described next.

CHECK the **Switch \* and #** checkbox to cause the [\*] character to be used for multiplication and [#] for GPSS indirect addressing instead of the other way around.

CHECK the **Single Calendar Day** checkbox to cause simulations to use a single FEC time segment list instead of multiple ones. In certain simulations this may be a little faster, but will not alter the results of the simulation.

CHECK the **I/O Stream Error Stops** checkbox if you want any errors other than "File Not Found" and "End of File" encountered by Data Streams in a simulation to HALT the simulation. If this box is unchecked you can handle such errors yourself within the simulation. See Section 4.16 for more information on Data Streams.

## The Reports Page

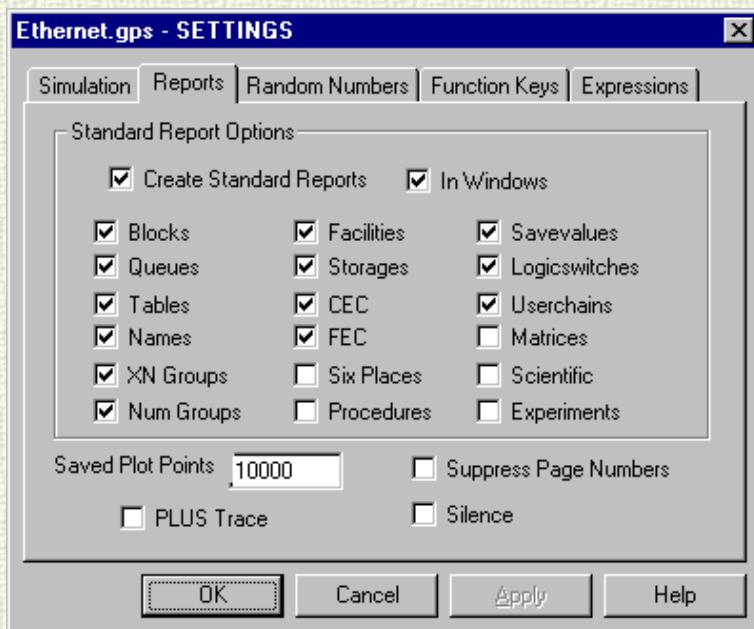


Figure 2♦15. The Reports Page of the Settings Notebook

The Reports Page is concerned primarily with the appearance of Reports and Plots.

The Standard Report Options groupbox contains the itemized GPSS Entity subreports to be included in each Standard Report. Check those to be included.

CHECK the **Create Standard Reports** checkbox to enable the creation of Standard Reports automatically when every simulation ends (i.e. when the Termination Count reaches zero). The REPORT Command is still available even if this is left unchecked.

CHECK the **In Windows** checkbox to open a Text View of each Report Object instead of just saving them in a file. This checkbox depends on the previous checkbox being checked.

Check the **Six Places** checkbox to set the printed precision of fractional real values to 6 decimal places instead of 3 in Reports and Status Messages.

Check the **Scientific** checkbox to use the mantissa-exponent representation of extreme numbers in reports and in Data Streams.

Below the Standard Reports Options groupbox are four more Settings.

The **Saved Plot Points** value tells GPSS World how much space to allocate to a circular buffer for recent Plot points. If you scroll or print a Plot Window, these values are used to rebuild the plot. If you don't have enough, the leftmost segment of the scrolled or printed plot will be missing. A value too large can waste virtual memory.

Check the **Suppress Page Numbers** checkbox to prevent GPSS World from inserting page number headers when printing the object's Text View.

Check the **PLUS Trace** checkbox to enable the Journalizing of a trace message for each PLUS Procedure invocation executed by your Simulation Object. Arguments are evaluated before writing them to the Journal Window. If you insert calls to some simple procedure and turn on the PLUS Trace you will get a trace of your variables (i.e. the ones used as arguments) in the Journal Window.

Check the **Silence** checkbox if you errors associated with this object to be reported silently, with no audible beep.

## The Random Numbers Page

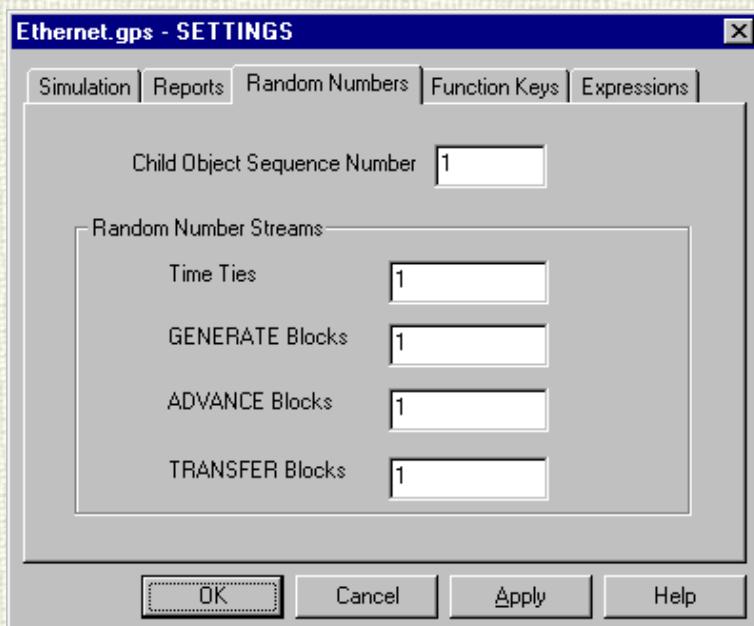


Figure 2◆16. The Random Numbers Page of the Settings Notebook

The Random Numbers Page lets you control the internal Random Number Streams of the simulation and the external numbering of child objects.

To keep naming of objects unique, GPSS World adds a sequence number to the names of newly created child objects. The number to be used for the next sequence number is kept here, in the Settings of the parent object. To change it just modify the **Child Object Sequence Number** field.

The Random Number Streams groupbox contains the remaining settings on the page. They determine which GPSS Random Number Stream is to be used for the purpose of resolving Time Ties, time randomization in GENERATE and ADVANCE Blocks, and Next Block in TRANSFER Blocks.

The **Time Ties** entry field allows you to specify that when imminent events occur at the same time, the next event is to be selected randomly. To do so, place the Entity Number of the Random Number Generator to be used in the **Time Ties** entry field. If you use a value of 0, time ties will not be randomized. A fuzz value of one part per billion is used in the equality criterion for real time values.

The **GENERATE Blocks** entry field allows you to specify which Random Number Generator is to be used when a GENERATE Block calculates an interarrival time from Operands A and B. If you specify a nonpositive number, Random Number Generator number 1 is used.

The **ADVANCE Blocks** entry field allows you to specify which Random Number Generator is to be used when an ADVANCE Block calculates a delay time from Operands A and B. If you specify a nonpositive number, Random Number Generator number 1 is used.

The **TRANSFER Blocks** entry field allows you to specify which Random Number Generator is to be used when a TRANSFER Block selects a probabilistic destination Block. If you specify a nonpositive number, Random Number Generator number 1 is used.

## The Function Keys Page

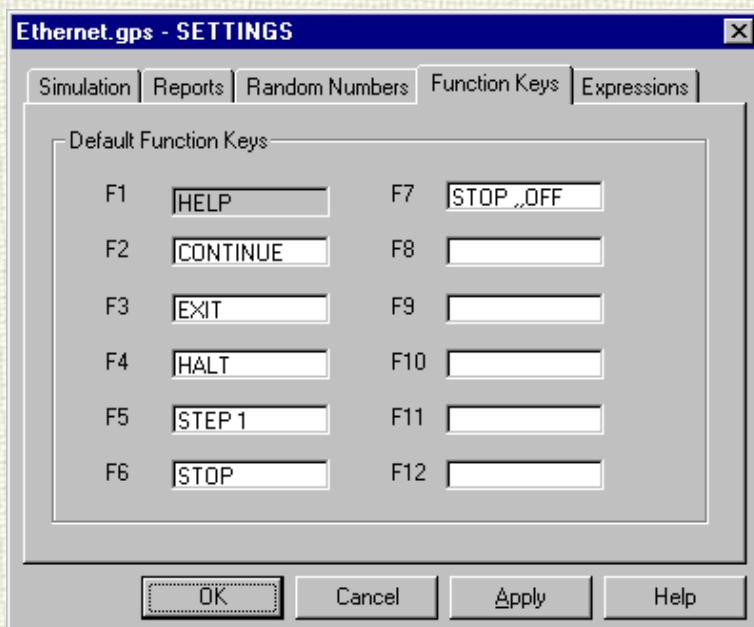


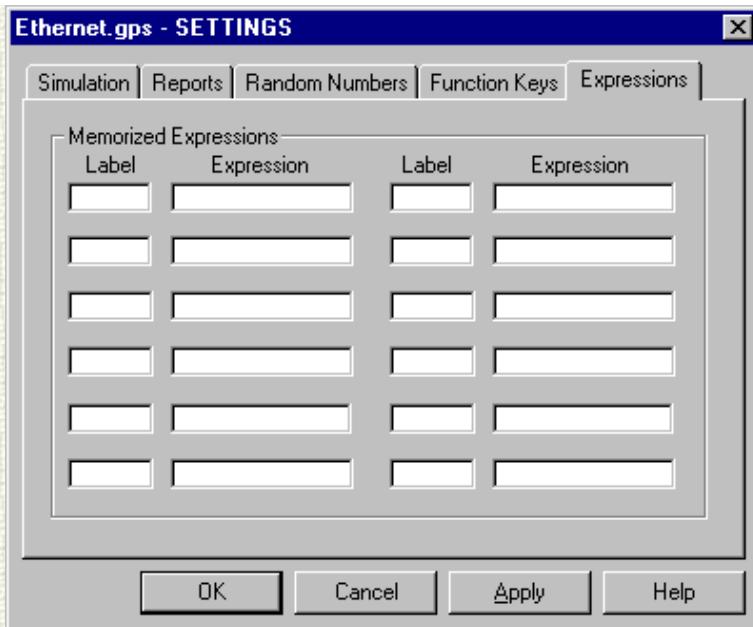
Figure 2♦17. The Function Keys Page of the Settings Notebook

You can enter GPSS Statements into the fields on this page to load them into Function Keys, and/or you can change the tabstops of a Model Object.

When a Simulation Object is selected all you have to do to send the loaded GPSS Statement is to press the Function Key. Using an INCLUDE Statement permits you to send a Statement list of any complexity, including PLUS Procedures and Experiment definitions.

The Model Tabstops section allows you to set the tabstops of Model Objects. This may be desirable when you use the Block Creation Dialogs which themselves use tabs in the Block Statements they create. Tabstops are disabled for Simulation or Report Objects. Tabstops must be nonzero and strictly increasing from the first (leftmost) to the last. You can specify the tabstops in either of two units: characters or twips. There are 1440 twips in an inch (approximately 567 in a centimeter).

## The Expressions Page



*Figure 2◆18. The Expressions Page of the Settings Notebook*

When you open an Expression Window or a Plot Window on a Simulation Object, you must specify the expression to be watched as well as a name to be used as a label. This page allows you to save complex expressions with the Simulation Object. Then, when you open a Window using the expression, you can select it from a list instead of having to retype it.

Both the Expression Window and the Plot Window give you a chance to save expressions currently in use in the window. You can do this by

**SELECT    Edit / Expression Window**

and

**SELECT    Edit / Plot Window**

And then selecting the expression and

**CLICK ON    Memorize**

Alternately, you can type the label and expression into this page of the Settings Notebook. The effect is the same. You will now be able to bring back the labeled expression simply by selecting it.

## 2.3. Controlling a Session

Normal installation of GPSS World places a line in your operating system's **Start / Programs** Menu that you click on to start the program. Alternately, you can create a shortcut to GPSS World and place it on your personal computer's "desktop", or anywhere you like. Simply drag the GPSS World icon from the **Start / Programs** menu.

You can open an object in GPSS World Session merely by double-clicking on it. When you run GPSS World the first time, it registers the file types and extensions of GPSS World with the Windows operating system. This allows Windows to associate GPSS World with any of its Object files. Thereafter you can double click on any of the file icons to start a GPSS World Session.

### 2.3.1. Using the Main Window

Once you have started a Session you can control it using your mouse and the keyboard. You can choose menu items by clicking with your mouse, by using menu mnemonic keys, and by using accelerator keys. Many interactions require you to type information using the keyboard, and the set of Functions Keys on your keyboard can be loaded with any Commands you choose. As described in the previous section, you edit the Settings of an object in order to load Commands into the Function Keys.

Before you begin, you should familiarize yourself with the Main Menu. A quick overview is presented above, in Section 2.2.2. Many of the Main Menu items are disabled and appear gray until they can be used. As you open and create GPSS World Objects, the valid menu commands will become enabled as appropriate.

## Using The Main Menu

Although all the GPSS World Objects have small menus on their own windows, most actions start in the Main Menu. All menus are used in a similar fashion. Some menu sequences result in the creation of a Dialog Window containing several controls such as buttons and entry fields. Entry fields normally require you to use the keyboard to type in additional information.

To initiate the menu actions, you have three choices. First, you can simply use the mouse to click on the succession of menu items. Second you can press the **[Alt]** key to activate the menu, and then press the underlined (i.e. the mnemonic) letter in each menu item, in sequence. Third, if the lowest level menu item has an accelerator key listed to the right of the item name in the menu, you can simply press that combination of keys to begin the action, if it is enabled.

## The Accelerator Keys

Accelerator keys are key combinations that immediately enter a menu command into GPSS World. Usually, they are listed just to the right of the menu item they effect. They are enabled only when their associated menu item is. To activate an accelerator key, hold down all keys in the combination except the last, then press the last key, then release all keys.

### GPSS World Hot Keys

Hot keys are quick ways to send GPSS World Commands to a Simulation Object. They are:

**[Ctrl]+[Alt]+[S]** - Create Simulation

**[Ctrl]+[Alt]+[R]** - Retranslate

**[Ctrl]+[Alt]+[L]** - Repeat Last Command

**[Ctrl]+[Alt]+[C]** - CONTINUE

**[Ctrl]+[Alt]+[H]** - HALT

**[Ctrl]+[Alt]+[1]** - STEP 1

### Text View Navigation Keys

Navigation Keys let you jump to specific locations in a Text View. They are:

**[Ctrl]+[Alt]+[B]** - Next Bookmark

**[Ctrl]+[Alt]+[G]** - Go To Line

**[Ctrl]+[Alt]+[N]** - Next Error

**[Ctrl]+[Alt]+[P]** - Previous Error

**[Ctrl]+[Alt]+[F]** - Find / Replace

### Text Editing Accelerators

Editing is faster using Text Edit Accelerators instead of menu selection. These keys are:

**[Ctrl]+[Z]** - Insert Line

**[Ctrl]+[D]** - Delete Line

**[Ctrl]+[I]** - Undo

**[Ctrl]+[C]** - Copy to Clipboard

**[Ctrl]+[X]** - Cut to Clipboard

**[Ctrl]+[V] - Paste from Clipboard**

also,

**[Ctrl]+[Ins] - Copy to Clipboard**

**[Shift]+[Del] - Cut to Clipboard**

**[Shift]+[Ins] - Paste from Clipboard**

## General Accelerators

There are several other general purpose keys. They are:

**[Ctrl]+[O] - File Open**

**[Ctrl]+[S] - File Save**

**[Ctrl]+[P] - File Print**

**[Alt]+[F6] - Next Pane**

**[Shift]+[F6] - Previous Pane**

## Online Help

The **[F1]** key opens the GPSS World Help System any time it is pressed. You can then explore the contents and index of the "Windows Help" form of the *GPSS World Reference Manual*.

In addition, if a menu item or control is selected, **[F1]** will open context sensitive help on that topic. Or, if the cursor is embedded in a GPSS World keyword, such as a Command Name, in a Text View, if you press the **[F1]** key, you will open the online Reference manual to information on that topic.

## Error Messages

When an operation cannot be completed, GPSS World displays an Error Message in a dialog box or a Text Window. Most messages are intended to be self-explanatory. Even so, a further explanation of each message can be found in Chapter 14 of this manual.

## Using Text Windows

The Text Windows of all GPSS World Objects have text editing capabilities. You can enter text into a Text Window by opening a file, by typing, or by pasting from the clipboard. Text can be modified in a variety of ways.

There are two place-marking indicators that you can use when a Text View has the input focus. In a Text View, the mouse pointer is the I-beam icon that travels as you move the mouse. The cursor, or insertion point, is the flashing vertical line in the text window that marks the text insertion point. Text you type is placed at the insertion point indicated by the cursor.

The editing operations available are:

Move the cursor to the position of the mouse pointer by clicking mouse button 1.

Move the cursor by arrow keys in the keypad.

Move the cursor and scroll using the **[PgUp]** and **[PgDn]** keys.

Move the cursor to the beginning and end of a text line by pressing **[Home]** and **[End]**, respectively.

Scroll using the scroll bars of the Text Window.

Break a text line with **[Enter]**.

Delete selected text with **[Del]**.

Undo last edit with **Edit / Undo**.

Delete line with cursor with **Edit / Delete Line**

Insert new line after cursored line with **Edit / Insert Line**.

Selection operations include

Select word by double click on button 1.

Swipe select by click and hold button 1, move mouse, release button 1.

Extended select by **[Shift]** + click mouse button 1 to select from cursor to mouse pointer.

Clipboard operations include

◆ Cut selected text by **Edit / Cut**. Delete text and place copy in clipboard. You may also use the key combination **[Ctrl] + [X]**.

◆ Copy selected text by **Edit / Copy**. Copy text to clipboard. You may also use the key combination **[Ctrl] + [C]**.

◆ Paste to cursor by **Edit / Paste**. Insert a copy of text from clipboard. You may also use the key combination **[Ctrl] + [V]**.

Other operations using keyboard accelerators are described above.

You can change the font of text in a Text View. First select the text, then

#### CHOOSE **Edit / Font**

This will open the Font Dialog, where you can specify typeface, size, and character type for the selected text. GPSS World uses Courier as the default font. It has the advantage of being a monospaced font with uniform character widths. This has the advantage of easy column alignment in GPSS Statements.

### 2.3.2. Building Models

The physical process of model development starts with the creation of a Model Object that contains Model Statements written by you. Model Statements are defined in Chapters 6, 7, and 8 of this manual. Often it's easiest to begin building a simulation by modifying an existing model. The Samples folder is rich source of small GPSS World Model Objects.

Model Statements can exist in additional Text Objects (.txt) as well as the primary Model Object (.gps), permitting you to share code across projects. Although there's only one Model Object in a simulation, there can be many Text Objects. These are connected to the Model Object with INCLUDE Statements, which take the name of the Text Object file (.txt) as an argument.

Model development begins by opening a Text View of the Model Object. A complete set of text editing operations is available for modifying and adding Model Statements. If you prefer, the use of the Block Input Menu and one or more Block Creation Dialogs makes it relatively easy to correctly specify the Blocks Statements in your Model. Each Block Statement is inserted in a line following that of the flashing insertion point in the Model Object. Online Help guides you all the way and syntax errors are prevented.

A model must be Translated in order to Create a Simulation Object. The Translation is started via the **Command / Create Simulation** item in the main menu of the Model Window. The Translator determines if there are syntax errors in the model. If so, a circular queue of error messages is created so that the errors can be found and corrected easily. If no errors are found, the resulting simulation automatically sent to the Simulation Object, replacing any that already existed. When the Simulation Object receives a simulation, it immediately performs all embedded Commands in the order encountered by the Translator.

## INCLUDE Commands

You can use an INCLUDE Command anywhere in a Model except in a PLUS Procedure. Normally, they are used in Model Objects to incorporate statements from one or more Text Objects. INCLUDE Statements can be nested to a depth of 5, but they cannot be used inside a PLUS Procedure.

The syntax of the INCLUDE Command is:

**INCLUDE "FileSpec"**

Where you replace the italicized word *FileSpec* with your file specification, including extension (normally .txt), and enclose it in double quotes. The default directory is that of the Model Object.

INCLUDE Commands have other uses, as well. You can type an INCLUDE Command as a Custom Interactive Command. This causes a whole file of Model Statements to be Translated and sent to the Simulation Object. An INCLUDE Command can be loaded into a function key, just like any other GPSS Command. In this manner a single keystroke can initiate a complex Command File or PLUS Procedure redefinition.

Before you use a file in an INCLUDE Command, you should Translate it by itself to remove syntax errors. To do so, copy its Model Statements to the clipboard, open a new temporary Model Object, paste into it from the clipboard, and Translate it using **Command / Create Simulation**. This allows you to correct any Syntax Errors easily before combining the Text Object into a more complicated model.

## Translating Models

Simulation Objects begin as Translated Model Objects. When you Translate a model by selecting **Command / Create Simulation**, the GPSS World Translator checks for syntax errors, creates a simulation, and sends it to a newly created Simulation Object. If successful, Translation activates all the Interactive Commands and Simulation Windows, so that the state of the newly created simulation can be monitored and controlled. After that, you can send any Model Statement to the existing simulation for processing.

GPSS World features a new high performance Model Translator that is hundreds of times faster than its predecessor. The largest models are Translated in no more than a few seconds. When the Translator detects one or more errors in the model, it creates a circular list of error messages that can be accessed from the Text View of the Model Object. You can traverse the error list quickly using the **Search / Next Error** or **Search / Previous Error** menu commands. Both have keyboard mnemonics and accelerators. Each stop in the list places the error message in the Status Line of the Main Window and places the insertion point in the Model Text View just before the syntactical element that caused the error. It's then a simple matter to correct each error and move to the next.

If an error occurs in an INCLUDED Text Object, and you cannot find it easily, you can paste those statements into a new temporary Model Object and Translate it by itself. It is best to place only tested statements into an include-file.

Blocks are placed in the simulation in the same order that they are encountered by the Translator. Commands are accumulated in an ordered list that is sent to the Simulation Object with the simulation. The list is scanned for Immediate Commands (SHOW, HALT, and CONDUCT), and the remainder are placed on the simulation's Command Queue and executed one after the other. Command Lists sent interactively to an existing simulation are treated the same way.

Dynamic simulation windows (not Snapshots) are automatically refreshed when you Translate a model to ensure that the view is current. If an entity can no longer be found, say, in a Table Window, the Window will be closed automatically.

## Controlling Simulations

GPSS Commands are used to control the running of simulations. They are discussed in detail in Chapter 6. They may be embedded in a model or they may be sent as Interactive Statements.

Any Model Statement may be sent to an existing simulation. Such statements are called Interactive Statements. You can control and view running simulations very closely with the operations that are available. To do so, use the **Command** menu in the Main Window. If the Command you need is not listed, select **Command / Custom** and type it into the entry field, and select **OK**. This action Translates the Command and sends it to the simulation. The Command menu is disabled until there is a simulation to receive interactions.

You may use GPSS Block Statements and PLUS Procedure definitions as Interactive Commands. Block Statements sent interactively to an existing simulation cause a temporary Block to be created and immediately destroyed after the Active Transaction attempts to enter it. This mode is called *Manual Simulation*. Procedure Statements sent to an existing Simulation Object register a user created PLUS Procedure, replacing any of that same name that may already exist.

Interactions are logged in the Journal View of the Simulation Object.

## Running

Normally, a START Command is used to begin a simulation. It could be entered interactively or could be part of the Model Object. The START Statement specifies the Termination Count, which is an important state variable of the simulation. The Termination Count must be decremented to zero for the simulation to end. Any TERMINATE Statement in the simulation can decrement this count.

When the Simulation Object executes a START Command, Transactions will begin to circulate in the simulation. Normally, the simulation will end by itself and will produce a Standard Report when the Termination Count reaches zero. If an error stop occurs, no report is written. Reporting options can be changed in the Settings of the Simulation Object. These are described above in Section 2.2.3.

Multiple simulations can be run unattended by using RESET, CLEAR, and START in a sequence of Commands. These are commonly used in PLUS Experiments, which can control and analyze extensive sets of simulation runs. PLUS Experiments are discussed in this manual in Section 8.2.4 , and in Lesson 19 of *The GPSS World Tutorial Manual*.

### Stopping a Simulation

You may stop a running simulation in several ways.

- ◆ First, you can interrupt a simulation by sending a HALT Command. HALT can be asserted by the **[Ctrl]+[Alt]+[H]** Hot Key, a Function Key, an Interactive Command from the Command menu, or as part of a Command File.
- ◆ Second, you can enter one or more stop conditions with the STOP Command, before you START or CONTINUE the simulation. STOP commands set conditions which when detected will cause the simulation to halt. When the simulation stops, previous stop conditions are not removed. If you CONTINUE a simulation after a stop condition was found, the original condition that caused the simulation to stop is ignored to allow the stopped Transaction to proceed. STOP conditions can be controlled by "Point and Shoot" operations in the Debug Toolbar using only the mouse. This is discussed below, and in Chapter 5.
- ◆ Third, you can resume the simulation with a STEP Command. After the specified number of Block entry attempts, the simulation will stop automatically.
- ◆ Fourth, when an error condition is detected, an Error Stop occurs. The simulation is halted and messages are sent to identify the reason.
- ◆ Fifth, you can simply wait for the simulation to complete. This occurs when the Termination Count falls to or below 0.

In each of these cases, when the simulation stops, the state of the simulation is reported to the Journal View of the Simulation Object.

### Interactive Statements

Any Model Statement can be sent to an existing simulation. These are called Interactive Statements. They can be Commands, PLUS Procedure definitions, or even Block Statements.

Every simulation has a Command Queue associated with it. The Simulation Object performs each Command on the queue one after the other, until it is HALTED or until it runs out of things to do. Even Commands in the Model, other than HALT and SHOW, are placed on the Command queue before they are performed.

Interactive Commands are either *Immediate* or *Queued*. Immediate Commands, such as HALT and SHOW, are performed as soon as the Simulation Object receives them. All other Commands are queued. They are placed at the end of a list of Commands which have not yet been completed.

The HALT Command is a special case. Not only is it an immediate Command, but it also deletes any remaining Queued Commands still on the Command queue. After a HALT Command is performed, the Simulation Object has nothing more to do on behalf of that simulation, for the moment, since the Command Queue is then empty.

Commands are not the only statements that can be used interactively. When a PLUS Procedure Statement is sent interactively, it causes the procedure to be registered within the simulation. Thereafter, the procedure can be invoked from any PLUS Expression. If a Procedure with the same name already exists within the simulation, it is redefined.

A PLUS Experiment is a special kind of Procedure. PLUS Experiments are can be sent interactively, as well. If any like-named PLUS Procedure (or Experiment) is currently registered with the Simulation Object, it is replaced. Then the Experiment can be invoked using a CONDUCT Command, which is discussed in Chapter 6.

Interactive Block Statements are called Manual Simulation Statements. When received by the Simulation Object, they cause a temporary Block to be created, and the Active Transaction to attempt to enter it. Then the Block is destroyed. In this manner, GPSS Blocks can be used as though they were commands, providing for a fine level of control over the simulation. Data Stream file I/O is supported interactively, as well. Data Streams are discussed in Section 4.16.

GENERATE Blocks are the only ones which cannot be used in Manual Simulation Mode. Instead, SPLIT Block Statements can be used to create Transactions interactively.

There are several ways to send a Model Statement to the simulation. You can use the Command submenu of the Main Window, you can use a "Hot Key", and you can associate a GPSS Statement with a Function Key in the Model Settings Notebook.

To send a Command List to the Simulation Object

CHOOSE **Command / Custom**

type the Command list, and

SELECT **OK**

This will Translate the Command list and send it to the simulation for execution.

It is often convenient to put a list of frequently used Commands in a small text file. You can then use an INCLUDE Command to send the whole sequence to the Simulation Object. Even easier, you can load a function key with an INCLUDE Command, and have the whole Command List performed by a single keystroke.

#### **Redefining Entities**

There are fifteen different entity types that you can create. The entity types are Transactions, Blocks, Facilities, GPSS Functions, Logicswitches, Matrices, Queues, Storages, Savevalues, Tables, Userchains, Variables, Numeric Groups, Transaction Groups, and Random Number Generators. The better you understand these entities and their properties, the more powerful your simulations will be. Chapter 4 is devoted to entities.

With the exception of Transactions, entities are never deleted from the simulation. However, some entity types can be redefined interactively. You can use STORAGE, TABLE, QTABLE, MATRIX, or VARIABLE Commands using the same label to do so. Similarly, PLUS Procedures may be redefined interactively.

A location label, or simply location, is a name used to label a Block Statement. This field is not required in a Block Statement. Unlike other entity types, Blocks may not be redefined in GPSS World. A similar flexibility can be achieved by using an EXECUTE Block and redefining its Operand A.

#### **Viewing**

A wide variety of windows are available for viewing simulations. Multiple windows of the same type may be opened on any simulation. All windows can be printed or saved for later use.

Simulation windows are divided into two categories: Snapshots and Dynamic Windows. All Simulation Snapshots and Windows are opened using the **Window** submenu in the Main Window. Use **Window / Simulation Snapshot** for a Snapshot and **Window / Simulation Window** for a Dynamic Window. The menus are disabled until there is a simulation to see.

Snapshots are not updated online. They take a single picture that represents a view of the simulation at a single instant.

You can take Snapshots of the following items:

- ◆ Current Stop Conditions.
- ◆ Any Transaction in the simulation.

- ◆ The Current Events Chain.
- ◆ The Future Events Chain.
- ◆ Numeric Groups.
- ◆ Userchains.
- ◆ Transaction Groups.

Dynamic Windows are updated as the simulation changes. They are refreshed when you Translate a model or cause a RESET event. You can open an online view of

- ◆ The Block Structure of the model.
- ◆ An arbitrary Expression list.
- ◆ The Facility Entities.
- ◆ The Logicswitch Entities.
- ◆ Any cross-section of a Matrix Entity.
- ◆ A Plot of multiple arbitrary Expressions.
- ◆ The Queue Entities.
- ◆ The Savevalue Entities.
- ◆ The Storage Entities.
- ◆ Any Table Entity.

Dynamic windows place a load on the system because large numbers of messages returned from the simulation must be processed. We recommend that only a few online windows be used at a time when the simulation must be run. It is much easier to open a large number of Snapshots and Windows when the simulation is HALTED first.

Each of these Snapshot and Dynamic Window types is presented in Chapter 5.

## Printing Windows

Any simulation window can be printed by selecting **File / Print** from the Main Menu.

## Testing

After you have created one or more GPSS models, you then normally enter a period of testing, tracing, and debugging your simulations. GPSS World provides you with many testing facilities, including visual aids, for use during the testing process. In addition, the high level of interactivity provided by GPSS World allows you to manipulate the simulations for the purpose of problem determination.

First, you need to verify that the simulation is behaving as you expect. The GPSS World graphical windows are ideal for this purpose. We suggest that you start the simulation, and then begin to explore the overall behavior of your models as they converge to steady state behavior.

One of the most useful windows for debugging is the Journal View of the Simulation Object. When you cause the Trace Indicator of a Transaction to be turned on, all Block entries of that Transaction cause a Trace Message to be displayed in the Journal View. You can place TRACE Blocks inside the model for this purpose, or you can use TRACE Statements in Manual Simulation Mode, to control the Trace Indicator of individual Transaction Entities. The PLUS Trace Setting can now be used to aid the development of PLUS Procedures. It is discussed below.

For serious debugging you will probably need to place stops at certain places in the simulation, and then observe the state of the simulation by taking snapshots and opening dynamic windows. Snapshots of the CEC and FEC, in conjunction with the STEP Command, are often sufficient to determine why certain events occurred. The FEC and CEC can be visualized in snapshots or can be displayed using a REPORT Command.

Difficult debugging problems can often be best tackled by running a simulation to an important time, and then saving the Simulation Object. Stop conditions and memorized expressions are automatically

saved with it. Then, by repeatedly reopening it as a new Simulation Object, you can rerun the simulation from the saved state.

All SNAs are accessible using the SHOW Command. They can be visualized dynamically in an Expression Window or a Plot Window.

Sometimes it is useful to make corrections in the middle of a debugging run. GPSS World allows you to redefine entities, redefine PLUS Procedures, change the values of user defined names, and to manipulate the simulation by entering Manual Simulation Statements. You can then continue the simulation without a retranslating.

### Error Stops

An Error Stop is an error condition that prevents a Statement from completing normally within a simulation. You must correct the conditions causing the Error Stop before the simulation can proceed.

### The Debug Toolbar

Most of the dynamic Simulation Windows have an additional toolbar called the Debug Toolbar that lets you debug using only mousing operati

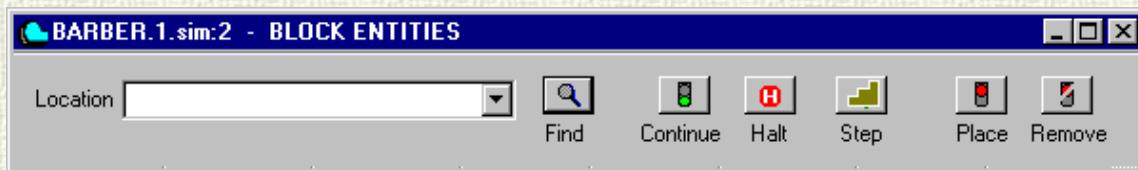


Figure 2♦19. The Blocks Window Debug Toolbar

The Blocks Window has the best Debug Toolbar because you can select a Block Icon with the mouse, then click on the Place button to place a Stop Condition that HALTs the simulation when a Transaction attempts to enter that Block. When the Block is selected, if you click on the Remove button, the original Stop Condition is removed.

Most of the other Simulation Windows have toolbars without the Place and Remove buttons. But they do have the rest. Use the Halt button to send a HALT Command to the Simulation Object. The Continue button resumes the simulation and the Step button causes the simulation to attempt to enter a single Block, and then HALT.

You can view the collection of Stop Conditions in the User Stops Snapshot.

### **SELECT Window / Simulation Snapshots / User Stops**

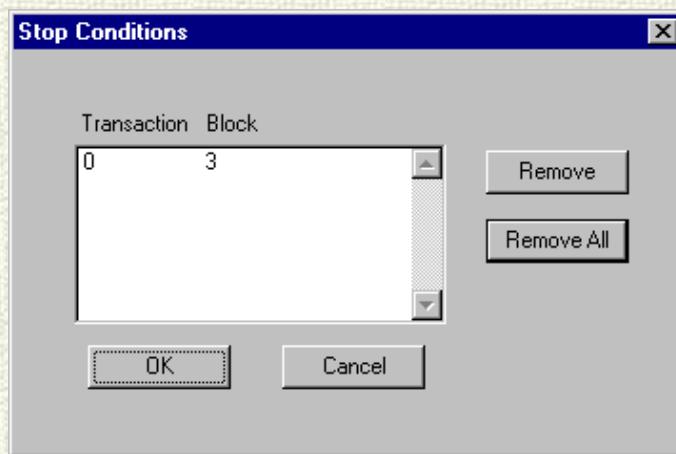


Figure 2♦20. The User Stops Snapshot

The User Stops Snapshot displays the set of all active Stop Conditions. A Stop Condition is defined by a Transaction Number and a Block Entity Number. Any attempt of a Transaction of the stated number to enter a Block of the stated number causes a HALT Command to be invoked. The value of 0 is special. Since no Transaction or Block has a number of 0, this value is used to indicate "Any". For example, the window above shows a single Stop Condition that occurs when any Transaction attempts to enter Block 3.

Stop Conditions are added to the list using the STOP Command or via a Debug Toolbar. You can remove any or all Stop Conditions by STOP „OFF Command, via a Debug Toolbar, or by selecting

and clicking the Remove or Remove All buttons in the User Stops Snapshot.

### PLUS Traces

When you are developing Model Objects that include your own PLUS Procedures, you should consider temporarily turning on the PLUS Trace in Page 2 of the Settings Notebook. Use **Edit / Settings** in the Main Window in order to do this. Then, PLUS Procedure invocations will be recorded by trace messages written to the Simulation's Journal Window.

You may find it useful to define a PLUS Procedure that takes one or more arguments but does nothing else. Then, you can insert invocations in your PLUS code that will cause traces of your User Variables to appear in the Journal Window. These can be easily removed after your PLUS Procedures are fully tested.

## Experimentation

After you have established credibility in your GPSS Model, you typically enter the experimental phase of the project. During this period, you run your model repeatedly to experiment with the effects of proposed actions on your simulated system. It is up to you to establish that the effects you have observed in your simulations are above the statistical noise level. The Analysis of Variance provided by the ANOVA library procedure can give you confidence that your results are due to more than just random variability.

The power of PLUS Experiments enables you to run complex experiments unattended, even exploring experimental response surfaces and collecting data for advanced statistical treatment. The Automatic Experiment Generators of GPSS World are at your disposal. Be sure to read Chapter 13 in this manual and study Lessons 19 and 20 in *The GPSS World Tutorial Manual* to get started with your experimentation.

### Batch Mode

There is a background Batch Mode of operation that is useful when a large amount of processing is needed which does not require interactions on your part. Let's assume that the GPSS World Module Directory is either the DOS current directory or is listed in the PATH variable. From a DOS Command Line type

**GPSSW FileSpec BATCH**

where *FileSpec* is the file name of a Model Object or a Simulation Object. If the Objects are located in directories other than the GPSS World Module Directory, you should give a fully qualified pathname.

In Batch Mode GPSS World opens the object, and attempts to continue processing in a minimized window. If the Object is a Model, GPSS World automatically issues a Create Simulation menu command to translate it. If the Object is a Simulation, GPSS World passes a CONTINUE Command to it. You should insert an EXIT 1 Command or an Exit(1) invocation to close the session and save all the files.

You should not use Batch Mode until your Model has been well tested. Any Syntax Errors or Error Stops will cause the Session to be stopped. You will then have to deal with them by opening the window and interacting with the Session.

### 2.3.3. Artificial Limits

#### Memory Request Limits

An arbitrary memory limit is used to protect your simulations from gross overstatements of memory requests. It is used to trigger an error condition when a memory request exceeds the limit. It applies to allocations of Table Entities, Matrix Entities, and Plot Windows.

You can change the limit if you need to by altering the limit in the Simulate page of the Model Settings Notebook. Select **Edit / Settings** and turn to the **Simulate** page. Enter the new limit in the **Max Memory Request** entry field, and CLICK on **OK**.

## Stack Size

The simulation stack is used for nested procedure calls. It is preallocated, and can cause a failure if it is not large enough when your simulation invokes PLUS Procedures to a deep nesting level. You can change the limit if you need to by altering the limit in the Simulate page of the Model Settings Notebook. Select **Edit / Settings** and turn to the **Simulate** page. Enter the new limit in the **Stack Size** entry field, and CLICK on **OK**.

## Circular References

An arbitrary limit is used to protect your simulations from circular references. If your simulation uses highly nested GPSS, or PLUS procedure calls, it is possible that you may need to increase this value. You can change it in the Simulate page of the Model Settings Notebook. Select **Edit / Settings** and turn to the **Simulate** page. Enter the new limit in the **Max Evaluation Depth** entry field, and CLICK on **OK**.

[\[Table of Contents\]](#)

# Chapter 3 - Model Statements

A GPSS World Model is a sequence of Model Statements. A Model Statement may be a GPSS Statement or a PLUS Procedure definition. A GPSS Statement is either a Command or a Block Statement.

PLUS is an acronym for the Programming Language Under Simulation, GPSS World's embedded programming language. PLUS Experiments are a kind of Procedure.

This chapter describes the use of Model Statements in the GPSS World simulation environment. It then discusses the general structure of GPSS Statements. Individual GPSS Commands are discussed in detail in Chapter 6, and Block Statements are in Chapter 7. Of the PLUS Statements, only PROCEDURE and EXPERIMENT are Model Statements. PLUS is described in detail in Chapter 8.

## 3.1. Using Model Statements

Before you can run a simulation, you must Create a Simulation Object. This is called the Initial Model Translation. It's easy. You just open a Model Window on the Model Object, and

CHOOSE    **Command / Create Simulation**

You will be told if any corrections need to be made, due to syntax errors.

When you have Translated a model successfully, the Simulation Object automatically gets a copy of the Translated Model, and executes any Commands embedded in the Model Files. You can now open any of the online windows upon the simulation. If you have put one or more START Commands into one of the Model Files, the Simulation Object executes them in order with the other Commands, without waiting to be told to.

Any Model Statement can be part of the Initial Model Translation, or can be sent to an existing simulation later, as an Interactive Statement.

### 3.1.1. Initial Model Translation

When a Model is Translated successfully, its Block Structure is created, a Procedure Registration List is created, a Command List is created, and the whole thing is sent to the Simulation Object.

When a model is Translated, the Block sequence is preserved, and separately, the initial Command sequence is preserved. When the Simulation Object receives a Command it either performs it immediately, or places it on the Command Queue to be dealt with in turn.

### 3.1.2. Interactive Statements

After the Initial Model Translation, you may send any Model Statement to the existing Simulation Object. These statements are called Interactive Statements. They are themselves Translated, and sent to the Simulation Object, which receives and executes them.

What happens depends on what kind of Model Statement is sent. PROCEDURE and EXPERIMENT Statements define or redefine a PLUS Procedure in the simulation. Commands are either executed immediately (SHOW or HALT), or are placed on the simulation's Command Queue, to be dealt with in sequence.

Interactive Block Statements are executed immediately in Manual Simulation Mode. This means that a temporary Block is created and the Active Transaction attempts to enter it. Then, the Block is destroyed.

### 3.1.3. Block Sequence

The Block structure of the simulation is set when the Model Object is Translated. It is not altered until the next Retranslation. During the Model Translation, as the Translator encounters each Block Statement, it appends a representation of it to the simulation's Block structure.

After a simulation is STARTed, it is the original sequence of Block Statements that determines the sequence of actions in the simulation. Unless the flow of Transactions is modified, each Active Transaction attempts to enter one Block after the other in the original sequence.

Block Statements received by the simulation after the original Model File was Translated, are NOT incorporated into the Block structure of the simulation. Instead, they cause a temporary block to be created, used, and destroyed in Manual Simulation Mode.

### 3.1.4. Command Sequence

Commands are either *Immediate* or *Queued*. Only HALT and SHOW are Immediate; all others are Queued. When the Simulation Object receives an Immediate Command it executes it right away. Queued Commands, on the other hand are placed in the Simulation's Command Queue. The Simulation Object executes one Queued Command after the other, occasionally interrupted by Immediate Commands.

A Command List is created by the Initial Model Translation and sent to the Simulation Object, just as Interactive Commands are sent. As the Translator encounters Commands during the Initial Model Translation, they are placed on the list. When the whole model has been Translated, the Command List is sent to the Simulation Object after the Block structure and the Procedure Registration List.

START, CONTINUE, and CONDUCT are Queued Commands. When the Simulation Object takes one off the Command Queue, the simulation runs until its Termination Count is nonpositive. More Commands may be waiting to be done after that.

The CONDUCT Command is used to run Experiments, which are a special kind of PLUS Procedure. Experiments are registered in a Simulation Object like any other PLUS Procedure, and they can have arguments passed to them by the CONDUCT Command.

The HALT Command is a special case. Not only is it an Immediate Command, but it has the effect of clearing off all Commands from the Command Queue. When you send a HALT Command to the Simulation Object, the simulation is brought to the Halted State, and all activity on behalf of the simulation comes to an end. You are then free to control the simulation as you please.

### 3.1.5. Procedure Sequence

During the Initial Model Translation, when the Translator encounters a Procedure or Experiment Statement, it puts the Translated Procedure on the Procedure Registration List. The Simulation Object registers all the user-defined Procedures when it receives the Translated model.

When the Simulation Object receives an Interactive Procedure Statement, it incorporates the Translated Procedure into the simulation's Procedure Library, replacing any existing Procedure of the same name.

The invocation of a Procedure occurs when an Expression with a Procedure Call, or a Procedure Call Statement, in a PLUS Procedure is executed. Invocations can be performed interactively, as well, by placing a Procedure invocation in the operand of a SHOW Command.

Experiments can only be invoked by CONDUCT Commands. Procedures invoked during an Experiment invocation have a special property: they may call the DoCommand library procedure. This is discussed in Chapter 8.

### 3.1.6. Saving Objects

All GPSS World Objects can be saved and opened in a later session. Models, Simulations, Reports, and Text Objects can all be modified and saved at any time. The File / Open Menu Command is available to bring the object back.

A Simulation can be saved at any point in its run for the purposes of debugging, demonstration, replay, or simply to save an intermediate state of the Simulation Object

The act of saving a Simulation or Experiment causes a HALT Command to be sent to the Simulation Object. In the case of an Experiment, you will need to save the state of the completed part of the Experiment in Global Variables and Global Matrices if you don't want to restart the Experiment from the beginning when you reopen it.

## 3.2. GPSS Statements

There are two kinds of GPSS Statements: Block Statements, which cause a GPSS Block Entity to be created, and Commands, which do not. Commands either define a non-Block Entity or cause an action to occur. PLUS Statements are not discussed here.

Unlike a PLUS Statement, a GPSS Statement must reside on a single line of text, consisting of up to 250 characters.

GPSS Statements are composed of parts called *fields*. A field is a variable number of printable characters terminated by white space or a delimiter. Although the exact composition of a Statement varies a little, in general, a Statement is laid out as follows: Line number (optional) | Label (optional) | Verb (required) | Operands (depend on verb) | Comment (optional).

Arbitrary Line numbers are retained for compatibility purposes. If used, they must begin in column 1. However, they are ignored by the Translator. A line number given in an error message is an absolute cardinal number, denoting the place of the line in the Model File, and is not a user-selected line number.

### 3.2.1. GPSS Commands

The commands are described in detail in Chapter 6. They are:

**BVARIABLE** - Define a Boolean Variable Entity.

**CLEAR** - Reset statistics and remove Transaction.

**CONDUCT** - Execute a registered PLUS Experiment.

**CONTINUE** - Resume the simulation.

**EQU** - Assign a value to a User Variable.

**EXIT** - End the GPSS World Session.

**FUNCTION** - Define a Function Entity.

**FVARIABLE** - Define an Fvariable Entity.

**HALT** - Stop the simulation and delete all Queued Commands.

**INCLUDE** - Read and Translate a secondary Model File.

**INITIAL** - Initialize or modify a Logicswitch, Savevalue, or Matrix Entity.

**INTEGRATE** - Automatically integrate a time differential in a User Variable.

**MATRIX** - Define a Matrix Entity.

**QTABLE** - Define a Qtable Entity.

**REPORT** - Set the name of the Report File or request an immediate report.

**RESET** - Reset the statistics of the simulation.

**RMULT** - Set the seeds of the first 7 Random Number Generators

**SHOW** - Evaluate and display Expression.

**START** - Set the Termination Count and begin a simulation.

**STEP** - Attempt a limited number of Block entries.

**STOP** - Set a stop condition based on Block entry attempts.

**STORAGE** - Define a Storage Entity.

**TABLE** - Define a Table Entity.

**VARIABLE** - Define a Variable Entity.

### 3.2.2. GPSS Block Statements

A Block Statement creates a GPSS Block. When a Transaction enters a Block Entity, a special action occurs which is determined by the nature of the Block. Block Statements may be inserted into a Model Object using the fullscreen text editor or using a Block Creation Dialog. It is also possible to use the text editor to place Block Statements into Text Objects that can be referred to by an INCLUDE Statement in the Text Object. The Block structure of the simulation is set when the Model Object is Translated. After a simulation is STARTed, it is the original sequence of Block Statements that determines the sequence of actions in the simulation. Unless the flow of Transactions is modified, each Active Transaction attempts to enter one Block after the other in the original sequence. Block Statements received by the simulation after the original Model File was Translated, are NOT incorporated into the Block structure of the simulation. Instead, they cause a temporary block to be created, used, and destroyed in Manual Simulation Mode, which is discussed above.

The GPSS Block Statements are described, in detail, in Chapter 7. They are:

**ADOPT** - Change Assembly Set.

**ADVANCE** - Place Transaction on Future Events Chain.

**ALTER** - Test and modify Transactions in a Group.

**ASSEMBLE** - Wait for and destroy related Transactions.

**ASSIGN** - Modify Transaction Parameter.

**BUFFER** - Place Transaction at end of the Current Events Chain.

**CLOSE** - End the Data Stream.

**COUNT** - Place count of entities into a Transaction Parameter.

**DEPART** - Decrement content of a Queue Entity.

**DISPLACE** - Change the Next Sequential Block of a Transaction.

**ENTER** - Occupy or wait for storage units in a Storage Entity.

**EXAMINE** - Test group membership.

**EXECUTE** - Perform action specified by a different Block.

**FAVAIL** - Change status of a Facility Entity to "available".

**FUNAVAIL** - Change status of a Facility Entity to "not available".

**GATE** - Test entity and modify Transaction flow.

**GATHER** - Wait for related Transactions.

**GENERATE** - Create Transaction and place on Future Events Chain.

**INDEX** - Modify Transaction Parameter.

**INTEGRATION** - Turn the integration of a User Variable On or Off.

**JOIN** - Place a member into a Numeric or Transaction Group.

**LEAVE** - Release storage units of a Storage Entity.

**LINK** - Move Transaction to Userchain Entity.

**LOGIC** - Modify Logicswitch Entity.

**LOOP** - Decrement Parameter, jump to different Block if result is non zero.

**MARK** - Place value of system clock into Transaction Parameter.

**MATCH** - Wait for related Transaction to reach conjugate MATCH Block.

**MSAVEVALUE** - Assign value to Matrix Entity element.

**OPEN** - Initialize a Data Stream.

**PLUS** - Evaluate PLUS Expression and save result in Parameter.

**PREEMPT** - Displace Facility owner.

**PRIORITY** - Modify Transaction priority.

**QUEUE** - Increment content of a Queue Entity.

**READ** - Bring the next line of data from a Data Stream.

**RELEASE** - Free Facility Entity.

**REMOVE** - Take a member out of Numeric or Transaction Group.

**RETURN** - Free Facility Entity.

**SAVAIL** - Change status of Storage Entity to "available".

**SAVEVALUE** - Assign a value to Savevalue Entity.

**SCAN** - Test Transaction group, place value in Parameter.

**SEEK** - Change the line pointer in a Data Stream.

**SEIZE** - Assume ownership of or wait for a Facility Entity.

**SELECT** - Place selected entity number into Transaction Parameter.

**SPLIT** - Create related Transaction.

**SUNAVAIL** - Change status of Storage Entity to "not available".

**TABULATE** - Update Table Entity.

**TERMINATE** - Destroy Transaction, decrement Termination Count.

**TEST** - Test arithmetic condition and modify Transaction flow.

**TRACE** - Set Trace Indicator of the Active Transaction.

**TRANSFER** - Move to specified Block.

**UNLINK** - Remove Transaction from Userchain Entity.

**UNTRACE** - Turn off Trace Indicator in the Active Transaction.

**WRITE** - Send a value to a Data Stream.

### 3.3. Fields

The parts of a GPSS Statement are called fields. GPSS Statements are of free form in the sense that you do not have to worry about lining up fields with column numbers, except for line numbers. The format is much the same for all statements, with some fields being optional for some verbs. In general, the fields of a statement are laid out left-to-right in the following sequence:

- ◆ Line number (optional). Ignored.
- ◆ Label (variable, depends on verb field).
- ◆ Verb (required).
- ◆ Operands, including any conditional operator (variable, depends on verb).
- ◆ Comment (optional).

### **3.3.1 Line Numbers**

Line numbers are ignored by GPSS World. They are permitted only in GPSS Statements, and if used, they must begin in column 1. They are retained for compatibility purposes.

### **3.3.2 Labels**

The label field allows you to name and refer to entities with names of your choosing. A label must obey the rules of name construction described later in this chapter. You may assign your own values to names with EQU Commands and PLUS Assignment Statements. This is useful when you want to experiment with several values of some variable. However, it is easiest to let GPSS World use its own unique internal numbers when you name entities and Transaction Parameters. If you choose to assign your own numbers to entity names, you should assign the value in an EQU statement before you define or refer to the entity by name. If you assign your own values to names, then you are responsible for avoiding reference conflicts.

When GPSS World encounters a new name, it assigns a unique integer to the name. These integers are distinct and contiguous starting at 10,000. You may use the same name to refer to different entity types. For example You may have a Storage Entity and a Table Entity both named Motorpool. However if you use the name Motorpool as a label in a second STORAGE Command, the old Storage Entity will be replaced with a new one. Permanent Block Entities, however, are not replaced in this manner.

Blocks are a special case. An interactive Block Statement is executed as a temporary Block in Manual Simulation Mode. If you need to be able to change the characteristics of a Block in the middle of a simulation, you can redefine the named values used in operands using EQU Statements. You can also use an EXECUTE Block and modify its operand interactively. Blocks cannot be inserted interactively. A change to the permanent Block structure requires a retranslation.

You may not use a number as a name in a Label Field. That would violate naming conventions. If you want to explicitly control the number of an entity, you can assign a value to a name using an EQU statement before using the name as a label.

You may not use an SNA as a Block Label.

### **3.3.3 Verbs**

The Verb Field follows the Label Field and requires a GPSS keyword. It must be one of the GPSS Block names or one of the Commands. A list of valid Verbs is available through the online help facility.

### **3.3.4 Operands**

Most Verbs are followed by one or more operator and operand fields. These fields depend on the verb. Some are required and some are optional. The Online Help facility is always available to show you which syntax is acceptable. When you want to finish or skip an operand, you should press a delimiter key. Traditionally, operands are separated by commas. If all required information in a statement has been provided, you may skip optional fields.

The forms you may use in any given Operand Field are listed with the specific Statement descriptions in Chapters 6 and 7. This information is available as online help, as well.

### **3.3.5 Comments**

There are two kinds of comment fields. First, by starting a statement with ; or \*, the whole statement is considered to be a comment. Second, an optional comment field follows the operand field of any statement. Function followers and statements with expression fields do not have comment fields.

When all required operands of a statement have been completed, you can press ; to begin a comment. At this time you can enter a comment of both upper and lower case ASCII characters.

## 3.4. Expressions

### 3.4.1. Expressions in GPSS Statements

Expressions in GPSS World are defined as part of the PLUS Language. You can use them in GPSS Statement Operands and in PLUS Procedures.

Some Block operands permit the use of **parenthesized** PLUS Expressions. The list of acceptable syntactic variables will then include *ParenthesizedExpression* as one or the operand's alternative forms. Some Commands do not need to use the outer parentheses, but if you always parenthesize PLUS Expressions used in GPSS Statements, you will be safe.

It's easy to go between PLUS and GPSS. Most GPSS Statement Operands allow you to use a PLUS Expression. This means that you can provide a simple factor, a small calculation, or a PLUS Procedure invocation right there in the Operand. On the other hand, by using Expressions containing SNAs, you can access GPSS entity state variables, as well as global User Variables, from within PLUS Procedures.

Section 8.3.2 of this manual tells you how to build PLUS Expressions. All the building blocks except System Numeric Attributes are discussed there. We discuss SNAs in the next section.

### 3.4.2. System Numerical Attributes

System Numerical Attributes, or SNAs, are simulation "state variables" that are available for use through a simulation. They return numeric or string values, and may be used in GPSS Statement operands and in Expressions.

Some SNAs are maintained automatically, and others require a calculation when they are called. For example, the accumulated count of entries into a Block is available as an SNA. There are over 50 SNA classes, each of which will contribute to the power of your simulations. It is wise to become familiar with all of them. Many SNAs, such as PR, are evaluated on behalf of the active Transaction. This means that some attribute of a Transaction is necessary in order to evaluate the SNA. Such SNAs cannot be evaluated unless there is an Active Transaction. For example, you cannot refer to Transaction Parameters in the operands of GENERATE Blocks because when a Transaction enters a GENERATE Block for the first time, it has no Parameters. Other SNAs, such as the system clock C1, can be evaluated without referring to a Transaction.

### SNA Entity Specifiers

Most SNAs may be specified in one of many forms, beginning with the SNA class. For example the SNA W22 returns the number of Transactions waiting in Block number 22. The Block identifier in this case is the number 22, but there are several other possibilities. This SNA could have been specified using any of the following entity specifiers:

- ◆ **Wj** - where *j* is a positive integer, the number of the Block in the simulation.
- ◆ **W\$Name** - where *Name* is the location of the desired Block.
- ◆ **W\*j** - where *j* is a positive integer, the number of the Parameter of the active Transaction which contains the number of the desired Block. This is indirect addressing.
- ◆ **W\*Name** - where *Name* is the name of the Parameter of the Active Transaction which contains the number of the desired Block. This is indirect addressing.
- ◆ **W\$\$Name** - where *Name* is the name of the Parameter of the Active Transaction which contains the number of the desired Block. The \$ is not needed and is used only as a separator. Essentially, this is identical to **W\*Name**. This is indirect addressing.
- ◆ **W\*Parameter** - denotes that either the **W\*j**, the **W\*Name**, or the **W\$\$Name** may be used.

The Matrix Entity SNA class MX is a special case. It may contain up to 3 indirect addresses. For example the SNA MX\*Sales(\*Partnumber,\*January) locates the Matrix Entity whose number is in the Transaction parameter named Sales, and then retrieves the element whose row and column numbers are in the Parameters named Partnumber and January, respectively. Normally, the Active Transaction

would have already passed through three ASSIGN blocks initializing the Parameters Sales, Partnumber, and January with the numbers or name values desired.

Some SNA classes, such as A1, AC1, C1, M1, MP, PR, and TG1, are "atomic" SNAs. Atomic SNAs are complete and do not need an entity name or number to complete the evaluation of the SNA

When you are in any operand field, the valid SNA forms are available in an online help message.

## SNAs Available

The following SNAs are available for use in the operand fields and Expressions of commands and statements. In all cases *Entnum* must be replaced by any entity specifier. The entity specifier could be a name (preceded by a \$ separator) or number, or for indirect addressing, it could be an asterisk, \*, followed by a name or number. The formal rules for building operands are given in the Appendix.

Here is a complete list of SNAs available in GPSS World:

- ◆ **A1** - Assembly Set of the Active Transaction. Integer value.
- ◆ **AC1** - Value of absolute system clock. Simulated time since last CLEAR. Real value.
- ◆ **BVEntnum** - Result of evaluating Boolean Variable Entity *Entnum*. Real value.
- ◆ **C1** - Value of relative system clock. Simulated time since last RESET. Real value.
- ◆ **CAEntnum** - Average Userchain content. The time weighted average number of chained Transactions for Userchain *Entnum*. Real value.
- ◆ **CCEntnum** - Total Userchain entries. The count of all Transactions chained to Userchain *Entnum*. Integer value.
- ◆ **CHEntnum** - Current Userchain content. The current number of Transactions chained to Userchain *Entnum*. Integer value.
- ◆ **CMEntnum** - Maximum Userchain content. The maximum number of Transactions chained to Userchain *Entnum*. The "high water mark". Integer value.
- ◆ **CTEntnum** - Average Userchain residence time. The average duration of Transactions at Userchain *Entnum*. Real value.
- ◆ **FEntnum** - Facility busy. If Facility Entity *Entnum* is currently busy, *FEntnum* returns 1. Otherwise *FEntnum* returns 0. Integer value.
- ◆ **FCEntnum** - Facility capture count. The number of times Facility Entity *Entnum* has been SEIZED or PREEMPTed by a Transaction. Integer value.
- ◆ **FIEntnum** - Facility *Entnum* interrupted. If Facility Entity *Entnum* is currently preempted, *FIEntnum* returns 1. Otherwise *FIEntnum* returns 0. Integer value.
- ◆ **FNEntnum** - Function. Result of evaluating Function Entity *Entnum*. Real value.
- ◆ **FREntnum** - Facility utilization. The fraction of time Facility Entity *Entnum* has been busy. *FREntnum* is expressed in parts-per-thousand and therefore returns a value 0-1000, inclusive. May be nonintegral. Real value.
- ◆ **FTEntnum** - Average Facility holding time. The average time Facility Entity *Entnum* is owned by a capturing Transaction. Real value.
- ◆ **FVEntnum** - Facility in available state. *FVEntnum* returns 1 if Facility Entity *Entnum* is in the available state, 0 otherwise. Integer value.
- ◆ **GNEntnum** - Numeric Group count. *GNEntnum* returns the membership count of Numeric Group *Entnum*. Integer value.
- ◆ **GTEntnum** - Transaction Group count. *GTEntnum* returns the membership count of Transaction Group *Entnum*. Integer value.
- ◆ **LSEntnum** - Logicswitch set. *LSEntnum* returns 1 if Logicswitch Entity is in the "set" state, 0 otherwise. Integer value.

◆ **M<sub>B</sub>E<sub>n</sub>t<sub>m</sub>** - Match at Block. M<sub>B</sub>E<sub>n</sub>t<sub>m</sub> returns a 1 if there is a Transaction at Block Ent<sub>m</sub> which is in the same Assembly Set as the Active Transaction. M<sub>B</sub>E<sub>n</sub>t<sub>m</sub> returns a 0 otherwise. Integer value.

◆ **M<sub>P</sub>P<sub>a</sub>r<sub>e</sub>am<sub>e</sub>r** - Transit Time, Parameter. Current absolute system clock value minus value in Parameter Parameter. Real value.

◆ **M<sub>X</sub>E<sub>n</sub>t<sub>m,n</sub>** - Matrix Savevalue. The value in row m, column n of Matrix Entity Ent<sub>m</sub> is returned. In Matrices of more than 2 dimensions, all other indices are assumed to be 1. Unlike MX class SNAs, PLUS Expressions can refer to any element of a higher dimension matrix.

◆ **M<sub>1</sub>** - Transit time. M<sub>1</sub> returns the absolute system clock minus the "Mark Time" of the Transaction. Real value.

◆ **N<sub>E</sub>n<sub>t</sub><sub>m</sub>** - Block entry count. The total number of Transactions which have entered Block Ent<sub>m</sub> is returned. Integer value.

◆ **P<sub>P</sub>ar<sub>e</sub>am<sub>e</sub>r or \*P<sub>a</sub>ram<sub>e</sub>r** - Parameter value. P<sub>P</sub>ar<sub>e</sub>am<sub>e</sub>r or \*P<sub>a</sub>ram<sub>e</sub>r returns the value of Parameter Parameter of the Active Transaction. Integer, real, or string value.

Indirect addressing uses the notation SNA\*Parameter.

◆ **P<sub>R</sub>** - Transaction priority. The value of the priority of the Active Transaction. Integer value.

◆ **Q<sub>E</sub>n<sub>t</sub><sub>m</sub>** - Current Queue content. The current count value of Queue Entity Ent<sub>m</sub>. Integer value.

◆ **Q<sub>A</sub>E<sub>n</sub>t<sub>m</sub>** - Average Queue content. The time weighted average count for Queue Entity Ent<sub>m</sub>. Real value.

◆ **Q<sub>C</sub>E<sub>n</sub>t<sub>m</sub>** - Total Queue entries. The sum of all entry counts for Queue Entity Ent<sub>m</sub>. Integer value.

◆ **Q<sub>M</sub>E<sub>n</sub>t<sub>m</sub>** - Maximum Queue content. The maximum count of Queue Entity Ent<sub>m</sub>. The "high water mark". Integer value.

◆ **Q<sub>T</sub>E<sub>n</sub>t<sub>m</sub>** - Average Queue residence time. The time weighted average of the count for Queue Entity Ent<sub>m</sub>. Real value.

◆ **Q<sub>X</sub>E<sub>n</sub>t<sub>m</sub>** - Average Queue residence time excluding zero entries. The time weighted average of the count for Queue Entity Ent<sub>m</sub> not counting entries with a zero residence time. Real value.

◆ **Q<sub>Z</sub>E<sub>n</sub>t<sub>m</sub>** - Queue zero entry count. The number of entries of Queue Entity Ent<sub>m</sub> with a zero residence time. Integer value.

◆ **R<sub>E</sub>n<sub>t</sub><sub>m</sub>** - Available storage capacity. The storage content (or "tokens") available for use by entering Transactions at Storage Entity Ent<sub>m</sub>. Integer value.

◆ **R<sub>N</sub>E<sub>n</sub>t<sub>m</sub>** - Random number. R<sub>N</sub>E<sub>n</sub>t<sub>m</sub> returns a random integer 0-999 from Random Number Generator Ent<sub>m</sub>. Integer value.

◆ **S<sub>E</sub>n<sub>t</sub><sub>m</sub>** - Storage in use. S<sub>E</sub>n<sub>t</sub><sub>m</sub> returns the amount of storage content (or "tokens") currently in use by entering Transactions at Storage Entity Ent<sub>m</sub>. Integer value.

◆ **S<sub>A</sub>E<sub>n</sub>t<sub>m</sub>** - Average storage in use. S<sub>A</sub>E<sub>n</sub>t<sub>m</sub> returns the time weighted average of storage capacity (or "tokens") in use at Storage Entity Ent<sub>m</sub>. Real value.

◆ **S<sub>C</sub>E<sub>n</sub>t<sub>m</sub>** - Storage use count. Total number of storage units that have been acquired from Storage Entity Ent<sub>m</sub>. Integer value.

◆ **S<sub>E</sub>E<sub>n</sub>t<sub>m</sub>** - Storage empty. S<sub>E</sub>E<sub>n</sub>t<sub>m</sub> returns 1 if Storage Entity Ent<sub>m</sub> is completely available, 0 otherwise. Integer value.

◆ **S<sub>F</sub>E<sub>n</sub>t<sub>m</sub>** - Storage full. S<sub>F</sub>E<sub>n</sub>t<sub>m</sub> returns 1 if Storage Entity Ent<sub>m</sub> is completely used, 0 otherwise. Integer value.

◆ **S<sub>R</sub>E<sub>n</sub>t<sub>m</sub>** - Storage utilization. The fraction of total usage represented by the average storage in use at Storage Entity Ent<sub>m</sub>. S<sub>R</sub>E<sub>n</sub>t<sub>m</sub> is expressed in parts-per-

thousand and therefore returns a value 0-1000, inclusive. May be nonintegral. Real value.

◆ **SME<sub>Entnum</sub>** - Maximum storage in use at Storage Entity *Entnum*. The "high water mark". Integer value.

◆ **ST<sub>Entnum</sub>** - Average holding time per unit at Storage Entity *Entnum*. Real value.

◆ **SVE<sub>Entnum</sub>** - Storage in available state. *SVE<sub>Entnum</sub>* returns 1 if Storage Entity *Entnum* is in the available state, 0 otherwise. Integer value.

◆ **TBE<sub>Entnum</sub>** - Nonweighted average of entries in Table Entity *Entnum*. Real value.

◆ **TCE<sub>Entnum</sub>** - Count of nonweighted table entries in Table Entity *Entnum*. Integer value.

◆ **TDE<sub>Entnum</sub>** - Standard deviation of nonweighted table entries in Table Entity *Entnum*. Real value.

◆ **TG1** - Remaining Termination Count. TG1 returns the count that is decremented by TERMINATE Blocks with a positive A operand. This value is initialized by START Statements and indicates completion of the simulation when it is 0. Integer value.

◆ **VE<sub>Entnum</sub>** - Result of evaluating arithmetic or floating point Variable Entity *Entnum*. Real value.

◆ **WE<sub>Entnum</sub>** - Current Block count. The current number of Transactions in Block Entity *Entnum*. Integer value.

◆ **XE<sub>Entnum</sub>** - Savevalue. The value of Savevalue Entity *Entnum* is returned. Integer, real, or string value.

◆ **XN1** - Active Transaction number. The Transaction number of the Active Transaction is returned. Integer value.

◆ **Z1** - Free memory. Value returned by the Operating System. Integer value.

Special rules apply when a simulation is run in GPSS/PC Compatibility Mode. This are discussed more fully below.

## Operators

Operators are used to combine data elements in Expressions. Data are coerced if a specific type is required by the operator. For example, if a numeric operation is to be performed on a variable with a string value, the numeric equivalent derived from the characters in the string is used.

A Setting is available in the Simulation Page of the Settings Notebook which switches the roles of the [\*] operator and the [#] operator.

The operators used in GPSS World are:

^ Exponentiation. A^B returns A raised to the power of B.

# (or \*) Multiplication. A # B returns the numeric product of A and B.

/ Division. A / B returns the quotient of A divided by B.

\ Integer Division. A \ B returns the result of integer division of A by B.

@ Integer Remainder. A @ B returns the modulo division of A by B.

- Subtraction. A - B returns the difference of A and B.

+ Addition. A + B return the sum of A and B.

>= ◆GE◆ Greater Than or Equal To. A >= B returns 1 if A is numerically greater than or equal to B, 0 otherwise.

**<= ♦LE♦** Less Than or Equal To. A <= B returns 1 if A is numerically less than or equal to B, 0 otherwise.

**> ♦G♦** Greater Than. A > B returns 1 if A is numerically greater than B, 0 otherwise.

**< ♦L♦** Less Than. A < B returns 1 if A is numerically less than B, 0 otherwise.

**= ♦E♦** Equal A = B returns 1 if A is numerically equal to B, 0 otherwise.

**!= ♦NE♦** Not Equal. A != B returns 1 if A is numerically different from B, 0 otherwise.

**& ♦AND♦** Logical And. A & B returns 1 if and only if both A and B are non zero, 0 otherwise.

**| ♦OR♦** Logical Or. A ♦OR♦ B returns 1 if A or B or both are non zero, 0 otherwise.

### Precedence

When Expressions are evaluated, the precedence of operators determines the order of operations. An operator with a higher precedence is evaluated before an operator with a lower precedence. The precedence of operators is as follows, from highest to lowest:

**^** Exponentiation

**# (or \*) / \** Multiplication, Division, Integer Division

**@** Integer Remainder

**- +** Addition, Subtraction

**>= <= > <** Comparison Operators

**= !=** Equal, Not Equal

**&** Logical And

**|** Logical Or

## Indirect Addressing

All entities have positive entity numbers. When you refer to an entity by name, the value of the name is matched with the entity number of the entity. Indirect addressing means that the entity number is in a Transaction parameter. To use indirect addressing, you must have an asterisk, [\*] (or possibly [#] – see Settings), in the operand. For example, the SNA Q\*2 uses the value of parameter 2 as the Queue Entity number. This SNA returns the current content of that Queue Entity.

You must move an entity number (or name value) into a Transaction parameter before the SNA is evaluated.

Indirect addressing uses the notation SNA \*Parameter. As another example, to SEIZE the Facility whose number is the value returned by the function whose number is in a parameter named Tellerselector, you would type **SEIZE FN\*Tellerselector**.

### 3.4.3. Expressions in GPSS/PC Compatibility

In GPSS World, intermediate results and SNAs are never truncated unless you explicitly do so. However, for compatibility purposes, intermediate result truncations are retained in GPSS/PC Compatibility Mode. When simulations are run this way, SNAs always return truncated integers (unless used as Function Modifiers), and the intermediate results obtained while evaluating Bvariable Entities and Variable Entities, are truncated.

The use of GPSS/PC Compatibility Mode is discussed in Chapter 1.

In Compatibility Mode, when an Expression is used to define an SNA, its result is always truncated to an integer. You should choose the time units of the simulation so that these truncations are not

significant.

In Compatibility Mode, the evaluation of an Expression is affected by the Command it is used in.

- ◆ SHOW performs all operations as double precision floating point numbers and display the result in the data window as a double precision floating point number.
- ◆ VARIABLE creates a GPSS arithmetic variable which, when evaluated by an SNA call, performs all operations as double precision floating point numbers, then truncates all intermediate results before proceeding to the next operation. Finally it truncates and returns the overall result.
- ◆ FVARIABLE creates a GPSS "floating point" variable which, when evaluated by an SNA call, performs all operations as double precision floating point numbers, then truncates the result. Fractional values may be considered by an appropriate choice of units, such as "thousandths". Then an SNA value of 500 is interpreted as the number 1/2.
- ◆ BVARIABLE creates a GPSS Boolean variable which, when evaluated by an SNA call, performs all operations as double precision floating point numbers, then returns 1 if the result is non zero, 0 otherwise.
- ◆ FUNCTION Commands, of types other than C, evaluate the argument and select a list member based on the result to determine the final value, which is truncated.
- ◆ C Type FUNCTION evaluation begins with the evaluation of the argument. The result is always an integer, and it is used to identify the line segment of the Function. The argument is then used in a double precision linear interpolation to arrive at the double precision result of the Function. If the argument is an RN class SNA, a random fraction between 0 and .999999, inclusively, is used. If the Function is not used as a "Function Modifier" the final result is truncated.
- ◆ SNAs, except continuous Functions used as Function Modifiers, return an integer.

## 3.5. Names

You can create names to refer to Blocks, User Variables, or other GPSS Entities. Names must begin with an alphabetic character, and may contain up to 200 alphabetic and numeric characters, and underscores [ \_ ].

Your primary tasks in selecting a name are, first, to avoid using a Keyword, System Numeric Attribute, or System Numeric Attribute Class, and second, to use something meaningful that can be remembered.

A name may not be a verb or partial verb, a keyword, or a valid SNA. If you are unsure of which names are illegal, you will be safe if you include an underscore [ \_ ] in the names you create. Names beginning with at least 3 letters and then a digit are also safe. In any case, GPSS World will not let you create an invalid name.

### 3.5.1. Labels

Names used in the Label field of a GPSS Statement are used to refer to GPSS Entities, and are called Entity Labels, or just Labels. Block Labels are also called Locations.

Locations are assigned values corresponding to the Block Entity number. You should not use a Block location name for any other purpose. GPSS World will cause an Error Stop to occur if you attempt to do so. This eliminates the possibility of inadvertently altering which Block is associated with a label.

When a Simulation Object receives a name from a Session, it assigns a unique numeric value to the name. These system-defined numbers start at 10,000. You can assign your own value, of any data type, later.

If you want to force a named entity to have an entity number of your choosing, you must use the name in an EQU Statement before you define (or reference) the entity by name. It is then up to you to avoid different names inadvertently referring to the same entity. You may use a single name to refer to one entity of each entity type. For example, the SNAs Q\$BARBER and F\$BARBER refer to different entities. The first refers to a Queue Entity, the second to a Facility Entity. There is no confusion when the same name refers to different entity types.

When an entity is created with a label, it takes the current value of the label to keep as its permanent Entity Number. Even if the Named Value is later changed, the Entity Number of the created entity is not.

If it can, the Simulation Object will create a GPSS entity when it first encounters a name reference to that entity type. However, entities that cannot be created without additional information, such as Storage Entities, must be defined by GPSS Commands before they are referenced. The entities which must be defined before they are referenced are listed in Chapter 4.

### 3.5.2. User Variables

In addition to using Names as Labels, you can use them to represent your own values. Such variables are called User Variables, and are created by their occurrence in an EQU Command or in a PLUS Assignment Statement. They serve the traditional purpose of programming variables. A user created name which is first assigned a value with an EQU Command, but later appears as an Entity Label, is still known as a Label.

User Variables can be used to hold numeric or string values during a simulation. They can be referenced in PLUS Expressions, and can be altered by an EQU Command or an assignment Statement in a PLUS Procedure. In addition, they can be automatically updated by integration. This is discussed in Chapter 4, under the section [Continuous Simulation](#).

### 3.5.3. The Scope of Names

Except for temporary names, all user-created names are known throughout the whole model. The exceptions are those names appearing in the TEMPORARY or the TEMPORARY MATRIX Statements of a PLUS Procedure. New instances of temporary Named Values or Matrices are created when the PLUS Procedure is invoked and deleted when the PLUS Procedure exits. These names override any other like-named items in the Model during execution of the PLUS Procedure. Other names used in PLUS Procedures refer to entities known globally throughout the model.

Within an Experiment, you should place the results of simulations into Global Variables and Global Matrices in order to preserve them across simulations. For example, the ANOVA library procedure requires a Global Matrix Entity to be passed as an argument. Also, the DoCommand library procedure is available during an Experiment. Any strings passed to DoCommand should not include the names of TEMPORARY and TEMPORARY MATRIX variables or of procedure arguments, because the string is Translated in a Global Scope where temporary variables are not recognized.

## 3.6. Numbers

Numbers may be stored internally in any of three data types, integer, real, and string. Most variables can take on any of these three data types. Since numeric operators coerce their operands into suitable form, all three data types may be used in Expressions,

Integers are stored as 32 bit two's complement numbers. If an integer overflows it is converted into a real value.

Real values are stored as 64 bit double precision floating point numbers. The exponent can range from -308 to 308, whereas the precision is limited to approximately 15 decimal digits.

Strings are stored as an array of ASCII characters of any length, limited by the **Max Memory Request** in the Simulation Page of the Simulation Object's Settings. You can view or change the Settings by choosing the **Edit / Settings** menu item.

System Numerical Attributes now return values that may be either integer, real, or string. You must refer to the definition of the SNA given above to determine the possible data types returned.

Similarly, the system clock is a numeric value which may be either integer or real. This eliminates the need to use large time values to assure fine time granularity.

GPSS function entities are evaluated in double precision form, and the Y values in function follower Statements are, too. This limits precision to approximately 15 decimal digits for the Y value.

SNAs involving GPSS Variable Entities (all types) and those requiring a standard deviation or a division are calculated in double precision floating point format. As a result, these SNAs can experience numeric overflow and/or underflow. Since Expressions are evaluated as double precision floating point numbers, intermediate values are limited to 15 decimal digits of precision and 307 decimal digits of magnitude.

If an overflow occurs during the evaluation of an Expression, an Error Stop occurs. Arithmetic exceptions may occur when divisions, standard deviations, or logarithms are evaluated. An invalid argument to the library functions can also cause an Error Stop.

In the printing of report statistics, when a number in the report is too large for its space in the report, the format of the report is disrupted, but the correct value is printed and the report continues.

## 3.7. Using Strings

Strings are arrays of ASCII characters. Nearly any variable can take on a string value. Arithmetic operators will coerce string values to their numeric equivalent before executing their operations.

String constants are enclosed in double quote marks. When you input a string, such as the file specification in an INCLUDE Command, or a REPORT Command, you must enclose the sequence of ASCII characters in double quotes. Similarly, string values written in Standard Reports and in response to SHOW commands are shown enclosed in double quotes.

To insert a "string within a string" you should use 4 additional double quote characters around the inner string, which is to be sandwiched by 2 pairs of double quotes. When two double quotes together are encountered by the Translator, a single quote is placed in the target string.

Strings are used when you write out simulation results to a Result File. Strings may also be used to format your own ad hoc simulation reports. The Data Stream Blocks OPEN, CLOSE, READ WRITE, SEEK are available for these purposes, as are the PLUS library procedures of the same names. Strings are also used by the DoCommand library procedure during Experiments. Another use for strings is to create a trace for input to an animation post-processor.

You can create and manipulate strings using the routines in the PLUS Procedure Library. If you want to combine strings, use the Catenate library procedure to join 2, or use the more flexible PolyCatenate to join any number of strings. These, and others, are discussed in detail in Chapter 8.

[\[Table of Contents\]](#)

# Chapter 4 - GPSS Entities

GPSS is built around several elementary abstractions called entities. In order for you to be able to create complex simulation, you must acquire an understanding of these entities and of the rules by which they may be manipulated. GPSS entities are abstract objects that exist in a simulation. If you prefer a more concrete notion, you may think of a GPSS Entity as a set of numbers in the memory of your computer. The collection of all the entities is called the simulation. The most prominent entity types are Transactions and Blocks, because simulations, to a large extent, consist of many Transactions moving from one Block into the next. Transactions are the only entity types that can be deleted from the simulation. In all, there are a dozen or so entity types, and a simulation may contain many instances of any entity type. To be effective in creating simulations you must understand the properties of each of the GPSS entities and how to use GPSS Blocks in order to cause interactions among the entities.

GPSS entities are numbered. When you use a name to refer to an entity, the integer value associated with the name is used to find the entity. However you do not normally assign these integer values to names, although the EQU statement enables you to do so. GPSS World normally assigns a unique value greater than or equal to 10,000.

Most GPSS entities are created automatically when needed. For example, a reference to a Facility Entity using the name Barber will cause a Facility to be created if none existed before. This convenience can sometimes cause your simulation to use a lot of virtual memory due to a bug in your GPSS model. Your simulations can be extremely large. The use of virtual memory provides for simulations taking up to half a gigabyte (512 Megabytes).

Some entities must be specifically declared before they can be used. Generally these have an attribute, such as size, which must be made known to the Simulation Object. The name in the label field, called an Entity Label, is then used to refer to the entity.

The following entities must be declared before they can be used:

- ◆ Storage entities must be declared in STORAGE Statements.
- ◆ Arithmetic Variables must be declared in VARIABLE Statements.
- ◆ "Floating point" Variables must be declared in FVARIABLE Statements.
- ◆ Boolean Variables must be declared in BVARIABLE Statements.
- ◆ Matrices must be declared in MATRIX Statements, or Temporary Matrix PLUS Declarations.
- ◆ Tables must be declared in TABLE Statements.
- ◆ Qtables must be declared in QTABLE Statements.
- ◆ Functions must be declared in FUNCTION and Function Follower Statements.
- ◆ Transaction Parameters must be declared in ASSIGN, MARK, READ, SELECT, SPLIT, COUNT or TRANSFER SUB Blocks before they are referenced.

Some entity types bear one or more special relationships to Transactions. Storage and Facility Entities can be partially or wholly owned by Transactions. Other Transactions may then come to rest in the model while they wait for ownership. It is up to you to assure that ownership is eventually released by the original owners, by causing them to enter RELEASE, RETURN, or LEAVE Blocks. Otherwise, the simulation may not complete successfully. Some Blocks such as PREEMPT and FUNAVAIL have options that divert the path of the owning Transaction. However, it is still up to you to provide for the release of the owned entity.

## 4.1. Transaction Entities

Transactions move from Block to Block in a simulation in a manner which represents the real-world system you are modeling. Once a Transaction begins to move in the simulation, it continues to enter Blocks as long as it can. During a simulation, the Transaction which is attempting to move from Block to Block is called the *Active Transaction*. If a Transaction fails to find favorable conditions when it attempts to enter a Block, it may come to rest. Then, another Transaction is chosen to begin to move through the simulation until it, in turn, comes to rest.

Transactions are numbered sequentially throughout a session starting with 1. A CLEAR statement begins the numbering of Transactions at 1 again.

The behavior of a Transaction is determined somewhat by several state variables called Transaction Attributes. The important attributes of Transactions are:

- ◆ Parameters - Transaction Parameters are a set of values associated with a Transaction. Each Transaction may have any number of Parameters. Each Parameter has a Parameter Number, by which it is referenced, and a value. The Parameter Number is a positive integer. The value of any Parameter of the active Transaction may be returned by the SNA PParameter where *Parameter* is the name or number of the Parameter. If a Block operand specifies Parameter name or number as the desired value, P\$Parameter or PParameter should not be used, use only the Parameter Name or Number by itself.

For efficiency, you can allow GPSS World to directly access Parameters by using Parameter Blocks in your simulation. A Parameter Block is an array of contiguous Parameters that are allocated and freed as a single segment of memory. When you create the first Parameter, whose Parameter Number is in range, all the others Parameters in the segment are allocated as well. When you reference a Parameter in a Parameter Block, the Simulation Object can go directly to it, rather than having to check each Parameter individually. This can save a lot of time. All you have to do to use Parameter Blocks is to declare them in the Simulate Page of the Model Settings Notebook. This is discussed in Chapter 2.

Parameters are used in GPSS indirect addressing. When the number of the desired GPSS entity is kept as a Parameter value of the Active Transaction, the name or number of the desired GPSS entity specifier can be replaced with an indirect reference to the Transaction Parameter. There is a more detailed description of this in Section 3.4.

Transaction Parameters must be created and assigned values before they can be referenced. ASSIGN, MARK, and TRANSFER SUB, SELECT, SPLIT and COUNT Blocks create a Transaction Parameter if one does not exist.

◆ Priority - the priority of a Transaction determines the preference it receives when it and other Transactions are waiting for the same resource. Transactions with higher priority values receive preference. The most important priority queues in the simulation are the Current Events Chain, Facility Delay Chains, and Storage Delay Chains. The Future Events Chain is not a priority chain. These structures are explained in more detail in Chapter 9. The effect of priority is that a Transaction will be chosen ahead of lower priority Transactions when a new Active Transaction, or a new Facility or Storage owner must be chosen. Transactions within a priority are usually scheduled first come, first served.

◆ Mark Time - The absolute clock time that the Transaction first entered the simulation or entered a MARK Block with no A operand.

◆ Assembly Set - A positive integer kept internally in each Transaction. Assembly Sets are used to synchronize Transaction in ASSEMBLE, GATHER, and MATCH Blocks. When a Transaction is created by a GENERATE Block, its Assembly Set is set equal to its Transaction Number. When a Transaction is created by a SPLIT Block, its Assembly Set is set equal to that of the parent Transaction. A Transaction can modify its Assembly Set by entering an ADOPT Block.

◆ Delay Indicator - A flag kept in each Transaction that is set by Block entry refusal, and is reset by entry into TRANSFER SIM Block. It is used by TRANSFER SIM Blocks to redirect Transactions.

◆ Trace indicator - A flag kept in each Transaction that causes a trace message to be generated each time the Transaction enters a Block. The Trace Indicator is set by a TRACE Block and reset by an UNTRACE Block.

◆ Current Block - The Entity Number of the Block which contains the Transaction.

◆ Next Block - The Entity Number of the Block which the Transaction will attempt to enter next.

◆ Chains - The state of a Transaction is determined to some degree by the chains on which it resides. Chapter 9 contains a more detailed description of Transaction chains. A Transaction is said to be in exactly one of several states:

◆ ACTIVE - The Transaction is the highest priority Transaction on the Current Events Chain.

◆ SUSPENDED - The Transaction is waiting on the Future Events Chain or the Current Events Chain to become the active Transaction.

◆ PASSIVE - The Transaction has come to rest in the simulation on a User Chain, Delay Chain, or Pending Chain.

◆ TERMINATED - The Transaction has been destroyed and no longer exists in the simulation.

In addition, there is another state which is not mutually exclusive with the others:

◆ PREEMPTED - The Transaction has been preempted at a Facility and is on one or more interrupt Chains.

## The Active Transaction

At any instant during the discrete phase of a simulation, one specific Transaction is attempting to enter a new GPSS Block. That Transaction is called the Active Transaction. Generally, the active Transaction moves as far as it can through the simulation. When it cannot move further, another Transaction is chosen to be the Active Transaction. There can be no more than one Active Transaction.

More specifically, the Active Transaction is the highest priority Transaction on the Current Events Chain when the last Block scheduling occurred. This is described in more detail in Chapter 9.

## Related SNAs

The SNAs associated with Transactions are:

◆ A1 - Assembly Set. A1 returns the Assembly Set of the Active Transaction.

◆ MBE<sub>n</sub>t<sub>m</sub> - Match at Block. MBE<sub>n</sub>t<sub>m</sub> returns a 1 if there is a Transaction at Block *n* which is in the same Assembly Set as the Active Transaction. MBE<sub>n</sub>t<sub>m</sub> returns a 0 otherwise. This SNA class should not be used in a Refuse Mode GATE or TEST Block condition test, you should use MATCH Blocks instead.

◆ MPParameter - Transit Time, Parameter. Current absolute system clock value minus value in Transaction Parameter *Parameter*.

◆ M1 - Transit Time. M1 returns the absolute system clock minus the "Mark Time" of the Transaction.

◆ PParameter or \*Parameter - Parameter value. PParameter or \*Parameter returns the value of Parameter *Parameter* of the Active Transaction.

◆ PR - Transaction priority. The value of the priority of the Active Transaction.

◆ XN1 - Active Transaction. The Transaction Number of the Active Transaction.

## 4.2. Block Entities

The GPSS Block entity is the basic structural element of the simulation. It is useful to think of a GPSS model by its Block diagram. This is the connected network of Block symbols which correspond to the positions of Block entities in the simulation. Representations of the Block diagram can be viewed in the Block Input Window or a Blocks Window. Each Transaction in the model is contained in exactly one Block, but most Blocks may contain many Transactions.

The sequence of Blocks encountered by various Transactions determines the nature and much of the outcome of any simulation. Each Transaction enters one Block then the next, until it is TERMINATED or the simulation ends. Transactions occasionally must wait in a Block until conditions are favorable for entry into the next Block. This may happen in any of several ways, the details of which are better described in Chapter 9.

Each type of Block is associated with an action that transforms other entities in the simulation. These actions are described in Chapter 7 and are supervised by the Transaction Scheduler, described in Chapter 9. In general, a Block first determines if the active Transaction can enter. If so, several Block, Transaction, and system wide statistics are updated. Then the Block-specific action occurs and the Transaction's next Block is chosen. Usually, the "Next Sequential Block" (NSB) is scheduled.

The permanent Blocks in a simulation are created during the Initial Model Translation from the Block Statements in the model. The order of Blocks in the Translated simulation is the same as the order of Block Statements encountered by the Translator. Line numbers are ignored by GPSS World. The GPSS Block Statements may be created in the Model Object using the fullscreen text editor or through one or more Block Creation Dialogs, accessed through the Edit / Insert Block menu command.

Block Statements sent to an existing simulation create a one-time temporary Block in a mode called "Manual Simulation". Such interactive Statements cause the active Transaction to attempt a Block entry and then destroy the Block. In this way, Block statements can be used interactively to control the simulation. After the Block action occurs, the active Transaction resumes its old path in the model unless you had entered a TRANSFER statement or some other Block statement with an alternate destination. This is discussed further under manual simulation in Chapter 2.

### Related SNAs

The SNAs associated with Blocks are:

- ◆ **NEntrnum** - Block entry count. The total number of Transactions which have entered Block *Entrnum* is returned.
- ◆ **WEntrnum** - Current Block count. The current number of Transactions in Block *Entrnum* is returned.

## 4.3. Facility Entities

A Facility is an entity which has several attributes, the most important of which is ownership. A Facility may be owned by a single Transaction, in which case it is said to be busy. Or it may not be owned at all, in which case it is said to be idle. Unlike a Storage Entity, a Facility cannot be freed by a Transaction which never owned it. Transactions acquire ownership of a Facility by successfully entering a SEIZE or PREEMPT Block. A PREEMPT Block has the power to displace the existing owner of the Facility. If a Transaction cannot acquire ownership, it comes to rest on a Facility Transaction Chain.

A Facility has several waiting lines for Transactions which are waiting for some Facility-oriented event to occur. Each Facility has a Delay Chain for normally waiting Transactions, a Pending Chain for Interrupt Mode preemptions which were not allowed, and an Interrupt Chain for previously preempted Transactions. Transactions waiting on a Delay Chain, a Pending Chain, or an Interrupt Chain are said to be "in contention" for the Facility. Each Transaction which owns a Facility must eventually give up ownership by entering a RELEASE or a RETURN Block. Since a contending Transaction will generally become the owner of the Facility, contention for a Facility carries the obligation of eventually releasing the Facility.

Those Transactions which fail in their attempt to enter a SEIZE Block come to rest in priority order on the Delay Chain of the Facility. When a Facility is freed by an owning Transaction, the next owner is chosen from occupants of the Facility's Transaction chains. The pending Interrupt Mode preemptions are chosen first, followed by previously preempted Transactions, followed by Transactions waiting normally in priority order on the Delay Chain.

## Entity States

A Facility that is owned by a Transaction is in the "busy" state. If a Facility is not owned by a Transaction, it is in the "idle" state.

A Facility may be available or unavailable. When it is available, Transactions acquire and give up ownership of the Facility normally. When the Facility is unavailable, ownership is not given to newly arriving Transactions. FAVAIL Blocks are used to place a Facility in the available state, and FUNAVAIL Blocks are used to place it in the unavailable state. Any Transactions present at the Facility at the time it was made unavailable are disposed of as specified in the FUNAVAIL Block operands.

## Related Blocks

There are several GPSS Blocks which can be used with Facilities:

- ◆ SEIZE Blocks attempt to take ownership of a Facility.
- ◆ RELEASE Blocks relinquish ownership of a Facility.
- ◆ PREEMPT Blocks attempt to take ownership of a Facility, possibly displacing the existing owner.
- ◆ RETURN Blocks relinquish ownership of a Facility.
- ◆ FAVAIL Blocks place a Facility in the available state.
- ◆ UNAVAIL Blocks place a Facility in the unavailable state.

## Related SNAs

The SNAs associated with Facilities are:

- ◆  $FEnignum$  - Facility busy. If Facility  $Entnum$  is currently busy,  $FEnignum$  returns 1. Otherwise  $FEnignum$  returns 0.
- ◆  $FCEnignum$  - Facility capture count. The number of times Facility  $Entnum$  has been SEIZED or PREEMPTed by a Transaction.
- ◆  $FIEnignum$  - Facility  $Entnum$  interrupted. If Facility  $Entnum$  is currently preempted by a Transaction in an Interrupt Mode PREEMPT Block,  $FIEnignum$  returns 1. Otherwise  $FIEnignum$  returns 0.
- ◆  $FREnignum$  - Facility utilization. The fraction of time Facility  $Entnum$  has been busy.  $FREnignum$  is expressed in parts-per-thousand and therefore returns a real value between 0 and 1000.
- ◆  $FTEnignum$  - Average facility holding time. The average time Facility  $Entnum$  is owned by a capturing Transaction.
- ◆  $FVEnignum$  - Facility in available state.  $FVEnignum$  returns 1 if Facility  $Entnum$  is in the available state, 0 otherwise.

## 4.4. Function Entities

GPSS Function Entities are used to return a value derived from some argument, such as a random number. Actually, any SNA may be used as an argument. A Function is defined by a FUNCTION Command followed by one or more Function Follower Statements. The A operand of the FUNCTION Statement specifies the argument, and the B operand of the FUNCTION statement specifies the Function type and the number of data pairs to appear on the Function Follower Statements. It is the numbers, names, and/or SNAs in the Function Follower Statements that complete the definition of the function entity.

For many purposes, it is more convenient to use PLUS Procedures than Function Entities. The PLUS Language is described in Chapter 1. However, GPSS Functions are well suited for use as list based

functions and empirical probability distributions. In addition, a GPSS Function may be much more efficient in terms of computer time than an equivalent Procedure.

There are 5 different types of function entities:

- ◆ Type C - "Continuous" valued Function. Performs a linear interpolation. A random argument is a special case.
- ◆ Type D - Discrete valued Function. Each argument value or probability mass is assigned an separate value. A random argument is a special case.
- ◆ Type E - Discrete, "attribute valued" Function. Each argument value or probability mass is assigned an SNA to be evaluated. A random argument is a special case.
- ◆ Type L - List valued Function. The argument value is used to determine the list position of the value to be returned.
- ◆ Type M - List valued Function. The argument value is used to determine the list position of the SNA. This SNA is evaluated and returned as the result of the function.

A Function used in operand B of an ADVANCE or GENERATE Block is called a "Function Modifier". The double precision floating point result of the function is multiplied by the evaluated A operand. The result is then used as the time increment required by the Block.

The values specified in a FUNCTION declaration, i.e. in Function Follower statements, are kept internally as double precision floating point values. These values have a precision limited to approximately 15 decimal digits and a magnitude limited to 306 decimal digits. Linear interpolation when evaluating a random function entity involves a random fraction from 0-.999999 taken from the specified random number generator. This random number is multiplied by the interpolation factor and added to the base of the interval.

The FUNCTION Command is discussed in more detail in Chapter 6

## Related SNA

The SNA associated with functions is:

- ◆ *FNEnnum* - Function. Result of evaluating Function *Ennum*.

## 4.5. Logicswitch Entities

A Logicswitch Entity is the simplest entity with only two states: "Set" or "Reset". There are BLOCKS which alter a Logicswitch and SNAs which return the state of a Logicswitch.

Logicswitches are given the value of 0 when created, or when a CLEAR Command (without the OFF option) is sent to the Simulation Object.

## Related Block

- ◆ LOGIC Blocks set, reset, or invert the state of a Logicswitch Entity.

## Related SNA

The SNA associated with Logicswitches is:

- ◆ *LSEnnum* - Logicswitch set. *LSEnnum* returns 1 if Logicswitch is in the "set" state, 0 otherwise.

## 4.6. Matrix Entities

A Matrix Entity is an array of elements, each of which can take on a value. The size of a Matrix is limited by the Maximum Memory Request in the Simulate page of the Model Settings Notebook. You can adjust this if you need to increase the permitted matrix size.

Matrices can have up to 6 dimensions, and can be global and permanent, or local and temporary. A permanent Matrix Entity is defined by a MATRIX Command in the model, or one sent to an existing simulation. Such Matrix Entities have global scope, and can be referenced anywhere in the model.

The elements of global Matrices are given the value of 0 when created, or when a CLEAR Command (without the OFF option) is sent to the Simulation Object. A Matrix Element, or the whole Matrix, may be set in the UNSPECIFIED state by an INITIAL Command.

Temporary Matrices are defined by Temporary Matrix Declarations in PLUS Procedures. They are created when a Procedure is invoked, and destroyed afterward. They have local scope, and can be referenced only within the Procedure in which they are declared. Elements of Temporary Matrices are not initialized. You cannot use one in an Expression until you assign it a value.

## Related Block

- ◆ MSAVEVALUE Blocks modify or assign a value to an element of a Matrix Entity.

## Related Command

- ◆ INITIAL Commands assign a value to a matrix element, without the need for an Active Transaction.

## Related PLUS Statements

- ◆ Temporary Matrix Declarations are used to create temporary Matrices in PLUS Procedures.
- ◆ Assignment Statements can assign a value to any element of any Matrix defined globally or local to the Procedure.

## Related SNA

The SNA associated with matrices is:

- ◆ MXEntrnum(m,n) - Matrix Savevalue. The value in row m, column n of Matrix Entity Entrnum is returned. In Matrices of more than 2 dimensions, all other indices are assumed to be 1. Unlike MX class SNAs, PLUS Expressions can refer to any element of a higher dimension matrix.

## 4.7. Queue Entities

Queue Entities must not be confused with QUEUE Blocks, which are instances of Block entities. Queue Entities are used primarily for the collection of statistics. An accumulation of current count, total entries, total entries finding a zero current count, the maximum count, and the count-time product. QUEUE and DEPART Blocks are used to update the statistics associated with a Queue Entity. The usual procedure is to "sandwich" a SEIZE, PREEMPT, or ENTER Block between QUEUE and DEPART Blocks. Then the queuing statistics for the associated Facility or Storage Entity are kept and reported automatically. Several basic Queue Entity statistics can be retrieved by SNA calls. Also, frequency distributions can be accumulated by the use of QTABLE Commands.

## Entity States

The most important attribute of a Queue Entity is its content. The content of a Queue Entity changes when QUEUE and DEPART Blocks are entered. It can be thought of as the count of items in a waiting line. Several statistics related to the content are maintained automatically. These are accessible through the SNAs described below.

## Related Blocks

There are several GPSS Blocks which can be used with Queue Entities:

- ◆ QUEUE Blocks increase the content of a Queue Entity.

- ◆ DEPART Blocks reduce the content of a Queue Entity.

## Related SNAs

The SNAs associated with Queue Entities are:

- ◆ *QEnnum* - Current Queue content. The current count value of Queue *Ennum*.
- ◆ *QAEnnum* - Average Queue content. The time weighted average count for Queue *Ennum*.
- ◆ *QCEnnum* - Total queue entries. The sum of all Queue entry counts for Queue *Ennum*.
- ◆ *QMEnnum* - Maximum Queue contents. The maximum count of Queue *Ennum*. This is the "high water mark".
- ◆ *QEEnnum* - Average Queue residence time. The time weighted average of the count for Queue *Ennum*.
- ◆ *QXEnnum* - Average Queue residence time excluding zero entries. The time weighted average of the count for Queue *Ennum* not counting entries with a zero residence time.
- ◆ *QZEnnum* - Queue zero entry count. The number of entries of Queue *Ennum* with a zero residence time.

## 4.8. Storage Entities

A Storage Entity is associated with a number of storage units which are allocated and returned by Transactions. Storage Entities can be used as "token pools" for controlling the flow of Transactions in the model.

When a Transaction ENTERs a Storage Entity, it utilizes or occupies one or more storage units of the Storage Entity. A Transaction is denied entry into an ENTER Block if its storage demand cannot be met. Such a Transaction comes to rest on the Delay Chain of the Storage Entity. Then it must wait until other Transactions free enough storage by entering LEAVE Blocks.

Storage capacity may be released by any Transaction, even if it had not previously ENTERed the Storage Entity. However, if more capacity is released than was declared in the STORAGE Command an Error Stop occurs.

When a Transaction enters a LEAVE Block and gives up one or more storage units, other Transactions are sought which can have their storage demands satisfied. A "first-fit-with-skip" discipline is used to schedule Transactions which are waiting. This means that each Transaction on the Delay Chain is tested for fit in the Storage Entity, starting with the highest priority. If a fit is found, the Transaction is removed from the Storage Delay Chain, allowed to enter the ENTER Block, and placed on the CEC behind its priority peers. Then the next Transaction on the Storage Delay Chain is tested.

ENTER and LEAVE Blocks are used to update the statistics associated with a Storage Entity and several SNAs are available which return derived statistics.

Storage entities must be defined by a STORAGE Command.

## Entity States

A Storage Entity is empty when all storage units are allocatable and full when no storage units are allocatable. A Storage Entity may be neither empty nor full. In addition, a Storage Entity is either available or unavailable. Storage requests are granted only if the Storage Entity is in the available state.

## Related Blocks

There are several GPSS Blocks which can be used with Storage Entities.

- ◆ ENTER Blocks attempt to increase the contents of (place tokens in) a Storage Entity.

◆ LEAVE Blocks decrease the contents of (remove tokens from) a Storage Entity.

◆ SAVAIL Blocks place a Storage Entity in the available state.

◆ SUNAVAIL Blocks place a Storage Entity in the unavailable state.

## Related SNAs

The SNAs associated with Storage Entities are:

◆ *REntnum* - Unused storage capacity. The storage content (or spaces available for use by "tokens") available for use by entering Transactions at Storage Entity *Entnum*.

◆ *SEntnum* - Storage in use. *SEntnum* returns the amount of storage content (or "token" spaces) currently in use by entering Transactions at Storage Entity *Entnum*.

◆ *SAEntnum* - Average storage in use. *SAEntnum* returns the time weighted average of storage capacity (or "token" spaces) in use at Storage Entity *Entnum*.

◆ *SCEntnum* - Storage use count. Total number of storage units that have been acquired from Storage Entity *Entnum*.

◆ *SEEntnum* - Storage empty. *SEEntnum* returns 1 if Storage Entity *Entnum* is completely unused, 0 otherwise.

◆ *SFEntnum* - Storage full. *SFEntnum* returns 1 if Storage Entity *Entnum* is completely used, 0 otherwise.

◆ *SREntnum* - Storage utilization. The fraction of total usage represented by the average storage in use at Storage Entity *Entnum*. *SREntnum* is expressed in parts-per-thousand and therefore returns a real value between 0 and 1000, inclusively.

◆ *SMEntnum* - Maximum storage in use at Storage Entity *Entnum*. The "high water mark".

◆ *STEntnum* - Average holding time per unit at Storage Entity *Entnum*.

◆ *SVEntnum* - Storage Entity in available state. *SVEntnum* returns 1 if Storage Entity *Entnum* is in the available state, 0 otherwise.

## 4.9. Savevalue Entities

A Savevalue Entity is associated with a variable that can take on any value. The value may be assigned or modified by Blocks and may be returned by an X class SNA.

Savevalues are given the value of 0 when created, or when a CLEAR Command (without the OFF option) is sent to the Simulation Object. A Savevalue may be set in the UNSPECIFIED state by an INITIAL Command.

## Related Block

◆ SAVEVALUE Blocks assign or modify the value of a Savevalue Entity.

## Related Command

◆ INITIAL Commands assign a value to a Savevalue Entity, without the need for an Active Transaction.

## Related SNA

The SNA associated with Savevalue Entities is:

◆ *XEntnum* - Savevalue. The value of Savevalue Entity *Entnum* is returned.

## 4.10. Table Entities

A Table Entity is a set of integers used to accumulate data for a histogram. Each integer represents a frequency class in a histogram.

A Table Entity is defined by a TABLE Command.

### Related Blocks

- ◆ TABULATE Blocks update the histogram data accumulated in a Table Entity.

### Related SNAs

The SNAs associated with tables are:

- ◆  $TBEntnum$  - Nonweighted average of entries in Table Entity  $Entnum$ .
- ◆  $TCEntnum$  - Count of nonweighted table entries in Table Entity  $Entnum$ .
- ◆  $TDEntnum$  - Standard deviation of nonweighted table entries in Table Entity  $Entnum$ .

## 4.11. Userchain Entities

A Userchain Entity contains is a special Transaction Chain, called a User Chain, that can be manipulated by LINK and UNLINK Blocks. Userchains are useful for modeling complex scheduling and queuing algorithms. They provide closer control of Transaction queuing than is available with entity Delay Chains.

A flag called a "Link Indicator" is part of each Userchain Entity. The Link Indicator is useful for using a User Chain to control the queuing of Transactions on a resource. It is discussed in detail with the descriptions of LINK and UNLINK Blocks in Chapter 7.

### Related Blocks

There are two GPSS Blocks which can be used with Userchain Entities:

- ◆ LINK Blocks conditionally place a Transaction on a User Chain.
- ◆ UNLINK Blocks remove Transactions from a User Chain.

### Related SNAs

The SNAs associated with Userchain Entities are:

- ◆  $CAEntnum$  - Average Userchain content. The time weighted average number of chained Transactions for Userchain Entity  $Entnum$ .
- ◆  $CCEntnum$  - Total Userchain entries. The count of all Transactions that have been chained to the User Chain of Userchain Entity  $Entnum$ .
- ◆  $CHEntnum$  - Current Userchain content. The current number of Transactions chained at Userchain Entity  $Entnum$ .
- ◆  $CMEntnum$  - Maximum Userchain content. The maximum number of Transactions chained at Userchain Entity  $Entnum$ . The "high water mark".
- ◆  $CTEntnum$  - Average Userchain residence time. The average duration of Transactions at Userchain Entity  $Entnum$ .

## 4.12. Variable Entities

A Variable Entity is a complex expression which can be calculated on demand. All Variable Entities may be defined from Expressions which include constants, SNAs, arithmetic library functions and arithmetic and logical operators. Expressions are discussed in Chapter 8.

A variable is defined by a VARIABLE, FVARIABLE, or BVARIABLE Command.

- ◆ VARIABLE creates a GPSS arithmetic variable entity which, when evaluated by an SNA call, evaluates the Expression and returns the overall result. In GPSS/PC Compatibility Mode, intermediate results are truncated.
- ◆ FVARIABLE creates a GPSS "floating point" variable entity which, when evaluated by an SNA call, evaluates the Expression and returns the result.
- ◆ BVARIABLE creates a GPSS Boolean variable entity which, when evaluated by an SNA call, evaluates the Expression, then returns 1 if the result is non zero, 0 otherwise. BVARIABLES return 1 if true, 0 if false.

### Related SNAs

The SNAs associated with variables are:

- ◆ BVE $n$  - Result of evaluating Boolean Variable Entity  $n$ .
- ◆ VEn $n$  - Result of evaluating arithmetic or floating point Variable Entity  $n$ .

## 4.13. Numeric Group Entities

A Numeric Group Entity is a set of numeric values. Numeric Groups are useful for recording events or for describing the state of a process which you are simulating.

Numeric Group operations are faster for integers than for real values.

### Related Blocks

There are several GPSS Block s which can be used with Numeric Groups Entities:

- ◆ JOIN Blocks place a value into a Numeric Group.
- ◆ REMOVE Blocks take a value out of a Numeric Group.
- ◆ EXAMINE Blocks test values in a Numeric Group.

### Related SNA

The SNA associated with Numeric Groups is:

- ◆ GNE $n$  - Numeric Group count. GNE $n$  returns the membership count of Numeric Group Entity  $n$ .

## 4.14. Transaction Group Entities

A Transaction Group is a set of Transactions. There is no limit to the number of Transaction groups you may have, and no limit to the number of groups to which a single Transaction may belong. Transaction Groups are useful for classifying and accessing Transactions. The active Transaction can test the Transaction Parameters of the members of any Transaction Group.

### Related Blocks

There are several GPSS Blocks which can be used with Transaction Groups:

- ◆ JOIN Blocks place the entering Transaction into a Transaction Group.
- ◆ REMOVE Blocks take some group members out of a Transaction Group.
- ◆ EXAMINE Blocks test members of a Transaction Group.
- ◆ SCAN Blocks test and/or modify members of a Transaction Group.
- ◆ ALTER Blocks test and/or modify members of a Transaction Group.

## Related SNA

The SNA associated with Transaction Groups is:

- ◆ *GTEntnum* - Transaction Group count. *GTEntnum* returns the membership count of Transaction Group *Entnum*.

## 4.15. Random Number Generators

The GPSS World random number streams are generated by a maximal period 32 bit multiplicative congruential algorithm. The period is 2<sup>31</sup>-2 and does not include 0. You may include any number of random number generators in the simulation without declaring them. The initial seed of the random number generator is the same as the entity number of the random number generator entity. However, only random number generators numbered 1 through 7 can be controlled by an RMULT statement.

The GPSS World Pseudo-random number generation algorithm is based on Lehmer's Multiplicative-Congruential algorithm, with a maximal period. The algorithm produces pseudo-random numbers in the open interval 0 to 2,147,483,647 and it generates 2,147,483,646 unique random numbers before repeating itself. There is an additional shuffling step used by GPSS World. The RN class SNA returns 0-999, inclusively and the evaluation of random functions uses a random number drawn from 0-.999999, inclusively.

The important attributes of random number generators are:

- ◆ Seeds. Unless changed by an RMULT statement, the initial seed is equal to the entity number of the random number generator. For example, RN2 starts with a seed of 2.
- ◆ System usage. GPSS World uses Random Number Generators in the scheduling of time ties, in Fractional Mode TRANSFER Blocks and to sample random numbers for GENERATE and ADVANCE Blocks. You can select which random number generator number is to be used as the source of the random number. This is set in the "Random" page of the Model Settings Notebook.
- ◆ SNA Values. When accessed as an SNA, a random integer value 0-999 is returned. You may build larger random numbers by using Expressions such as 1000#RN2+RN2 to define a new variable entity. The newly defined random numbers are returned by calls to a V class SNA. You may find it more convenient to use the built-in probability distributions in the Procedure Library. They are discussed in Chapter 8.
- ◆ Interpolation values. Fractional values 0-.999999 are drawn from the random number stream when used for interpolation in a continuous random Function.

## Related SNAs

The SNAs associated with Random Number Generator Entities are:

- ◆ *RNEntnum* - Random number. *RNEntnum* returns a random integer 0-999 from Random Number Generator Entity *Entnum*.

## 4.16. Data Streams

A Data Stream is a sequence of text lines used by a GPSS World simulation. Each Data Stream is identified by a unique number, so that many can be processed at the same time within in a single simulation. Data Stream numbers are arbitrary positive integers, assigned by you.

You can use a Data Stream to read and write to a file, or to maintain a set of directly accessible data in the memory of your computer. Open, close, read, write and seek operations exist both as GPSS Blocks and as PLUS library procedures. We denote the Blocks in all capital letters, and the Procedures only capitalizing the first letter. You can perform complex file input/output operations within your own PLUS Procedures or by using higher level GPSS Block Entities, and you may mix the two modes. The Blocks are discussed in Chapter 7 and the PLUS Procedures in Chapter 8. For simplicity, we consider only Blocks in the following discussion.

The basic unit of a Data Stream is the *text line*, which is a string of printable characters, including blanks. The built-in library of string Procedures can be used to manipulate text lines. When a string is used as a text line, unprintable characters encountered by a READ or WRITE operations cause the line to be truncated.

Elementary operations involving Data Streams, such as READ, WRITE, and SEEK each involve a single line of text. Data read from a file is stripped of any terminating CR or LF character before being brought into the simulation. Similarly, a CR/LF sequence is added automatically to any text string written by the simulation. Therefore, you do not need to add control characters within your simulation. You will pass a text string with no CR or LF to a WRITE Block, and you will receive a text string with no CR or LF from a READ Block.

There are two types of Data Streams:

1. Input/Output (or just, I/O, or "File") Streams for accessing files,
2. In-Memory Streams for testing, and for direct access of internal data.

Each Data Stream has a *Current Line Position* This is a 1-relative index to the next line position to be read or written. For example, if a READ is issued when the Current Line Position is 1, the first text line in the Data Stream is retrieved. The SEEK operation can be used to change the Current Line Position in I/O Streams and In-Memory Streams.

A WRITE to an I/O Stream or to an In-Memory Stream is operated in either *Insert Mode* or *Replace Mode*. The Current Line Position is used slightly differently in these two modes.

There are 5 Blocks used to process Data Streams: OPEN and CLOSE, which initialize and complete Data Stream processing, respectively. READ and WRITE, which add to and retrieve text lines from the Data Stream, respectively. SEEK is used to set the Current Line Position in I/O Streams and In-Memory Streams.

### 4.16.1. Data Stream Operations

Each of the following operations can be performed as a GPSS Block or a PLUS library procedure call. The two methods are equivalent and interchangeable, except that Replace Mode is available in the WRITE Block but not the Write() library procedure. In this section we describe primarily the use of GPSS Blocks.

#### OPEN

The OPEN Block initializes the Data Stream and sets the Current Line Position to 1.

You determine which type of Data Stream is to be used by how you specify Operand A of OPEN. This operand requires a string constant to describe the Data Stream. Remember that String constants are PLUS Expressions. You must parenthesize any PLUS Expression when you use it as a GPSS Block Operand.

An I/O Stream is described by a file specification and an In-Memory Stream is described by a null string.

For example, in the OPEN Block Statement, you could specify

**OPEN ("MYFILE.TXT")**

to open an I/O Stream, or

**OPEN (" ")**

to open an In-Memory Stream.

If, when you create an file type Data Stream, you use a file name without a path, the directory of the Simulation Object is assumed to be the location of the file. When an existing file is found, it is completely loaded into virtual memory during the processing of the OPEN Block. If no file is found, it is assumed that you are creating one, and processing continues.

If the Data Stream has already been opened, the open operation concludes without raising an error condition.

You can issue open operations within your PLUS Procedures by invoking the Open() library procedure.

After the open operation completes, all data is kept as part of the Simulation Object until the Data Stream is closed. Any changes to the data are returned to the file system only when the Data Stream is terminated by a CLOSE Block or a Close() library procedure.

## CLOSE

The CLOSE Block releases resources used by the Data Stream, and returns the error code. For IO Streams, it writes the data from virtual memory to the disk file.

You can issue close operations within your PLUS Procedures by invoking the Close() library procedure.

## READ

The READ Block retrieves the next text line. In IO and In-Memory Streams it retrieves the text line at the Current Line Position, and increments the Current Line Position. If no text line is there, the Active Transaction takes the Alternate Destination but does NOT store an error code internally.

You can issue read operations within your PLUS Procedures by invoking the Read() library procedure.

## WRITE

The WRITE Block passes a line of text to the Data Stream.

The action depends on which mode the WRITE Block operates in. If Operand D of WRITE is ON, Insert Mode is used. This is the default. If Operand D is OFF, Replace Mode is used.

Although you can issue write operations within your PLUS Procedures, Replace Mode is not supported by the Write() library procedure.

### Insert Mode

This is the default mode for WRITE operations.

Action:

1. Move all text lines at, or after, the Current Line Position down one position.
2. If the Current Line Position is after the last text line, set it to just after the last text line in the Data Stream.
3. Place a copy of the new text line at the Current Line Position.
4. Increment the Current Line Position.

### Replace Mode (WRITE Blocks only)

Action:

1. If the Current Line Position is after the last text line, fill any intervening line positions with null text lines.
2. Delete any text line at the Current Line Position.
3. Place a copy of the new text line at the Current Line Position.
4. Increment the Current Line Position.

## SEEK

The SEEK Block sets the Current Line Position. The Current Line Position is never allowed to be less than 1. Attempts to do so, set it to 1.

You can issue seek operations within your PLUS Procedures by invoking the Seek() library procedure, which returns the previous Line Position that existed before the invocation. This return value is not available when the SEEK Block is used.

### 4.16.2. Using Data Streams

You select the type of Data Stream by the way you specify Operand A of OPEN.

#### I/O Streams

If you specify a file specification in Operand A of OPEN, an I/O Stream is created. If you do not include a file path in the specification, GPSS World assumes you are using the Simulation Object's directory.

In an I/O Stream, when you issue an OPEN, the whole file is brought into the memory of your computer. When you CLOSE, the whole file is written out to disk. Actual file writing only occurs at CLOSE time.

No error occurs if a file cannot be found. In that case, GPSS World assumes you intend to create one. If you need to verify that the file existed, you should issue a READ before continuing.

Here is a simple example using an I/O Stream. This is a small GPSS model segment that opens a file, reads the first text line from it, issues a SEEK to text line 20, writes a text line there, and closes.

```
*****
*
* Read and Modify MYFILE.TXT
*
*****
GENERATE 1,,,1
OPEN      ("MYFILE.TXT"),1,Done      ;Copy the file to memory
READ      Text_Parm,1,Done          ;Place text line in parm
SAVEVALUE Opening_Line,P$Text_Parm ;Text line to safeplace
SEEK      20,1                     ;Access text line 20
WRITE     ("New Line 20"),1,Done    ;Replace the line
Done      CLOSE      Error_Parm,1   ;Copy the file to disk
SAVEVALUE File_Error,P$Error_Parm ;Put in Standard Report
TERMINATE 1
```

#### In-Memory Streams

If you use a null string in Operand A of OPEN, you create an In-Memory Stream. In an In-Memory Stream, all text lines are kept in memory. When you CLOSE, all lines are deleted.

In operation, an In-Memory Stream works exactly like an I/O Stream, except that no data is in the stream when you start to use it, and the data is not saved when you are done. The simulation must load the Stream with text lines before it can read from it.

In-Memory Streams provide the advantage of direct access to data utilizing the SEEK Block and allow you to test the use of Data Streams without actually accessing a file.

### 4.16.3. How to Test For Errors

You can choose to handle Data Stream Errors yourself within the Simulation or you can force the Simulation to Error Stop when one is encountered. Normally no Error Stop occurs. This is controlled by a Setting on the Simulation Page of the Simulation's Settings. To change this setting choose the **Edit / Settings** menu item, and check the **I/O Stream Error Stops** checkbox.

Errors and other unexpected conditions cause a Data Stream Error Code to be stored internally, and cause the Active Transaction entering the Block to take the Alternate Block Destination, if any. A CLOSE Block can retrieve the Data Stream♦'s error code. Only the first non-zero Error Code is remembered. All others are discarded.

An Alternate Destination can be specified as Operand C in OPEN, CLOSE, READ, and WRITE Blocks. The Active Transaction will go to the Alternate Destination Block, instead of the Next Sequential Block, if an error condition occurs. If you do not use these operands, you are essentially ignoring error conditions, for the time being. Normally, you should use the Alternate Destination operands to send the Active Transaction to a CLOSE Block which retrieves the Error Code. You could then place the code in a Savevalue and terminate the simulation.

When the Active Transaction enters a CLOSE Block, the stored Error Code is placed in a Transaction Parameter. In addition, if the error code is non zero, and you have specified CLOSE Operand C, the Alternate Destination will be taken by the Active Transaction.

### End of Data

A special case occurs when the Active Transaction enters a READ Block, but there is no text line at the Current Line Position. This happens normally when you have read all the text lines in a file. This condition causes the Alternate Destination to be taken by the Active Transaction, but does NOT cause an error code to be stored internally.

### Error Codes

- ♦ 0 - No Errors.
- ♦ 10 - OPEN Error. Filename too long. Data Stream is not created.
- ♦ 11 - OPEN Error. Error while reading an external file. Data Stream is not created.
- ♦ 12 - OPEN Error. Memory request was denied while trying to read an existing file. Data Stream is not created.
- ♦ 21 - READ Error. A memory request was denied while trying to perform a READ.
- ♦ 22 - READ Error. Data Stream has not been successfully opened.
- ♦ 31 - WRITE Error. A memory request was denied while trying to perform a WRITE.
- ♦ 32 - WRITE Error. Data Stream has not been successfully opened.
- ♦ 41 - CLOSE Error. An I/O Error prevented the file from being written to disk.
- ♦ 43 - CLOSE Error. Data Stream has not been successfully opened.
- ♦ 51 - SEEK Error. Data Stream has not been successfully opened.

## 4.17. Continuous Simulation

GPSS World can automatically integrate systems of ordinary differential equations. Integration of User Variables in GPSS World is extremely easy. One or more INTEGRATE statements, and variable initialization, is all that is needed. Integration is done automatically by a modified variable step 5th order Runge-Kutta-Fehlberg method, RKF4(5).

Systems of ordinary differential equations, of any order, can be simulated. Plot and Expressions Windows are available for online viewing of the states of variables.

### 4.17.1 How to Set up an Integration

There are two things to do in order to set up the automatic integration of a User Variable. You must assert an INTEGRATE Command and you must give the variable an initial value.

Let us assume you have a simple ordinary differential equation of the form

$$y' = f(-)$$

where **f(-)** is an Expression, possibly involving the System Time, i.e. the AC1 System Numeric Attribute, and other User Variables. **f(-)** is the derivative of **y** with respect to time.

First, put the derivative in parentheses and set up the INTEGRATE Command as

**Y\_INTEGRATE ( f(-) )**

Second, make sure **Y\_** has a starting value, such as

**Y\_EQU 100.3**

Then, when the simulation advances the clock, it automatically sees to it that the value of the User Variable **Y** is kept current.

Since several single letter names clash with SNA classes, here we append an underscore to be sure that the name is unique. All other User Variables involved in the derivative, must be initialized, as well. PLUS Assignment Statements can also be used to assign values to User Variables.

Integrations are automatically begun in the active, or "enabled" state. However, you can turn an integration ON or OFF while a simulation is running by using one or more INTEGRATION Blocks. This is discussed in Chapter 7.

### 4.17.2 Basic Concepts

A derivative states how fast a variable is changing. For example, if your inventory is building up at a rate of 2 units an hour, and your simulation is in units of seconds, all you need to do is add

**Inventory INTEGRATE ( 2.0 / 3600 )**

and

**Inventory EQU 100**

or whatever the starting Inventory value is.

The rate of change (the derivative) leads GPSS World to automatically increase the numeric value of Inventory throughout the simulation. When sales occur, you can simply decrease the Inventory using a PLUS Procedure. The discrete sales event differs fundamentally in that it occurs with no simulated time duration, reducing the value of Inventory instantly.

Integration takes a lot more computer time than the evaluation of a closed form expression. In an example as simple as this one where the variable can be calculated as a function of time, it is much faster to simply calculate the value of the variable given the simulated time, than to integrate the User Variable. Integration should generally be reserved for those cases where you don't have the solution of the differential equation.

### 4.17.3 Thresholds

Integrated User Variables create Transactions when they cross thresholds. This makes it easy to use a continuously modeled value to trigger discrete events.

Each INTEGRATE Command may have zero, one, or two numeric thresholds. Operands B and C can be used to specify threshold 1, and/or operands D and E can be used to specify threshold 2. In either case, the first operand of the pair determines the value of the threshold, the second indicates the Block which will receive generated Transactions.

During the integration, if the value of the integrated variable crosses the value of a threshold, from either direction, a new Transaction is created. It is given a priority of 0, and is scheduled to enter the Block associated with that threshold in the INTEGRATE Command. The Transaction's time of entry into the model is estimated by a linear interpolation. To improve accuracy, the integration ministep is decreased when a threshold is imminent.

Thresholds may be constants, parenthesized Expressions, or even Procedure Calls. In addition, the Transactions generated by a threshold crossing may be used to move the threshold.

Both thresholds behave identically; there is no need to specify one as upper and the other as lower. It is the crossing of the threshold (in either direction) that triggers a Transaction arrival. If the direction of crossing is meaningful to your model, you will have to either keep track of the state of the integrated variable, or test for the direction of crossing when the Threshold Event occurs.

#### 4.17.4 Higher Order Equations

You must reduce the order of higher order ordinary differential equations. If you have higher order differential equations, before you enter them into GPSS World, you should rewrite them as a system of first order equations. This is relatively easy, and requires that you introduce a new variable for each of the intermediate derivatives.

For example, if you have

$$25 y''' - 6 y'' + y = 0.$$

Let  $u = y''$

$$v = u' = y'''$$

For even higher orders, we would continue to introduce variables.

Then, substituting into the original equation and using only one first order derivative, we have

$$25 v' - 6 u + y = 0.$$

Now we have the following system of equations:

$$y'' = u$$

$$u' = v$$

$$v' = 6/25 u - 1/25 y$$

In GPSS World, you could use the following Statements

**Y\_INTEGRATE U\_**

**U\_INTEGRATE V\_**

**V\_INTEGRATE ( (6/25) # U\_ - (1/25) # Y\_ )**

To these, you must add EQU Statements that initialize U\_, V\_, and Y\_.

## 4.17.5 Continuous Only

If you want to do a purely continuous simulation, you still need to create a termination condition by entry into a TERMINATE Block. Therefore, include a GPSS segment such as

```
GENERATE End_Time
```

```
TERMINATE 1
```

In the model, which is begun by a

```
START 1
```

## 4.17.6 Phases

Simulations run in alternate continuous and discrete phases. At any instant where events are scheduled, the simulation runs in a discrete phase. The clock does not advance within an instant in a discrete phase. Between instants, the simulation runs in a continuous phase, during which the integrations proceed in small time increments called ministeps. Plotted integration variables report intermediate values at the end of ministeps.

When a threshold crossing generates a Transaction, the simulation goes into a discrete phase. In this manner, the continuous and the discrete phases can be closely interrelated. Conversely, User Variables can be assigned new values in a discrete phase even if they are being integrated. You can do so using an EQU Command, or a PLUS Assignment Statement. If you want such assignments to occur within the running of the simulation, you must define a PLUS Procedure that makes the assignment. For example, if you defined a PLUS Procedure as

```
PROCEDURE SetPop(Pop_Level) FOXES = Pop_Level ;
```

you could reinitialize the FOXES User Variable by entering a PLUS Block, such as

```
PLUS (SetPop(200))
```

or by using a parenthesized Expression that invokes SetPop() in some other kind of Block.

## 4.17.7 Integration Error

A Model Setting called the Integration Tolerance is used to limit the local truncation error of integrations. This setting applies to all integrations performed during the simulation. If you make the tolerance smaller, the integrations will take longer, but will be more accurate. This is set in the Simulate Page of the Model Settings Notebook.

CHOOSE **Edit / Settings**

Then select the **Simulate** page. Then fill in the desired value in the entry box marked "Integration Tolerance". The installation default is  $10^{-6}$ .

## 4.17.8 Related Command

◆ INTEGRATE Commands set up automatic integrations.

## 4.17.9 Related Block

◆ INTEGRATION Blocks turn the integration of variables ON or OFF.

[\[Table of Contents\]](#)

# Chapter 5 - GPSS World Windows

This chapter is a display of all the major windows in GPSS World. The general operation of GPSS World, including menus and dialogs, is described in Chapter 2 of this manual.



Figure 5-1. The Main Window

A screenshot of the Model Text window titled "Barber.gps". The window displays GPSS code for a "Barber Shop Simulation". The code includes various commands like QTABLE, GENERATE, TEST, SAVEVALUE, ASSIGN, QUEUE, and SEIZE, along with comments explaining their functions. The window has scroll bars on the right side.

Figure 5-2. The Model Text Window

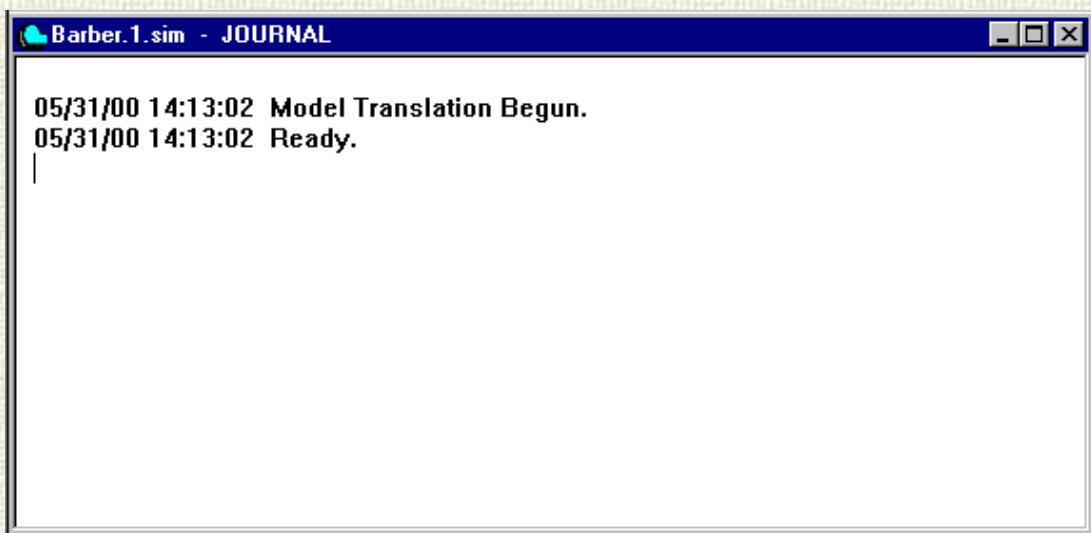


Figure 5-3. The Simulation Journal Window

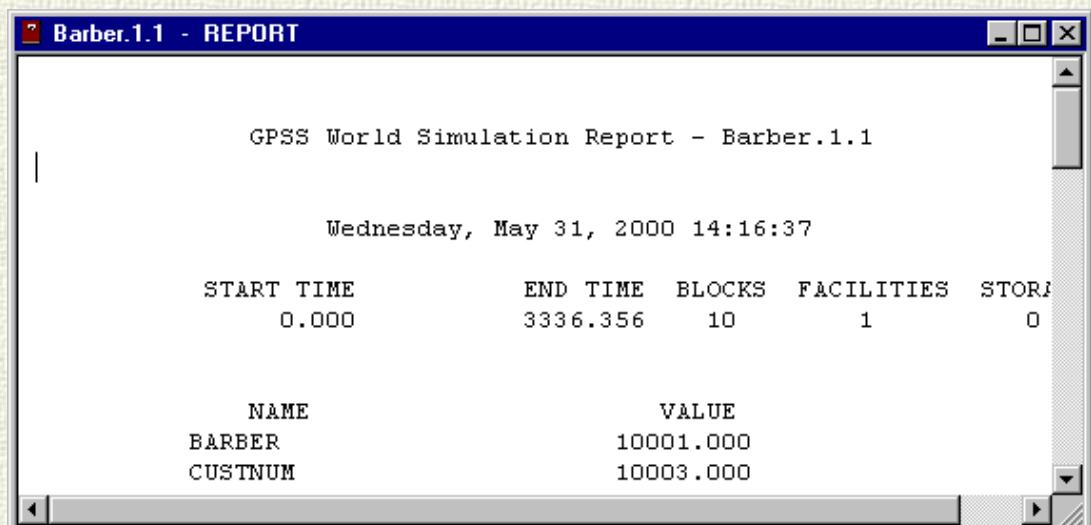


Figure 5-4. The Report Window

**Barber.1.sim:2 - BLOCK ENTITIES**

This screenshot shows the 'BLOCK ENTITIES' window for the simulation 'Barber.1.sim:2'. The window has a toolbar with buttons for Find, Continue, Halt, Step, Place, and Remove. A dropdown menu 'Location' is open. Below is a table with columns: Loc, Block Type, Current Count, Entry Count, Retry Chain, Line Number, and Include File.

Loc	Block Type	Current Count	Entry Count	Retry Chain	Line Number	Include File
2 TES	TEST	0	1002	0	9	0
3 SAV	SAVEVALUE	0	500	0	11	0
4 ASN	ASSIGN	0	500	0	12	0
5 QUE	QUEUE	1	500	0	13	0
6 SEI	SEIZE	1	499	0	14	0
7 DEP	DEPART	0	498	0	15	0
8 ADV	ADVANCE	0	498	0	16	0
9 REL	RELEASE	0	498	0	17	0
FINIS	TERMINATE	0	1000	0	18	0

Figure 5-5. Detail View of the Blocks Window

**Barber.1.sim:2 - BLOCK ENTITIES**

This screenshot shows the same 'BLOCK ENTITIES' window, but it is in a non-detailed view. The list of entities is shown as a vertical column of icons and labels, with 'FINIS' being the last item listed.

Figure 5-6. Non-detail View of the Blocks Window

**Orderpt.1.sim:2 - EXPRESSIONS**

This screenshot shows the 'EXPRESSIONS' window for the simulation 'Orderpt.1.sim:2'. It has a toolbar with buttons for Find, Continue, Halt, and Step. A dropdown menu 'Location' is open. Below is a table with columns: Label, Expression, and Value.

Label	Expression	Value
Daily Demand	X\$Sold	43.000
Inventory Level	X\$Stock	551.000

Figure 5-7. The Expression Window

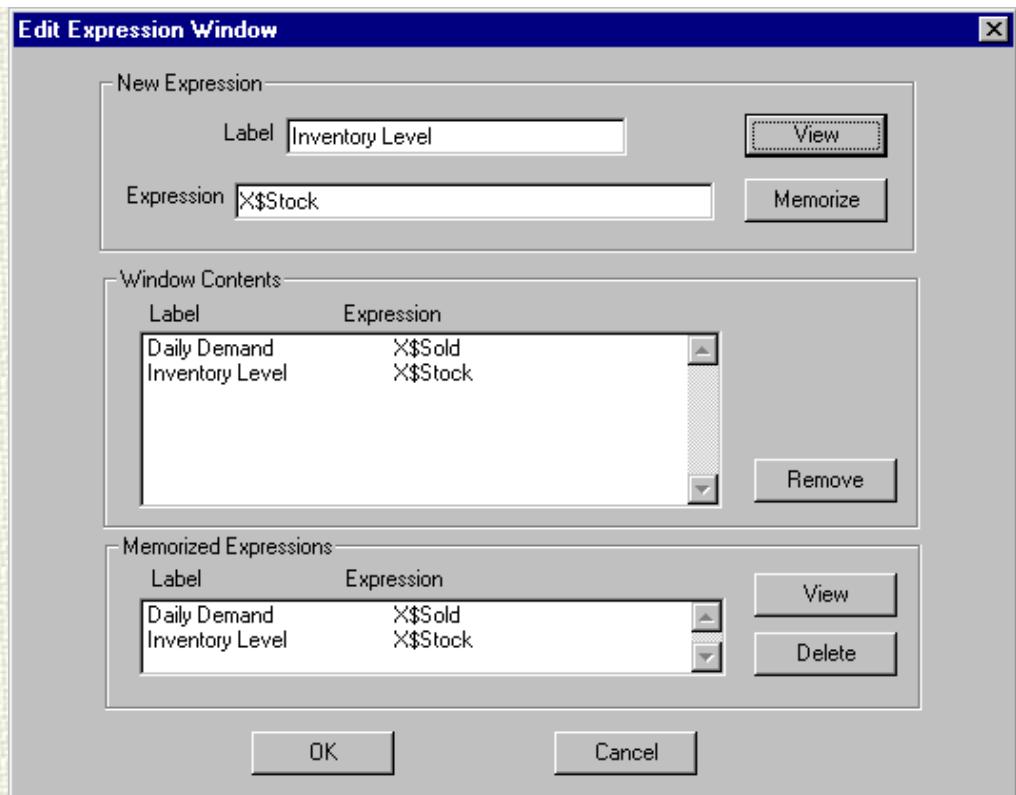


Figure 5-8. The Edit Expression Dialog

Fmsmodel.1.sim:2 - FACILITY ENTITIES									
Facility	Utilization	Delay Chain	Acquisitions	Available	Ave. Time	Owner XN	Retry ...	Pending...	I
F 1	0.028	0	9265	+	20.000	0	0	0	
F 2	0.018	0	5997	+	20.000	0	0	0	
F 3	0.010	0	3268	+	20.000	0	0	0	
F 4	0.028	0	9264	+	20.000	0	0	0	
F 5	0.015	0	5106	+	20.000	0	0	0	
F 6	0.012	0	4158	+	20.000	0	0	0	
F 7	0.019	0	6457	+	20.000	0	0	0	

Figure 5-9. Detail View of the Facilities Window

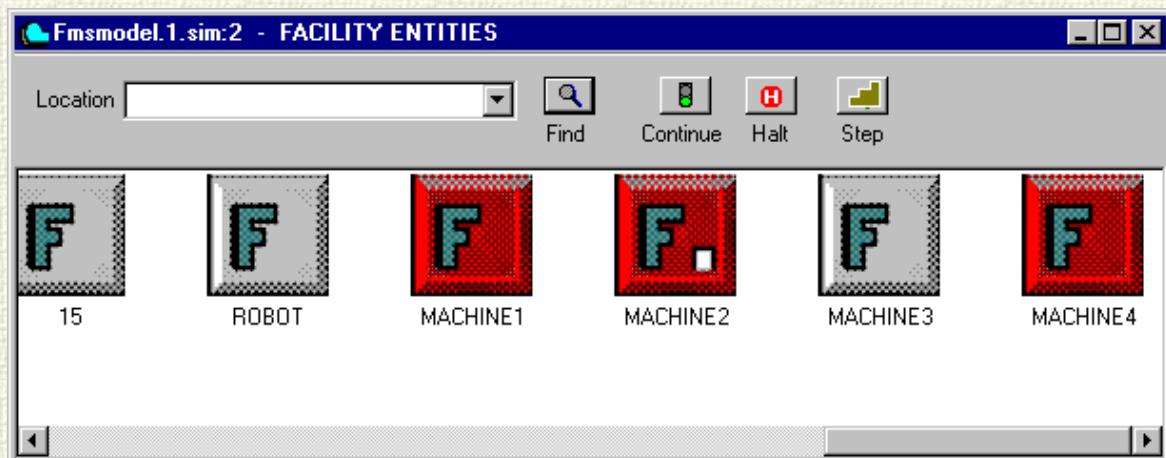


Figure 5-10. Nondetail View of the Facilities Window

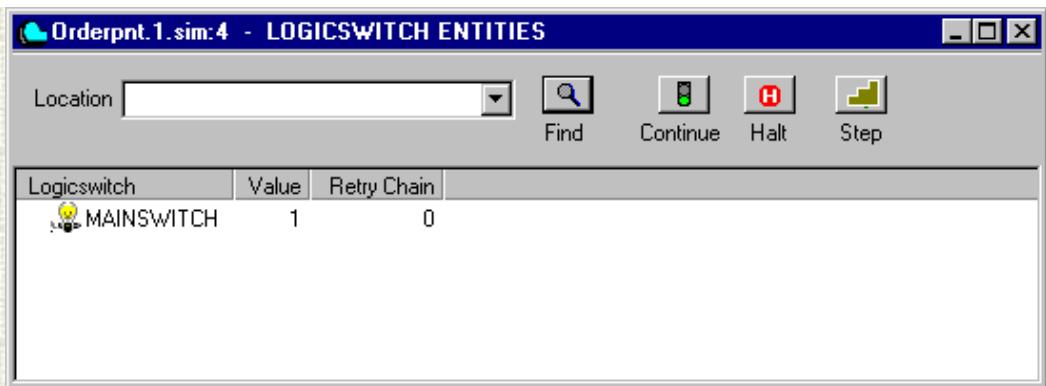


Figure 5-11. Detail View of the Logicswitches Window

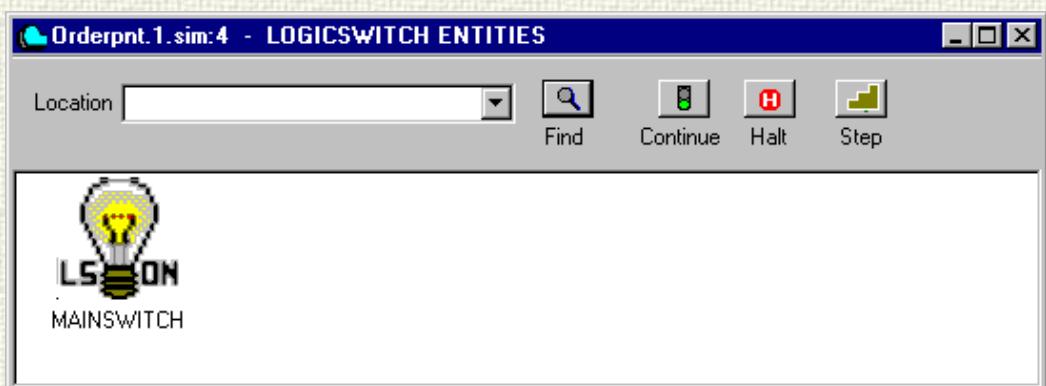


Figure 5-12. Non-detailed View of the Logicswitches Window

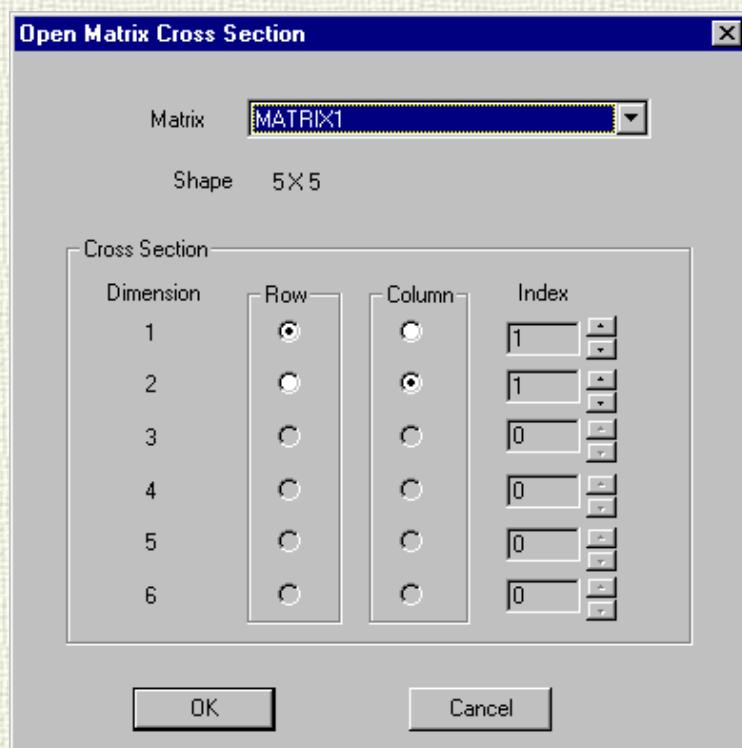


Figure 5-13. The Matrix Cross Section Dialog

Sample9.1.sim:2 - MATRIX WINDOW

MATRIX1		
Dim 2		
Dim 1	1	2
1	0	0
2	0	0
3	0	0
4	0	0

Figure 5-14. The Matrix Window

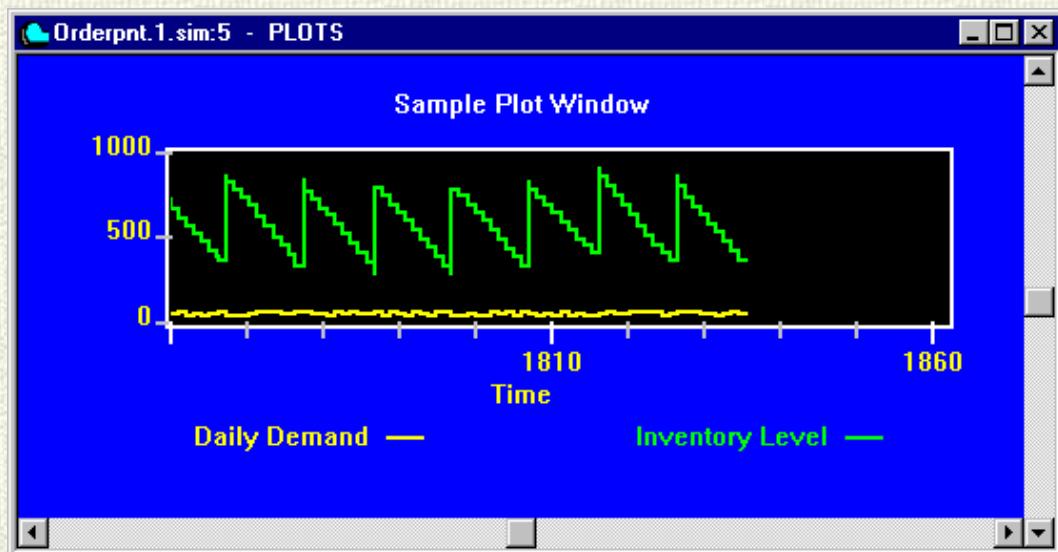


Figure 5-15. The Plot Window

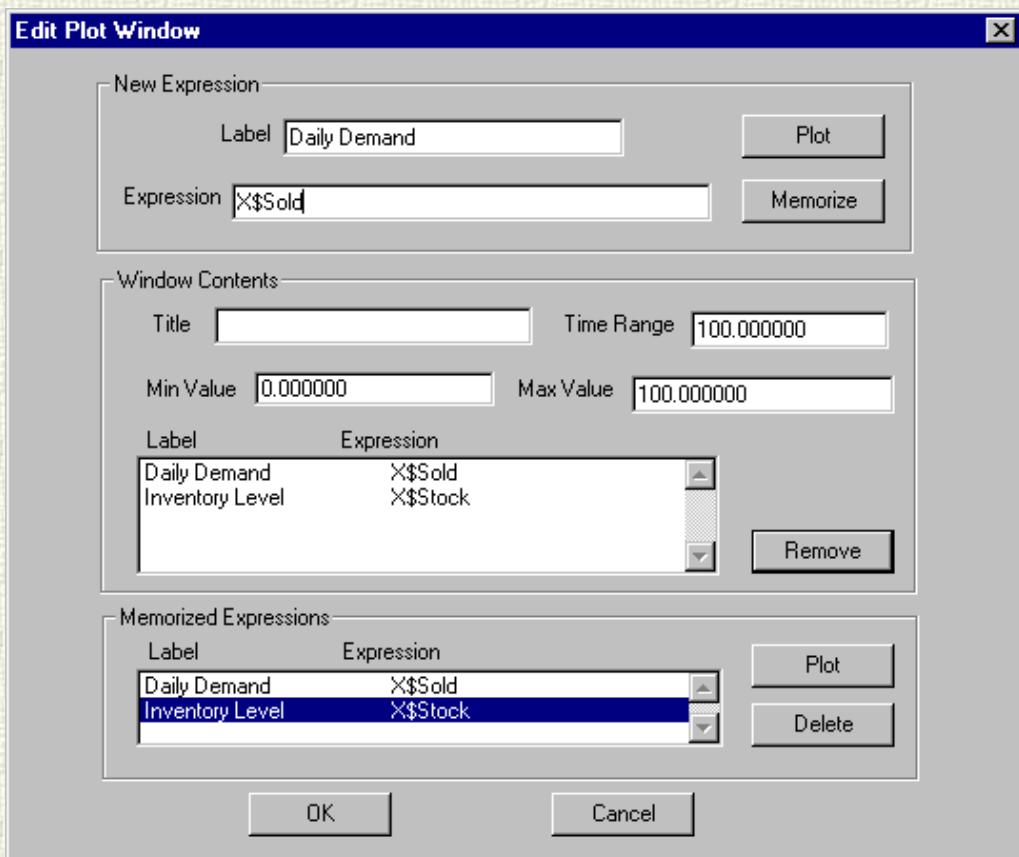


Figure 5-16. The Edit Plot Dialog

The screenshot shows a Windows-style dialog box titled "Fmsmodel.1.sim:4 - QUEUE ENTITIES". The interface includes a toolbar with "Location" dropdown, "Find", "Continue", "Halt", and "Step" buttons. Below the toolbar is a table with the following data:

Queue Entity	Current Cont...	Entry Count	Zero Entry Count	Maximum Content	Average Content	Average Time (+0)	Average Time (-0)	Retry Chain
ARRIVAL	0	9265	0	1	0.090	64.542	64.542	C
ONE	0	3268	1409	4	0.338	690.541	1213.926	C
WIPONE	0	3267	0	1	0.032	64.600	64.600	C
TWO	2	4158	909	9	1.188	1906.920	2440.434	C
WIPTWO	1	4156	0	1	0.041	65.336	65.336	C
THREE	0	2804	1381	6	0.292	694.031	1367.578	C

Figure 5-17. Detail View of the QueuesWindow

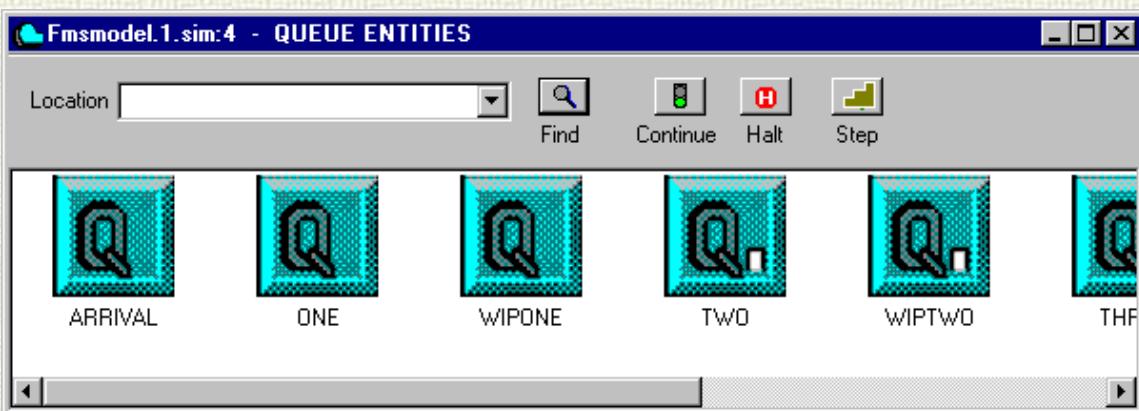


Figure 5-18. Nondetail View of the QueuesWindow

The screenshot shows a Windows-style dialog box titled "Fmsmodel.1.sim:5 - SAVEVALUE ENTITIES". The interface includes a toolbar with "Location" dropdown, "Find", "Continue", "Halt", and "Step" buttons. Below the toolbar is a table with the following data:

Savevalue	Value	Retry Chain
1	520.000	0
2	633.000	0
3	548.000	0
4	621.000	0
5	634.000	0
6	574.000	0
7	570.000	0

Figure 5-19. Detail View of the Savevalues Window

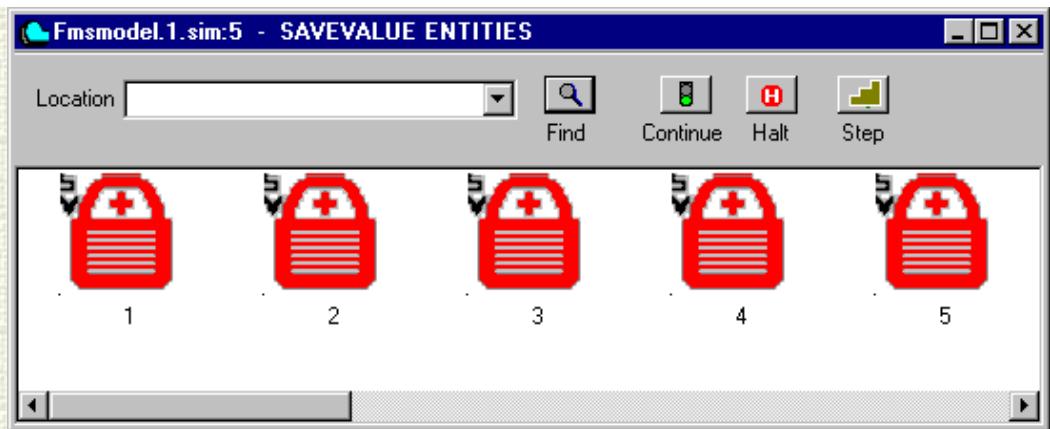


Figure 5-20. Nondetail View of the Savevalues Window

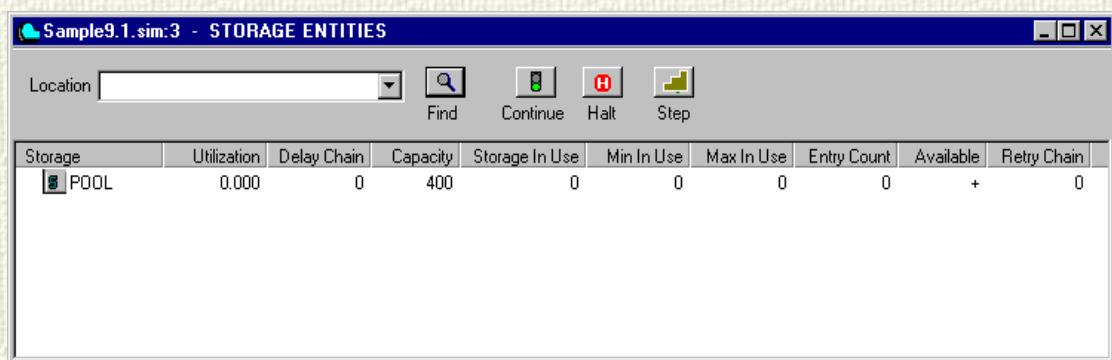


Figure 5-21. Detail View of the Storages Window

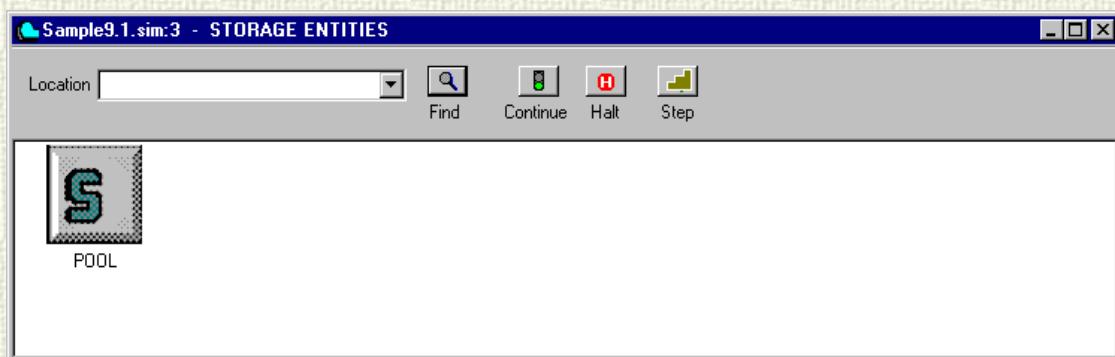


Figure 5-22. Nondetail View of the Storages Window

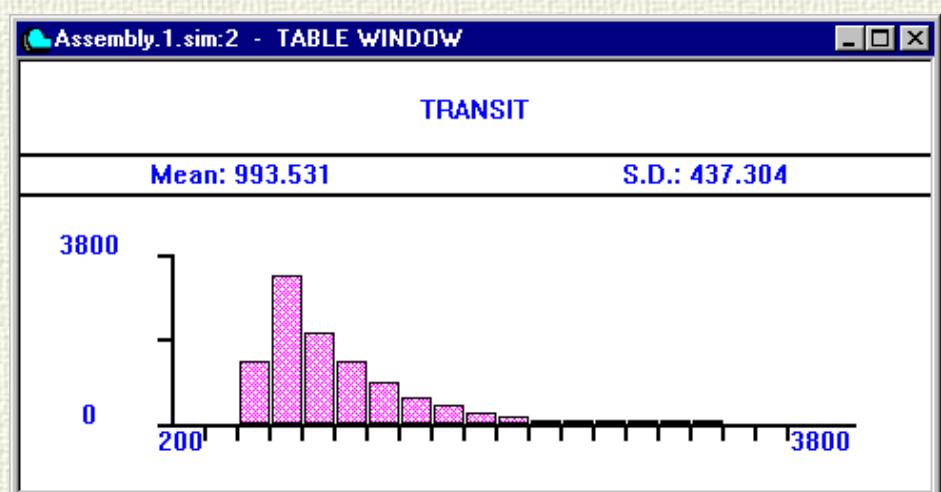


Figure 5-23. The Table Window

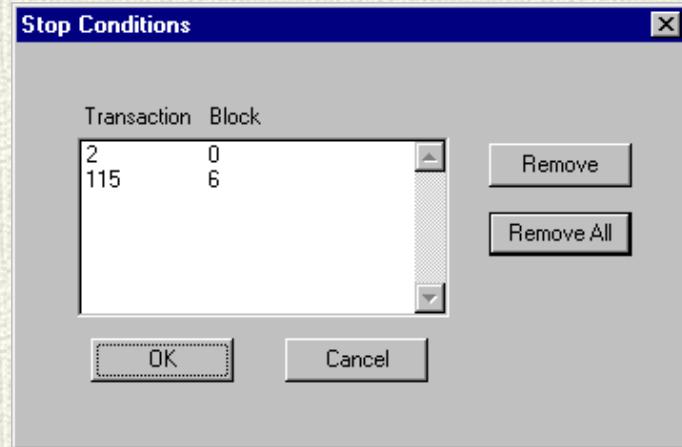


Figure 5-24. The User Stops Window



Figure 5-25. The Current Events Chain Snapshot

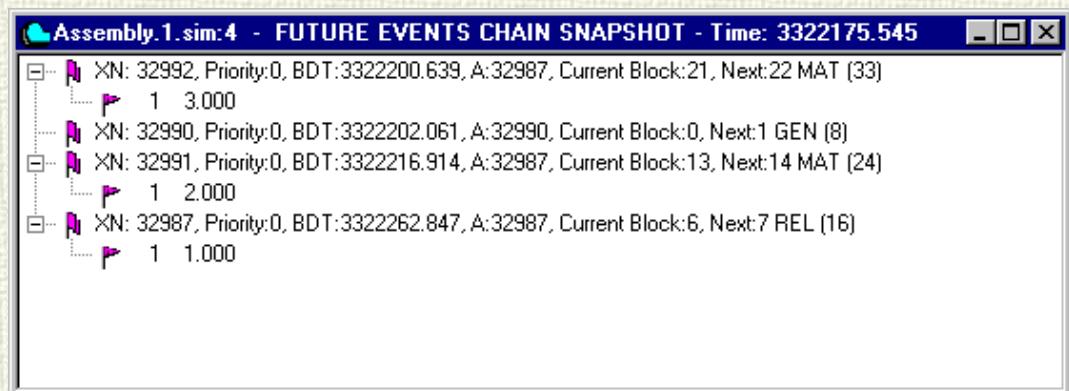


Figure 5-26. The Future Events Chain Snapshot

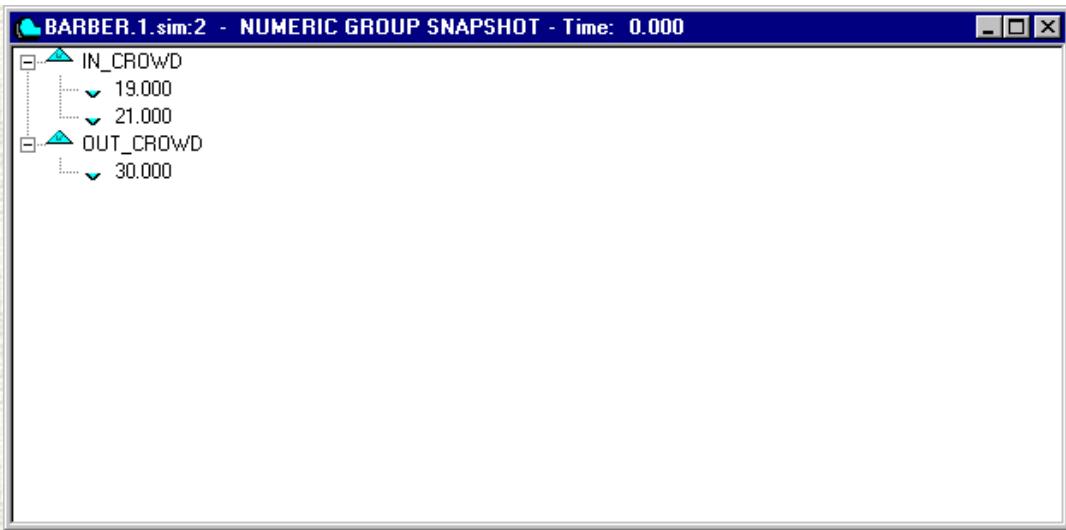


Figure 5-27. The Numeric Groups Snapshot

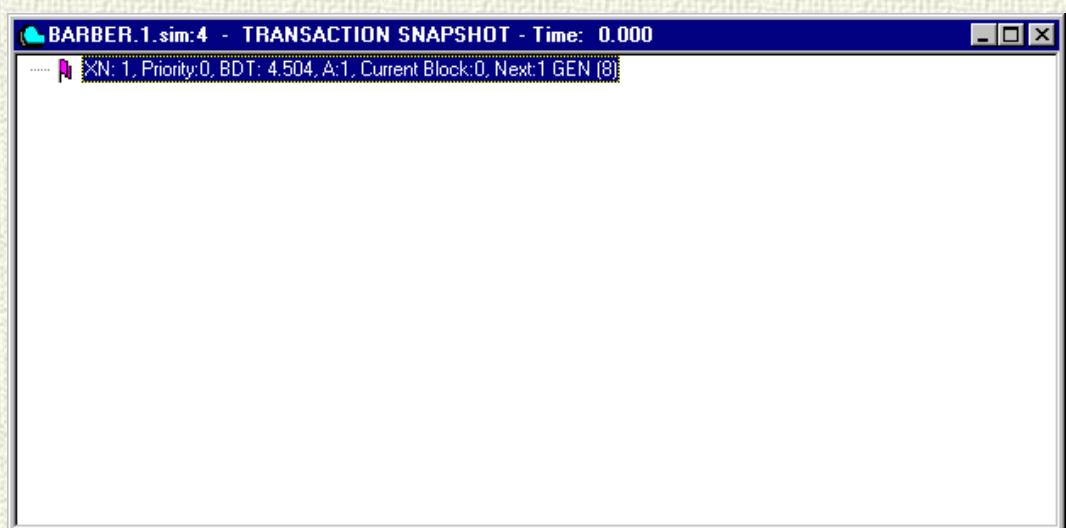


Figure 5-28. The Transaction Snapshot

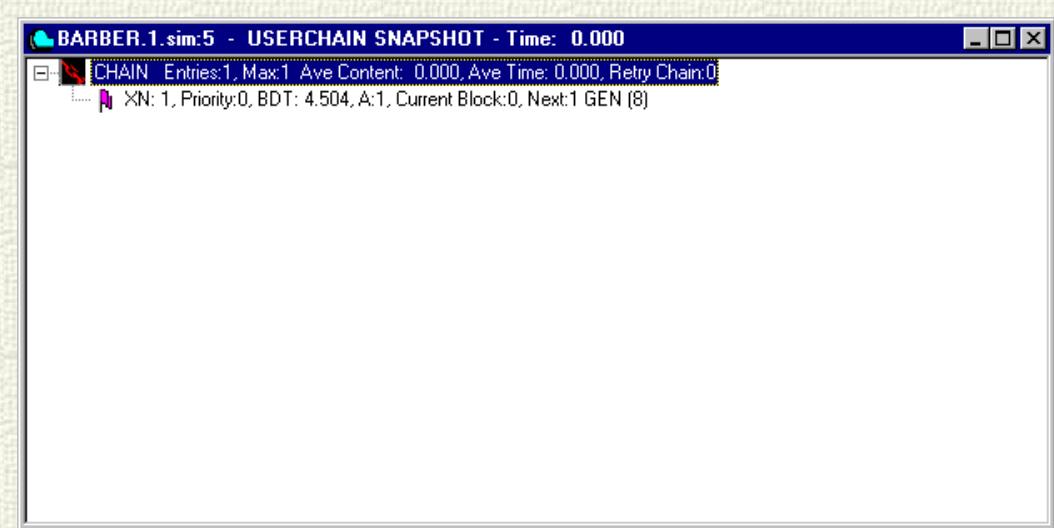


Figure 5-29. The Userchains Snapshot

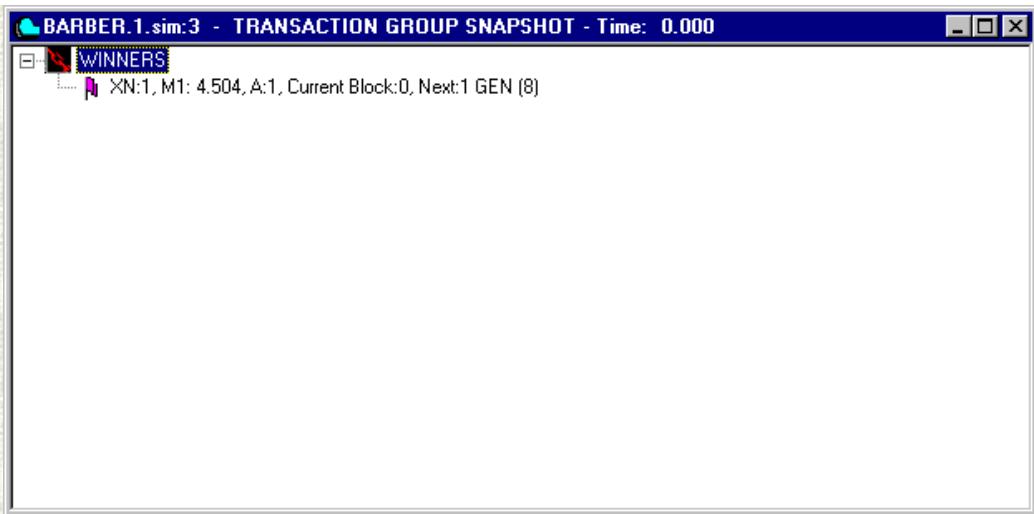


Figure 5-30. The Transaction Groups Snapshot

Insert GPSS Block into Model Object		
ADOPT	ASSEMBLE	ALTER
ADVANCE	CLOSE	COUNT
ASSIGN	GATE	DISPLACE
BUFFER	JOIN	EXAMINE
DEPART	LINK	EXECUTE
ENTER	LOGIC	FAVAIL
GENERATE	LOOP	FUNAVAIL
LEAVE	MATCH	GATHER
MARK	OPEN	INDEX
MSAVEVALUE	PREEMPT	INTEGRATION
PLUS	PRIORITY	SAVAIL
QUEUE	READ	SCAN
RELEASE	REMOVE	SELECT
SAVEVALUE	RETURN	SUNAVAIL
SEIZE	SEEK	TABULATE
SPLIT	TEST	TRACE
TERMINATE	UNLINK	UNTRACE
TRANSFER	WRITE	

Figure 5-31. The Block Input Menu



Figure 5-32. The ADOPT Block Creation Dialog

A screenshot of the 'Screening Experiment Generator' dialog box. The title bar says 'Screening Experiment Generator'. It contains fields for 'Experiment Name' (ScreenEthernet) and 'Run Procedure' Name (DoTheRun). A 'Factors' table lists variables A through F with their corresponding values: A (Node\_Count: 100, 300), B (Min\_Msg: 256, 512), C (Max\_Msg: 6072, 24288), D (Fraction\_Short\_Msgs: 600, 300), E (Intermessage\_Time: 1.0, 0.5), and F (empty). Below the table are 'Fraction' options (Full, Half, Quarter, Eighth, Sixteenth) and a 'Run Count' of 8. The 'Result' section includes an 'Expression' field with the value '(QT\$Global\_Delays/1000)'. At the bottom are checkboxes for 'Generate Run Procedure' (checked) and 'Load F11 with CONDUCT Command' (checked), along with 'Insert Experiment', 'Cancel', 'Help', and 'Alias Groups' buttons.

Figure 5-33. The Screening Generation Dialog

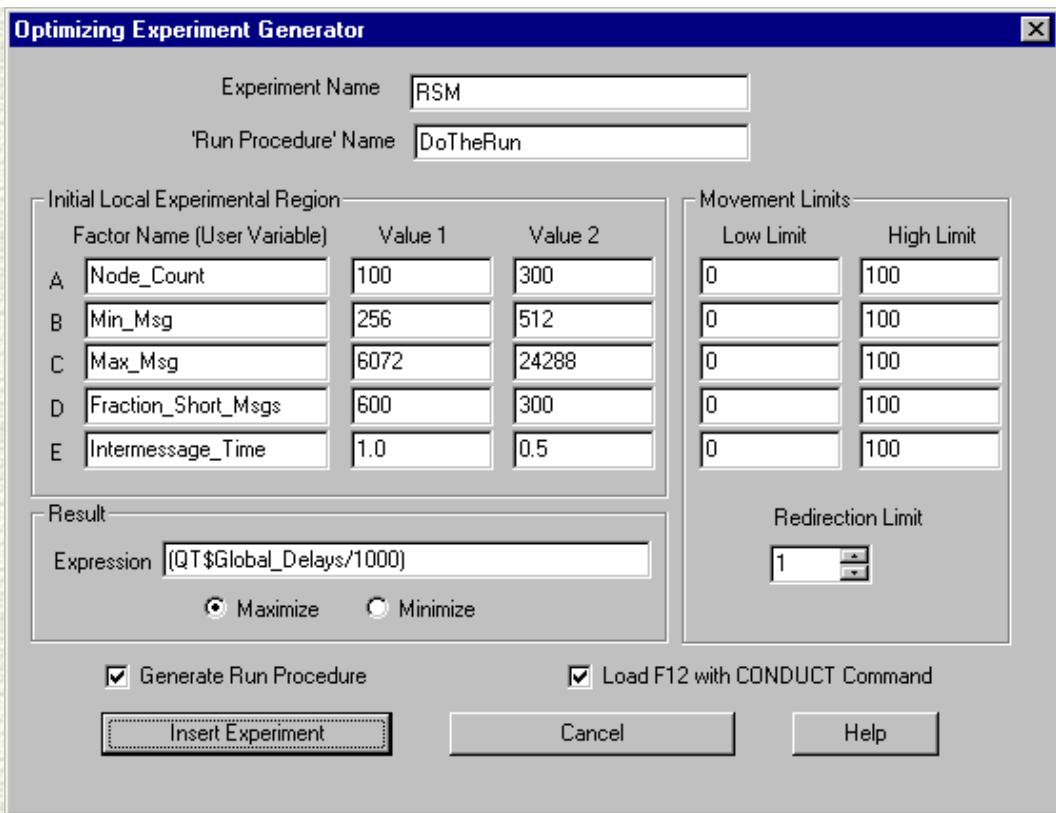


Figure 5-34. The Optimizing Experiment Generation Dialog

[\[Table of Contents\]](#)

# Chapter 6 - GPSS Commands

You use Commands to define entities and to control the running of simulations. Commands may be part of the Initial Model Translation, or they may be sent as Interactive Statements to an existing simulation.

As part of the Initial Model Translation, Commands are sent to the Simulation Object as a group, after all the Blocks have been sent. Otherwise, they are sent when you enter them. These are called Interactive Commands.

To send a Command to an existing simulation use the Command menu in the Model Window. This will Translate a Command and send it to the Simulation Object for execution.

Every simulation has a Command Queue associated with it. The Simulation Object performs each Command on the queue one after the other, until it is HALTED or until it runs out of things to do. Even Commands in the Model File, other than HALT and SHOW, are placed on the Command Queue before they are performed.

Commands are either *Immediate* or *Queued*. Immediate Commands, such as HALT and SHOW, are performed as soon as they are received by the Simulation Object. Other Commands are queued. They are placed at the end of a list of Commands which have not yet been completed. When a Simulation Object has no more Immediate Commands to do on behalf of a simulation, it performs the next Command on the simulation's Command Queue. If a simulation is running when an Immediate Command is received, the simulation is temporarily suspended while the Immediate Command is performed.

The HALT Command is a special case. Not only is it an Immediate Command, but it also deletes any remaining Commands still on the Command Queue. After a HALT Command is performed, the Simulation Object has nothing more to do on behalf of that simulation.

It is often convenient to put a list of frequently used Commands in a small text file. You can then use an INCLUDE Command to send the whole sequence to the Simulation Object. Even easier, you can load a function key with an INCLUDE Command, and have the whole Command list performed by a single keystroke. Chapter 2 shows you how to do this.

The Commands are:

**BVARIABLE** - Define a Boolean Variable Entity.

**CLEAR** - Reset statistics and remove Transaction.

**CONTINUE** - Resume the simulation.

**EQU** - Assign a value to a User Variable.

**EXIT** - End the GPSS World Session.

**FUNCTION** - Define a Function Entity.

**FVARIABLE** - Define an Fvariable Entity.

**HALT** - Stop the simulation and delete all Queued Commands.

**INCLUDE** - Read and Translate a secondary Model File.

**INITIAL** - Initialize or modify a Logicswitch, Savevalue, or Matrix Entity.

**INTEGRATE** - Automatically integrate a time differential in a User Variable.

**MATRIX** - Define a Matrix Entity.

**QTABLE** - Define a Qtable Entity.

**REPORT** - Set the name of the Report File or request an immediate report.

**RESET** - Reset the statistics of the simulation.

**RMULT** - Set the seeds of the first 7 Random Number Generators

**SHOW** - Evaluate and display Expression.

**START** - Set the Termination Count and begin a simulation.

**STEP** - Attempt a limited number of Block entries.

**STOP** - Set a Stop Condition based on Block entry attempts.

**STORAGE** - Define a Storage Entity.

**TABLE** - Define a Table Entity.

**VARIABLE** - Define a Variable Entity.

## Operands

Statements usually have one or more operands which you must fill in. Most operands have several different forms which are valid. In the descriptions which follow, a valid class of operands may be described by an italicized word. You must choose a member of the class and type it into the Operand field. For example, if one of the valid forms of an operand is given as *PosInteger*, you could type:

## 21

The italicized words are usually suggestive, but you may need to refer to the formal definition in the Appendix.

## Windows

A wide variety of windows are available for you to observe the effects of Commands on your simulations. In general, windows are specialized by the entity type.

- ◆ Model Window - Text View ◆ Full screen textual model editor.
- ◆ Journal Window - Record session events.
- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Expressions Window - Online view of values of Expressions.
- ◆ Facilities Window - Online view of Facility Entity dynamics.
- ◆ Logicswitches Window - Online view of Logicswitch Entity dynamics.
- ◆ Matrix Window - Online view of the dynamics of a Matrix cross-section.
- ◆ Plot Window - Online view of a plot of up to 8 Expressions.
- ◆ Queues Window - Online view of Queue Entity dynamics.
- ◆ Savevalues Window - Online view of Savevalue Entity dynamics.
- ◆ Storages Window - Online view of Storage Entity dynamics.
- ◆ Table Window - Online view of Table Entity dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.
- ◆ Numeric Groups Snapshot - Picture of the state of the Numeric Groups in the simulation.
- ◆ Userchains Snapshot - Picture of the state of the Userchain Entities in the simulation.
- ◆ Transaction Groups Snapshot - Picture of the state of the Transaction Groups in the simulation.

# BVARIABLE

A BVARIABLE Command defines a Bvariable Entity.

**NAME BVARIABLE X**

## Label/Operand

**NAME** - Entity Label for this entity. Required. The field must be *Name*.

**X** - Expression. Required. Must be *Expression*. Expressions are discussed in Section 3.4.

## Example

**LINE11 BVARIABLE (BV\$CLK♦AND♦BV\$PHASE2)**

This example defines a Bvariable Entity which is to be evaluated when a BV\$LINE11 SNA is encountered. When it is evaluated, the result is 1 ("TRUE") if the CLK and the PHASE2 Bvariable Entities are TRUE when evaluated. Otherwise, the LINE11 Bvariable returns a 0 ("FALSE").

## Action

The BVARIABLE Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

When a BVARIABLE Command is performed, the Simulation Object creates or redefines a GPSS Bvariable Entity. The Bvariable Entity so created is evaluated when a System Numerical Attribute in the BV class, referring to this particular entity, is encountered by the Simulation Object.

The Expression contained in a BVARIABLE Command is evaluated according to the rules in Chapter 3, and may include calls to PLUS Procedures or to Library Procedures. The final result is converted to integer 0, if 0, or to integer 1 if the result was not zero. The evaluation proceeds differently in GPSS/PC Compatibility Mode.

Expressions must be well-formed according to the rules of elementary algebra. A formal definition can be found in the Appendix. You may use any of the arithmetic and logic operators listed in Section 3.4. If SNAs are used in the Expression field, they are evaluated with respect to the Active Transaction. A Named Value which has not been explicitly assigned a value cannot be used as an item in an Expression. To do so, you must assign a value to it before the Expression is evaluated. Assignments to User Variables are done by EQU Commands or in PLUS Procedures.

Expressions in BVARIABLE Command are not limited to logical operators. They may include arithmetic operators and calls to Library Procedures. The truth values of TRUE and FALSE are treated internally as integer 1 and 0, respectively.

Once a Bvariable Entity is created in a simulation, it is never destroyed. However, it may be redefined later by an interactive BVARIABLE Command.

## GPSS/PC Compatibility

♦ All SNAs are truncated in GPSS/PC Compatibility Mode.

♦ In Bvariable Entity evaluation in GPSS/PC Compatibility Mode, the intermediate results are truncated.

## Related SNA

- ◆ *BVEntnum* - Result of evaluating Bvariable Entity *Entnum*.

# CLEAR

**A CLEAR Command returns the simulation to the unused state.**

## CLEAR A

### Operand

**A** - ON or OFF. If the A Operand is omitted, ON is assumed. Optional. The operand must be ON, OFF or *Null*.

### Action

The CLEAR Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

The CLEAR Command resets all statistics accumulators, clears all Transactions from the simulation, then primes each GENERATE Block with its first Transaction.

The state of all Facility Entities and Storage Entities is reset to show an unoccupied condition. The contents of all Blocks become 0.

When a CLEAR or CLEAR ON Command is performed:

- ◆ All Transactions are removed from the simulation.
- ◆ Current counts are set to 0.
- ◆ System clock is set to 0.
- ◆ Facilities are made idle and available.
- ◆ Tables are set to 0.
- ◆ Storages are set to full availability.
- ◆ Space-time products of Facilities, Storages, Queues, and User Chains are set to 0.
- ◆ Total counts are set equal to 0.
- ◆ Minimum and maximum values are set equal to current content in Queues Entities, Userchain Entities, and Storage Entities.
- ◆ Random number generators are not reset.
- ◆ The local count of generated Transactions in a GENERATE Block is set to 0.
- ◆ Members are removed from all numeric Groups.
- ◆ Savevalue Entities are set to zero.
- ◆ Logicswitch Entities are reset.
- ◆ Matrix elements are set to 0.

If a CLEAR OFF is used, all of the above occur except the last three items. When Operand A is off, Savevalue Entities, Logicswitch Entities, and Matrix Elements are left unchanged.

## Special Restrictions

None.

## Related SNAs

None.

# CONDUCT

A CONDUCT Command begins an experiment.

### CONDUCT A

#### Operands

A - PLUS Experiment Procedure Call. Optional. The operand must be *ProcedureCall*.

#### Action

The CONDUCT Command is an *Immediate Command*.that can only be sent to a HALTed Simulation Object.

The CONDUCT Command begins and passes arguments to a pre-registered PLUS Experiment in a Simulation Object. If the Simulation Object has only a single Experiment with no arguments, operand A is not required in the CONDUCT Command.

#### Example

```
CONDUCT MyExperiment( NumberOfTellers, StartingReplicateNumber )
```

In this example, the PLUS Experiment MyExperiment is started just like any other Procedure. The global User Variables NumberOfTellers and StartingReplicateNumber are used to tell the experiment where to begin or resume the simulation runs. The arguments are evaluated in the global context and passed to the invoked Experiment.

Once you have begun an Experiment with a CONDUCT Command, your ability to interact with the simulation is limited. You can always display the running Simulation System Clock ( **View / Clock** ), but generally you will have to HALT the Experiment in order to interact with the Simulation Object.

## Special Restrictions

A CONDUCT Command cannot be issued by a DoCommand invocation.

Only HALT Commands are available during an Experiment.

The DoCommand Library Procedure can be invoked only during an Experiment.

## Related SNAs

None.

# CONTINUE

**A CONTINUE Command causes a halted simulation to resume.**

## CONTINUE

### Operands

None.

### Example

#### CONTINUE

This Command is used to resume the execution of a simulation.

### Action

The CONTINUE Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

The CONTINUE Command causes a halted simulation to resume. A simulation is halted when it encounters a Stop Condition, is issued a HALT Command, encounters an Error Stop, Stop, or ends normally. Conditions can be set by a STOP or STEP Command.

If the simulation had encountered a Stop Condition, the CONTINUE Command skips the original Stop Condition but does not remove it. If the same condition occurs again, the simulation will stop again. Stop Conditions must be removed explicitly by the OFF option in a STOP Command, or in the Blocks Window. When a model is Translated, all Stop Conditions are removed. This is discussed in Chapter 2 in the Section, [Setting Stop Conditions](#).

A CONTINUE can be used when a simulation has been interrupted by a HALT Command. Since a HALT command removes all Commands from the simulation's Command Queue, only the simulation, and not succeeding Commands, will be resumed.

When the Simulation Object processes a CONTINUE Command, it first determines if a positive Termination Count exists. This means that a previous START Command has not been completed. If the Termination Count is not strictly positive, CONTINUE causes the optional standard report to be written but does not schedule any Transactions. Otherwise, the Simulation Object calls the Transaction scheduler to begin processing Transactions again.

### Hot Key

A CONTINUE Command can be sent to the simulation by pressing the [Ctrl] + [Alt] + [C] key combination. A GPSS World window must have the input focus.

# EQU

**An EQU Command evaluates an Expression and assigns the result to a Named Value.**

# NAME EQU X

## Label/Operand

**NAME** - Named Value to receive a value. Required. The field must be *Name*.

**X** - Expression. Required. Must be *Expression*. Expressions are discussed in Section 3.4.

## Examples

### Price EQU 19.95

This Command defines the name Price and assigns the value 19.95 to it. Future references to the Named Value Price will use a numeric value of 19.95.

## Action

The EQU Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

When the Simulation Object processes an EQU Command, it creates or redefines a Named Variable and evaluates the Expression in the Command. The name so created is associated with a value equal to the result of the evaluated Expression. This value replaces references to the defined name when an operand or Expression is evaluated later.

Named Values may be used for their intrinsic value as User Variables, or they may be used as entity specifiers as Entity Labels. Normally, you will not assign a value to a name used as an Entity Label. System defined names, i.e. names which have not yet appeared in an EQU Command, are not valid by themselves in Expressions or operands. However, they may be used as an entity specifier in an SNA. The Simulation Object will automatically assign a distinct value to such a name.

The Expression contained in a EQU Command is evaluated according to the rules in Chapter 3, and may include calls to PLUS Procedures or to Library Procedures. Expressions must be well-formed according to the rules of elementary algebra. A formal definition can be found in the Appendix. You may use any of the arithmetic and logic operators listed in Section 3.4. If SNAs are used in the Expression field, they are evaluated with respect to the Active Transaction. A Named Value which has not been explicitly assigned a value cannot be used as an item in an Expression. To do so, you must assign a value to it before the Expression is evaluated. Assignments to User Variables are done by EQU Commands or in PLUS Procedures.

Once a Named Value is created in a simulation, it is never destroyed. However, User Variables may change values as a result of later EQU Commands, assignments in PLUS Procedures, or integration. The numerical integration of User Variables is discussed in Chapter 1, and below under the INTEGRATE Command.

## Special Restrictions

- ◆ The values of Block labels may not be changed in an EQU Command.
- ◆ SNAs are evaluated with respect to the Active Transaction.
- ◆ If a name used as an entity specifier is changed after the entity is defined, you will not be able to access the original entity by using that name.
- ◆ FVARIABLE and BVARIABLE entities share the same name space.
- ◆ If you wish to assign a numeric value to an entity name for use in a SELECT Block, make sure the name/number assignments in the EQU Commands precede the entity definitions. For example:

**100 Stor1 EQU 1**

**200 Stor1 STORAGE 10000**

In SNAs and operands, this STORAGE may now be referred to by the number 1 or the name Stor1.

## Related SNAs

None.

# EXIT

**An EXIT Command concludes the GPSS World Session.**

## EXIT

### Operands

**A** - Exit Code. Optional. The operand must be **-1**, **0**, **1**, or **Null**.

### Action

The EXIT Command ends the session immediately.

The A Operand can be used to control the writing of Model Objects and Simulation Objects to files. If Operand a is **0** or not specified, all modified files bring up a message box inquiring as to whether or not each object should be saved. If Operand A is **1**, all Objects are saved. If Operand A is **-1**, no Objects are saved.

The EXIT Command can be used in Batch Mode, so that an "invisible Session" can be run and terminated without window operations. Batch Mode is discussed in Section 2.3.2.

The Exit Operation is also available as a library procedure and can therefore be initiated in a PLUS Procedure.

### Special Restrictions

None.

## Related SNAs

None.

# FUNCTION

**A FUNCTION Command defines the rules for a table lookup.**

There are several types of Function Entities. Each has its own rules pertaining to the table lookup. For each, the lookup table is specified in one or more Function Follower Statements. Type C Functions are a special case. They use a table lookup, followed by a linear interpolation.

The use of Function Commands to define probability distributions has been largely supplanted by the built-in distributions in the Procedure Library. This is discussed in Chapter 8. The old Function Types are still supported by GPSS World.

## NAME FUNCTION A,B

### Label / Operands

**NAME** - Entity Label this entity. Required. The field must be *Name*.

**A** - Function argument. Required. The operand must be *Name*, *PosInteger*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Function type (one letter) followed immediately by the number of data pairs in the Function Follower Statements. Required.

### Action

The FUNCTION Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

A FUNCTION Command together with one or more Function Follower Statements defines a GPSS Function Entity. Later references to an SNA of the class FN can evaluate the Function and return the result. Operand A of the FUNCTION Command is evaluated numerically. There are several types of Functions which must be considered individually. The type is specified in Operand B of the FUNCTION Command.

Each FUNCTION Command must be followed immediately by a list of data pairs, separated by slashes, which define a table. The text lines that contain the list are called Function Follower Statements. Each data pair has an X value and a Y value (or SNA) separated by a comma. Function Follower Statements create tables in the simulation which allow a mathematical function of one variable to be evaluated. When the Simulation Object encounters a reference to an FNEntnum SNA, it evaluates the Function Entity and returns the result. The tables and the manner in which they are referenced depends on the Function type.

### Function Modifiers

FN class SNAs used in Operand B of GENERATE or ADVANCE Statements are called Function Modifiers. When a Transaction enters a GENERATE or ADVANCE Block with a Function Modifier, the result of the Function is multiplied by Operand A of the Block and used as the time increment.

Operand C of an ASSIGN Statements is also called a Function Modifier, although it is specified differently. In this case, only the entity specifier, not the FN class SNA, is used in the ASSIGN Statement. When a Transaction enters an ASSIGN Block with a Function Modifier, Operand C is used to determine the Function Entity number. Then the result of evaluating that Function is multiplied by Operand B of the ASSIGN Block and the result is used as the ASSIGN value.

### Function Types

There are 5 different type of Function Entities:

#### Type C Functions

Type C - "Continuous" valued Function. Given an X value, after a linear interpolation, the Function returns a Y value. A random argument is a special case.

In Function Follower Statements of C type Functions, the X and Y Values must be *Integer*, or *Real*.

In a C type Function, without a random argument, the data pairs in the Function Follower Statements define a piecewise linear Function of the argument. The first data pair defines the left end point and the last data pair defines the rightmost end point. The X and Y values are stored as double precision floating point numbers.

The Function evaluation begins with the evaluation of the argument. The result is used to identify the line segment of the Function. The argument is then used in a double precision linear interpolation to arrive at the double precision result of the Function. If the argument falls outside the end points of the Function definition, the value at the nearest end point is returned.

When Operand A of the FUNCTION Command is an RN class SNA, the Function is said to have a random argument. A type C Function with a random argument is used to define a "continuous"

probability distribution. This is a special case. The Function is specified as the cumulative distribution function (CDF) with 0 as the value of the left end point and 1 the value of the right end point. As before, the CDF is specified as a piece wise linear function. A random number between 0 and .999999, inclusively, is taken from the random number stream and is used in a linear interpolation to arrive at the double precision value of the Function.

#### Example

Output FUNCTION V\$Input,C3  
1.1,10.1/20.5,98.7/33.3,889.2

This example defines a piece wise linear function with two line segments. When the Function Entity is evaluated for a FN\$Output SNA, first the Function argument V\$Input is evaluated. If the argument result is outside the defined range, 1.1 to 33.3, the nearest endpoint is returned. For example, if V\$Input returns 1 or less, FN\$Output returns 10.1.

If the argument result falls within a defined line segment, a linear interpolation is performed. For example, if V\$Input returns 25, then FN\$Output returns the result of the following calculation:

$$98.7 + (889.2 - 98.7) \# (25 - 20.5) / (33.3 - 20.5)$$

Therefore, FN\$Output returns 376.6101563.

#### Example

Xpdis FUNCTION RN200,C24  
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38  
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2  
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8

In this example we define the Function Entity named Xpdis. The FUNCTION Command names the Function, used random number generator 200, and tells GPSS World that it is to be a type C Function Entity with 24 data pairs to follow in one or more Function Follower Statements.

This is an example of an approximation to a negative exponential distribution with mean of 1. You can use a built in probability distribution from the Procedure Library, which is slightly more accurate, but Function Entities are generally more efficient.

### Type D Functions

Type D - Discrete valued function. Each argument value or probability mass is assigned an numeric value. A random argument is a special case.

In Function Follower Statements of D type Functions, the X Values must be *Expression*, and the Y Values must be *Integer*, *Real*, or *Name*.

In a D type Function, without a random argument, the data pairs in the Function Follower Statements define a set of argument values which are associated with specific Function values. The X values in the Function Follower Statements must be non decreasing. They are stored internally in double precision. When the Function is evaluated, the X values from the Function Follower Statements are searched from lowest to highest value. When an X value is found which is greater than or equal to the argument value, the corresponding Y value is returned. If there is no such X value, the Y value or named value associated with the largest X value is returned.

#### Example

Dlis1 FUNCTION X\$A2,D5  
1.1,6.9/2.1,7/6.33,9.4/7,10/9.9,12.01

A D type Function with a random argument is used to define a discrete probability distribution. The Function is specified as a cumulative distribution function (CDF) with 0 as the value of the left end point and 1 the value of the right end point. When such a Function is evaluated, a random number between 0 and .999999 is taken from the random number stream and is used. The smallest X value in the Function Follower Statement that is greater than or equal to the random number is selected. The associated Y value is returned as the value of the Function.

#### Example

Ran1 FUNCTION RN1,D5  
0,0/2,7.2/4,6.667/8,9.92/1,0,10

## Type E Functions

Type E - Discrete, "attribute valued" function. Each argument value or probability mass is assigned an SNA to be evaluated. A random argument is a special case.

In Function Follower Statements of E type Functions, the X Values must be *Expression*, and the Y Values must be *Integer, Real, Name, SNA*, or *ParenthesizedExpression*.

A type E Function is evaluated in the same way as a type D Function, except that a type E Function requires one more step. After the appropriate X value is chosen, the associated SNA (Y value) is evaluated and returned as the result of the Function.

### Example

Edisc FUNCTION X\$QRA,E5  
1,S\$Stor1/3,S\$Stor2/5,S\$Stor3/9,S\$Stor5/10,S\$Stor6

## Type L Functions

Type L - List valued function. The argument value is used to determine the list position of the value to be returned.

In Function Follower Statements of L type Functions, the X Values must be *Integer*, and the Y Values must be *Integer, Real, or Name*.

The Function Follower Statements define a list of values from which the result is chosen. When the Function is evaluated, the argument is evaluated and used as the ordinal number (X value) of the list member. The Y value with that position within the list is returned as the result. If the argument is too large or less than 1, an Error Stop occurs. X values must begin with 1 and be incremented by 1 for each successive data pair. X values may not be omitted in Function Follower Statements.

### Example

Listtype FUNCTION Q\$Barber,L5  
1,PAR1/2,PAR2/3,PAR3/4,PAR4/5,PAR5

## Type M Functions

Type M - Attribute list valued function. The argument value is used to determine the list position of the SNA to be evaluated and returned as the result.

In Function Follower Statements of M type Functions, the X Values must be *Expression*, and the Y Values must be *Integer, Real, Name, SNA*, or *ParenthesizedExpression*.

A type M Function is evaluated in the same way as a type L Function, except that a type M Function requires one more step. After the list position is chosen, the associated SNA is evaluated and returned as the Y value.

### Example

Mlist FUNCTION X\$Name1,M5  
1,Q\$Nnam1/2,Q\$NamX/3,Q\$Nam4/4,Q\$Nam6/5,F\$Tan1

## Rules For Functions

You must obey several rules when you create a Function. They apply to both FUNCTION Commands and/or Function Follower Statements.

- ◆ The X values of Function Follower Statements must be non decreasing.
- ◆ Function Follower Statements are NEVER line numbered.
- ◆ A Function which has a random argument must describe a valid cumulative probability distribution in the Function Follower Statements.
- ◆ All fields in a Function Command are required.

- ◆ All X values and Y values in Function Follower Statements are required.
- ◆ The number of data pairs stated in the B Operand of the FUNCTION Commands must correspond to pairs separated by slashes, [/], in the Function Follower Statements.
- ◆ X<sub>1</sub>, the first probability value specifying a random continuous Function must be 0. CDF values must be nonnegative, nondecreasing, and may not exceed 1.
- ◆ Function Follower Statements have no comments field.
- ◆ In a Function Follower Statement, an X value is followed by [.,], [.] is followed by a Y value, a Y value is followed by [/] or [CR], and [/] is followed by an X value.
- ◆ CDFs must be nondecreasing between 0 and 1, inclusively. Any missing probability in C type Functions is given to the rightmost interval, otherwise missing probability is an error.
- ◆ C,D, and L type Functions cannot have SNAs for Y values.
- ◆ E and M type Functions must have SNAs, or parenthesized Expressions, for Y values.
- ◆ L and M type Functions may not have random arguments.
- ◆ The Function Follower Statements of L and M type Functions must have sequential X values starting with 1.

The special rules applied when running in GPSS/PC Compatibility Mode are discussed in Chapter 3.

## Related SNA

- ◆ FNEnnum - Function. Result of evaluating Function Ennum.

# FVARIABLE

**An FVARIABLE Command defines a "floating point arithmetic" Variable Entity.**

**NAME FVARIABLE X**

## Label / Operand

**NAME** - Entity Label for this entity. Required. The field must be *Name*.

**X** - Expression. Required. Expressions are discussed in Section 3.4.

## Example

**VAR1 FVARIABLE 5#LOG(Q\$WAITINGLINE)**

This Command defines a "floating point" type Variable Entity which is to be evaluated when a V\$VAR1 is encountered. Variables defined by both VARIABLE and FVARIABLE Commands are accessed by SNAs of class V. When the V\$VAR1 SNA is evaluated, the Expression defining the Variable Entity named VAR1 must be evaluated. It begins with the evaluation of the Q\$WAITINGLINE SNA. The logarithm of the double precision result is calculated and multiplied by 5. The result is truncated and returned as the value of the SNA.

## Action

The FVARIABLE Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

When an FVARIABLE Command is performed, the Simulation Object creates or redefines a GPSS Variable Entity. The Variable Entity so created is evaluated when a System Numerical Attribute in the V class, referring to this particular entity, is encountered by the Simulation Object.

The Expression contained in an FVARIABLE Command is evaluated according to the rules in Chapter 3, and may include calls to PLUS Procedures or to Library Procedures. The evaluation proceeds differently in GPSS/PC Compatibility Mode.

Expressions must be well-formed according to the rules of elementary algebra. A formal definition can be found in the Appendix. You may use any of the arithmetic and logic operators listed in Section 3.4. If SNAs are used in the Expression field, they are evaluated with respect to the Active Transaction. A Named Value which has not been explicitly assigned a value cannot be used as an item in an Expression. To do so, you must assign a value to it before the Expression is evaluated. Assignments to User Variables are done by EQU Commands or in PLUS Procedures.

Once a Variable Entity is created in a simulation, it is never destroyed. However, it may be redefined later by an interactive FVARIABLE Command.

If the simulation is not run in GPSS/PC Compatibility Mode, FVARIABLE and VARIABLE Commands are treated the same.

## GPSS/PC Compatibility

◆ All SNAs are truncated in GPSS/PC Compatibility Mode.

## Related SNA

◆ VEnignum - Result of evaluating a Variable Entity Enignum.

# HALT

**A HALT Command interrupts the simulation and purges the Command Queue.**

## HALT

### Operands

None

## Action

A HALT Command is an Immediate Command, and therefore is not placed on the Command Queue by the Simulation Object. Instead, it is performed immediately, causing the simulation to be placed in

the Halted State, and removing any remaining Commands from the Command Queue. The simulation may be resumed by a later CONTINUE Command.

## Example

### HALT

This is the only way to use the GROUPS Command.

## Hot Key

A HALT Command can be sent to the simulation by pressing the **[Ctrl] + [Alt] + [H]** key combination. A GPSS World simulation window must have the input focus.

# INCLUDE

**The INCLUDE Command Translates a file of Model Statements.**

### INCLUDE A

#### Operand

**A** - Filespec. A string representing the file specification of the Secondary Model File or Command List to be Translated. Required. Operand must be *String*.

## Example

### INCLUDE "SAMPLE1.TXT"

In this example, when the Translator encounters the INCLUDE Command, it will include the statements from the SAMPLE1.TXT Text Object in the created simulation. Since no file path is given, GPSS World assumes the file is in the Model Directory.

#### Action

The INCLUDE Command is an *Immediate Command*. It causes statements from a Text Object to be retrieved and Translated. Text Objects are plain text files. When they are used in an INCLUDE Statement they are also called Secondary Model Files.

Operand A is used as the file specification of a Command List or a Secondary Model File. If a path is not included in the filespec, the location of the model file that contains the INCLUDE command will be used.

When an INCLUDE Command is Translated, it causes the GPSS World Translator to replace it with a Text Object, and to Translate that in place of the INCLUDE Command. The Translator opens the Secondary Model File and Translates each Model Statement one at a time.

Model Statements in a Command File or Secondary Model File are treated as if they occurred in line, in place of the INCLUDE Command. Nesting is allowed to a depth of 5. When all Model Statements have been Translated, they are sent to the Simulation Object for processing.

If any errors are detected, GPSS World sounds an audio signal, generates a syntax error message and attempts to find additional errors. The error message contains the offending line number in the Secondary Model File. Handling errors is discussed in Chapter 2. Audio sounds can be suppressed by selecting Silence in the Options page of the Configuration Notebook.

For testing purposes, it may be more convenient to Translate each Secondary Model File by itself, before using it in an INCLUDE Command. Here's how:

1. Read the Secondary Model File in a Model Window.
2. Translate the partial model represented in the Model Window.
3. Correct Syntax Errors.
4. Save the Secondary Model File.

INCLUDE Commands can be entered interactively, or loaded into Function Keys, just like any other GPSS Statement. This is discussed in Chapter 2.

## Model File Numbers

Model File Numbers are used in the Blocks Window and in Error Stop messages to identify the Block Statement that created a given Block Entity.

As the Translator encounters Model Files, it assigns an integer to be used later to identify the file. The Model Object is assigned the number 0, and succeeding INCLUDE Files are assigned succeeding integers as they are encountered by the Translator. Thereafter, Block Entities can be associated to the appropriate text line in the Model File.

Each occurrence of a file causes a distinct Model File Number to be assigned. Multiple occurrences of a single file are also assigned distinct numbers. Since each occurrence causes a distinct set of Blocks to be created, distinct Model File Numbers are required for unique identification.

## Special Restrictions

- ◆ You may use INCLUDE Commands to nest Model Files to a maximum depth of 5.
- ◆ You cannot place an INCLUDE Command inside a PLUS Procedure.

# INITIAL

**An INITIAL Command initializes a Matrix Entity, Logicswitch Entity, Savevalue Entity, or an element of a Matrix Entity.**

## INITIAL A,B

### Operands

**A** - Logicswitch, Savevalue, or Matrix element specified as SNA, or the name of a Matrix Entity. Operand A must have the form of an LS, X, or MX class SNA or a Matrix Name. Required. The operand must be *Name*, *LSPosInteger*, *LS\$Name*, *XPosInteger*, *X\$Name*, *MXPosInteger(m,n)* or *MX\$Name(m,n)*. Coordinates *(m,n)* must be *Name* or *PosInteger*.

**B** - Value to be assigned, or "UNSPECIFIED". The default is 1. Optional. The operand must be *Null*, *Number*, *String*, *Name*, or *UNSPECIFIED*.

### Action

The INITIAL Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

The INITIAL Command causes a value, specified by Operand B, to be assigned to the Logicswitch, Savevalue, or Matrix Entity as specified by Operand A. If Operand B is not used, a value of 1 is assigned to the entity, or element.

If Operand B is the keyword UNSPECIFIED, the Savevalue or Matrix, or Matrix element is placed in the unassigned "Unspecified" state. This can be used to indicate missing data in a Results Matrix which is to be analyzed by the ANOVA Library Subroutine.

If Operand A specifies a Logicswitch Entity, only the value of 0 or 1 is assigned. If Operand B is explicitly specified as 0, the value of 0 is assigned. Otherwise, the value 1 is assigned. The UNSPECIFIED option cannot be used with an LS class SNA, because Logicswitch Entities do not have an unspecified state.

If Operand A specifies the name of a Matrix Entity, all elements in the matrix are placed in the state indicated by the B operand. The default is a value of 1. The MX SNA can be used in Operand A to assign elements in 2 dimensional matrices, but you will need to use a PLUS Language assignment for matrices of more than 2 dimensions. To place such elements in the UNSPECIFIED state, you can INITIALize a Savevalue Entity to UNSPECIFIED and use PLUS to complete the assignment.

The INITIAL Command can be used when there is no Active Transaction, or when Transaction Parameter contents cannot be relied upon. However, a fuller range of operands are available using LOGICSWITCH, SAVEVALUE, and MSAVEVALUE Block Statements interactively.

## Examples

**INITIAL X\$Quote,"Now is the time ... "**

This Command assigns a string constant to the Savevalue Entity, QUOTE.

**INITIAL MX\$Inventory(Part\_39,Stocklevel),200**

This Command assigns the value 200 to the element of Matrix Entity named Inventory with row number of Part\_39, and column number of Stocklevel. The names Part\_39 and Stocklevel must have previously been assigned the appropriate integers in EQU Commands.

**INITIAL MainResult,UNSPECIFIED**

This Command prepares the previously defined Matrix Entity named MainResult for use in an experiment which may have missing data.

## Special Restrictions

- ◆ Operand A must have the form of an LS, X, or MX class SNA, or the name of a Matrix Entity.
- ◆ You cannot use Transaction Parameters in any part of Operand A.
- ◆ You cannot use UNSPECIFIED to initialize a Logicswitch.

## Related SNAs

- ◆ LSEnnum - Logicswitch. The value of Logicswitch Entity *Ennum* is returned.
- ◆ MXEnnum(m,n) - Matrix element value. The value in row m, column n of Matrix Entity *Ennum* is returned.
- ◆ XEnnum - Savevalue. the value of Savevalue Entity *Ennum* is returned.

## Related Blocks

- ◆ LOGIC - assign a value to a Logicswitch Entity.
- ◆ MSAVEVALUE - assign a value to or increment an element of a Matrix Entity.
- ◆ SAVEVALUE - assign a value to or increment a Savevalue Entity.

**An INTEGRATE Command sets up the integration and thresholds of a continuous variable.**

## NAME INTEGRATE A,B,C,D,E

### Operands

**NAME** - User Variable. Required. The field must be *Name*.

**A** - Derivative. Required. The operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, or *SNA*.

**B** - Threshold 1. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, or *SNA*.

**C** - Arrival Block 1. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, or *SNA*.

**D** - Threshold 2. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, or *SNA*.

**E** - Arrival Block 2. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, or *SNA*.

### Examples

Rabbits INTEGRATE (a\_ # Rabbits) - ( b\_ # Rabbits # Foxes)

Foxes INTEGRATE ( - c\_ # Foxes) + ( d\_ # Rabbits # Foxes)

Rabbits EQU 10000

Foxes EQU 1500

a\_ EQU 0.9

b\_ EQU 0.4

c\_ EQU 0.2

d\_ EQU 0.1

This example defines a "Predator-Prey" model relating a rabbit population to a fox population. The Expressions in the INTEGRATE Commands are used as the derivatives with respect to time. The values of the constants in the Expressions, and the initial values of the populations are set by the EQU Commands. When the simulation runs, the integrations are performed automatically when the clock advances.

X\_ INTEGRATE (Y\_),0.707,Wake\_Up

Y\_ INTEGRATE (-X\_)

X\_ EQU 1.0

Y\_ EQU 0.0

This example defines a coupled system of ordinary differential equations (ODEs) whose solution is X\_=cos(Y\_VAR) and Y\_=-sin(X\_VAR). The expressions in the INTEGRATE Commands are used as the derivatives with respect to time. The initial values of the continuous variables are set by the EQU Commands. When the simulation runs, the integrations are performed automatically between discrete time instants.

A threshold is used for the User Variable X\_. When this variable crosses the value 0.707, from either direction, a new Transaction is created and scheduled for the Block labeled WAKE\_UP.

X is an SNA Class, and therefore cannot be used as a Named Value. Here we use X\_. It is always safe to create names if they include an underscore character.

## Action

The INTEGRATE Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

When an INTEGRATE Command is performed, the data structures are set up so that a User Variable will be integrated automatically when the system clock advances. Integrations are done using a modified fifth order Runge Kutta Fehlberg method (RK4(5)), with a variable step size.

All User Variables involved in an integration must be given initial values before the simulation runs. You can do this with EQU Commands or by Assignment Statements in PLUS Procedures.

Operand A of the INTEGRATE Command is used for the derivative of the User Variable with respect to time. It can be very simple or quite complex. In the latter case you may want to define a PLUS Procedure and place a Procedure Call in the parenthesized Expression used for Operand A.

Each INTEGRATE Command may have zero, one, or two numeric thresholds. Operands B and C can be used to specify threshold 1, and/or operands D and E can be used to specify threshold 2. In either case, the first operand of the pair determines the value of the threshold, the second indicates the Block which will receive generated Transactions.

During the integration, if the value of the integrated variable crosses the value of a threshold, from either direction, a new Transaction is created. It is given a priority of 0, and is scheduled to enter the Block associated with that threshold in the INTEGRATE Command. The Transaction's time of entry into the model is estimated by a linear interpolation. To improve accuracy, the integration ministep is decreased when a threshold is imminent.

Thresholds may be constants, parenthesized Expressions, or even Procedure Calls. In addition, the Transactions generated by a threshold crossing may be used to move the threshold.

Integrations are automatically begun in the active, or "enabled" state. However, you can turn an integration on or off while a simulation is running by using one or more INTEGRATION Blocks. This is discussed in Chapter 7.

## Phases

Simulations run in alternate continuous and discrete phases. At any instant where events are scheduled, the simulation runs in a discrete phase. The clock does not advance within an instant in a discrete phase. Between instants, the simulation runs in a continuous phase, during which the integrations proceed in small time increments called ministeps. Plotted integration variables report intermediate values at the end of ministeps.

When a threshold crossing generates a Transaction, the simulation goes into a discrete phase. In this manner, the continuous and the discrete phases can be closely interrelated. Conversely, User Variables can be assigned new values in a discrete phase even if they are being integrated. You can do so using an EQU Command, or a PLUS Assignment Statement. If you want such assignments to occur within the running of the simulation, you must define a PLUS Procedure that makes the assignment. For example, if you defined a PLUS Procedure as

```
PROCEDURE SetPop(PopLevel) BEGIN  
    Foxes = PopLevel ;  
END ;
```

you could reinitialize the Foxes User Variable by entering a PLUS Block, such as

**PLUS (SetPop(200))**

or by using a parenthesized Expression that invokes SetPop() in some other kind of Block.

## Integration Error

A Model Setting called the Integration Tolerance is used to limit the local truncation error of each individual integration. If you make the tolerance smaller, the integrations will take longer, but will be more accurate. This is set in the Simulate Page of the Model Settings Notebook.

**CHOOSE View / Settings / Model**

then select the **Simulate** page. Then fill in the desired value in the entry box marked **Integration Tolerance**. The installation default is 10-6.

Continuous state modeling is also discussed in Chapter 2.

## Related Block

- ◆ INTEGRATION - Enables or disables the integration of a User Variable.

## Related Windows

- ◆ Expressions Window - Online view of values of Expressions.
- ◆ Plot Window - Online view of a plot of up to 8 Expressions.

## Restrictions

- ◆ If either Operand B or Operand C is used, both must be used.
- ◆ If either Operand D or Operand E is used, both must be used.

# MATRIX

A MATRIX Command defines a GPSS Matrix Entity.

**NAME MATRIX A,B,C,D,E,F,G**

## Label / Operands

**NAME** - Entity Label for this entity. Required. The field must be *Name*.

**A** - Unused field (for compatibility with older GPSS implementations).

**B** - Required. Maximum count of elements in first dimension. Number of matrix rows. The operand must be *PosInteger*.

**C** - Required. Maximum count of elements in second dimension. Number of matrix columns. The operand must be *PosInteger*.

**D** - Optional. Maximum count of elements in third dimension. The operand can be *PosInteger*.

**E** - Optional. Maximum count of elements in fourth dimension. The operand can be *PosInteger*.

**F** - Optional. Maximum count of elements in fifth dimension. The operand can be *PosInteger*.

**G** - Optional. Maximum count of elements in sixth dimension. The operand can be *PosInteger*.

## Example

**Inventory MATRIX ,1000,5**

This Command defines a Matrix Entity named Inventory with 1000 rows and 5 columns.

## Action

The MATRIX Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

A MATRIX Command causes a Matrix Entity to be created in the simulation. A Matrix Entity must be defined specifically in a MATRIX Command before it can be referenced.

The A Operand serves no purpose in GPSS World because there is no need to specify the precision of the Matrix elements. This operand is retained only for compatibility with older GPSS versions.

A Matrix Entity can have up to 6 dimensions. However, only the first two dimensions can be accessed in an MSAVEVALUE Block. In this case, all missing coordinates are presumed to be 1.

PLUS Procedures can access all elements of any matrix. If you need to use matrices of more than 2 dimensions, you will have to create one or more PLUS Procedures to access them. Matrices defined in a MATRIX Command have global scope and are known to all PLUS Procedures. In addition, temporary matrices with local scope can be created for the duration of a PLUS Procedure invocation. This is discussed further in Chapter 8.

Matrix Entities are never deleted from the simulation. However, a Matrix Entity may be redefined by another MATRIX Command.

When a Matrix Entity is first created, or when a CLEAR ON Command is used, all elements are given the value of 0. However, you can use the INITIAL Command to give elements the state of UNSPECIFIED. This is useful when you are using a Matrix Entity to hold the results of an experiment. When you pass such a Result Matrix to the ANOVA Library Procedure, the UNSPECIFIED elements will be treated as missing data, instead of results of 0.0.

## Memory Restriction

- ◆ Matrix Entities are limited to the maximum memory request in the Model Setting Notebook. This is discussed in Chapter 2.

## Related SNA

- ◆ MXentnum( $m, n$ ) - Matrix Entity element. The value in row  $m$ , column  $n$  of matrix  $entnum$  is returned. Only names, integers or P class SNAs can be used for row and column values.

## Related Block

- ◆ MSAVEVALUE - assign or increment an element of a Matrix Entity.

## Related Commands

- ◆ INITIAL - initialize a Matrix Entity to 0 or UNSPECIFIED.
- ◆ CLEAR - initialize a Matrix Entity to 0..

## Related Windows

- ◆ A Matrix Entity can be viewed in an online Matrix Window. This window shows a 2 dimensional cross section of any matrix.
- ◆ Any SNA can be viewed in an Expressions or Plot Window.

# QTABLE

**A QTABLE Command initializes a queue time frequency distribution table.**

**NAME QTABLE A,B,C,D**

## Label/Operands

**NAME** - Entity Label for this entity. Required. The field must be *Name*.

**A** - Name of Queue Entity. Required. The operand must be *PosInteger* or *Name*.

**B** - Upper limit of first frequency class. The maximum argument which causes the first frequency class to be updated. Required. The operand must be *Number* or *String*.

**C** - Size of frequency classes. The difference between the upper limit and lower limit of each frequency class. Required. The operand must be *Number* or *String*.

**D** - Number of frequency classes. The operand must be *PosInteger*.

## Action

The QTABLE Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation♦s Command Queue.

By using a QTABLE Command, statistics will be kept automatically for Transactions which enter QUEUE and DEPART Blocks. The Queue Entity being measured is specified in the A Operand of the QTABLE definition Command.

When a Transaction enters a QUEUE Block which refers to a Queue Entity which has one or more Qtables, a special timestamp is created and kept with the Transaction. To collect a data item, a Transaction must enter a DEPART Block which refers to this same Queue Entity. When such a DEPART Block is entered, a table argument is calculated by subtracting the old saved timestamp from the current system time. If the calculated table argument is less than or equal to the B Operand of the QTABLE Command, the first frequency class of the table is chosen. If the table argument does not fit into the first frequency class, the class is chosen by dividing the argument value by the C Operand of the QTABLE Command. The lower limit of a frequency class is included in the previous class. If the table is not large enough to accommodate this value, the last frequency class in the table is chosen.

Then the integer in the chosen frequency class and the total count for all classes is incremented by the B Operand of the DEPART Block. Finally, accumulators for the mean and standard deviation of the Qtable argument are updated.

A Qtable Entity can be redefined by a second QTABLE Command with the same label as the first.

## Example

**WaitTimes QTABLE WaitingLine,100,100,10**

In this simple example, the distribution of the QUEUE-DEPART intervals is entered into the table named WaitTimes. The QTABLE Command creates a table with a total of 10 frequency classes.

All time intervals at or below 100 cause the first frequency class of the table to be updated. Normally this means that the integer for the first frequency class is increased by 1. However, a weighting factor is available in the B Operand of the DEPART Block. This has the effect of adding the weighting factor to the integer for the frequency class. The weighting factor applies to the mean and standard deviation as well, having the same effect as multiple entries into the DEPART Block.

If the time interval of the Transaction is greater than 900, it will be placed in the tenth, and last, frequency class. If the time interval falls in neither the first nor the last frequency class, it is used to choose from the equally spaced frequency classes from the second to the ninth.

For example, if the time interval is 290, the third frequency class would be updated.

The statistics collected in a Qtable Entity are printed in the standard report. A sample report is in Chapter 11.

## Related SNAs

♦ **QEnignum** - Current queue content. The current count value of queue *Enignum*.

♦ **QAEnignum** - Average queue content. The time weighted average count for queue *Enignum*.

- ◆ *QCEnnum* - Total queue entries. The sum of all queue entry counts for queue *Ennum*.
- ◆ *QMEEnnum* - Maximum queue contents. The maximum count (high water mark) of queue *Ennum*.
- ◆ *QTEEnnum* - Average queue residence time. The time weighted average of the count for queue *Ennum*.
- ◆ *QXEnnum* - Average queue residence time excluding zero entries. The time weighted average of the count for queue *Ennum* not counting entries with a zero residence time.
- ◆ *QZEnnum* - Queue zero entry count. The number of entries of queue *Ennum* with a zero residence time.

## Memory Restriction

- ◆ Qtale Entities are limited to the maximum memory request in the Model Setting Notebook. This is discussed in Chapter 2.

## Related Blocks

- ◆ QUEUE - register statistics from the beginning of a waiting time.
- ◆ DEPART - register statistics for the end of a waiting time using an optional weighting factor.

## Related Windows

- ◆ A Qtale Entity can be viewed in an online Table Window.
- ◆ Any SNA can be viewed in an Expressions or Plot Window.

# REPORT

**A REPORT Command causes a report to be created immediately.**

**REPORT A,B**

## Operands

**A** - Must be *Null*.

**B** - NOW, to write a Standard Report, immediately. Optional. the operand must be NOW or *Null*.

## Example

**REPORT**

This Command directs GPSS World to create a Standard Report immediately.

**REPORT ,NOW**

This Command is kept for compatibility purposes. As in the previous example, it causes a Standard Report to be created immediately.

## Action

The REPORT Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

When a Simulation Object encounteres a REPORT command it immediately creaets a Standard Report according to the "In Windows" Setting of the Simulation Object. This can be found on the Reports page of the Settings Notebook.

If the "In Windows" option is set, the Simulation Object creates a new Report Object containing the new Standard Report. It may then be saved in a file or discarded. If "In Window" is not set, the Standard Report is created, given a sequential serial number, and placed in a file.

REPORT no longer uses operand A, and always assumes the NOW Operand is used in Operand B. It ignores the "Create Standard Report" Setting, which is used for automatic Standard Report creation, and it ignores operand B of the START Command used for the current simulation.

You do not normally need to use the REPORT Command. Reports are managed automatically according to the Configuration Settings. This is discussed in Chapter 11.

## Related Setting

- ◆ Standard Reports are normally sent to Report Windows but can be sent directly to file. This is controlled by the "In Window" setting in the Reports Page of the Settings Notebook.

# RESET

**A RESET Command marks the beginning of a measurement period.**

## RESET

### Operands

None.

## Action

The RESET Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

A RESET Command initializes the statistics accumulators without removing Transactions from the simulation. This is useful for replication of simulation experiments, and for throwing away data representing a transient startup period.

RESET sets the measurement period start time to equal the current system clock, and then initializes the statistics accumulators for Facilities, Queues, and Storage Entities. This provides for the start of a new time window used in the gathering of statistics.

A RESET Command does not remove Transactions form the simulation, whereas a CLEAR Command does

RESET has no effect on the random number generators, the system clock or the numbering of Transactions.

When a RESET Command is performed.

- ◆ Space-time products of Facilities, Storages, Queues, and Userchain Entities are set to 0.

- ◆ Total counts are set equal to current counts.
- ◆ Minimum and maximum values are set equal to current content in Queues, Userchains, and Storage Entities.
- ◆ Random number generators are not reset.
- ◆ The Relative Clock (time since last RESET) is set to 0.
- ◆ Table statistics are reset to 0.

# RMULT

**An RMULT Command sets the seeds for random number generators.**

**RMULT A,B,C,D,E,F,G**

## Operands

**A** - Seed for random number generator number 1. Optional. The operand must be *Null*, or *PosInteger*.

**B** - Seed for random number generator number 2. Optional. The operand must be *Null*, or *PosInteger*.

**C** - Seed for random number generator number 3. Optional. The operand must be *Null*, or *PosInteger*.

**D** - Seed for random number generator number 4. Optional. The operand must be *Null*, or *PosInteger*.

**E** - Seed for random number generator number 5. Optional. The operand must be *Null*, or *PosInteger*.

**F** - Seed for random number generator number 6. Optional. The operand must be *Null*, or *PosInteger*.

**G** - Seed for random number generator number 7. Optional. The operand must be *Null*, or *PosInteger*.

## Example

**RMULT „111**

In this example, random number generator 3 is initialized with a seed of 111.

## Action

The RMULT Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

The RMULT Command initializes up to 7 random number generators with new seeds. Only random number generators numbered 7 or below can be controlled by an RMULT Command. However, since the default seed of a random number generator is its entity number, you can in fact have any number of random number generators, and you can choose the initial seed of each. After the first simulation run the seeds can only be controlled easily with the RMULT Command on generators 1-7.

You can select which random number stream GPSS World uses it to calculate random time increments in ADVANCE and GENERATE Blocks, in Fractional Mode TRANSFER and TRANSFER PICK Blocks and to resolve the scheduling of time ties. This is set in the "Random" page of the Model Settings Notebook.

### CHOOSE View / Settings / Model

then select the **Random** page. Then fill in the desired Random Number Stream Entity number in the corresponding entry boxes. The installation default is to use Random Number Stream number 1 in each case.

Since GPSS World uses multiplicative congruential random number generators, it is possible to control pairs of random number streams for variance reduction purposes. This is done by matching the seeds of the random number generators. If  $s$  is the seed for the first random number generator, then a choice of  $2147483647-s$  for the seed of the second random number generator will generate a stream of random numbers with antithetic properties.

### Special Restriction

- ◆ Random number seeds must be positive integers.

### Related SNA

- ◆ *RNEnignum* - Random number. *RNEnignum* returns a random integer 0-999 from the random number generator *Enignum*.

### Related Window

- ◆ The ANOVA Window can perform Analysis of Variance and calculate confidence intervals.

# SHOW

**A SHOW Command sends an Expression for evaluation by the Simulation Object, and writes the result in the Status Line.**

### SHOW X

#### Operand

X - Expression.

#### Example

**SHOW 2#LOG(Q\$Barber)**

This Command finds the natural logarithm of the SNA Q\$Barber, doubles it, and writes the result in the Status Line of the Model Window.

#### Action

The SHOW Command is an *Immediate Command*. It is performed when received by the Simulation Object.

The SHOW Command evaluates an Expression in the context of the simulation, and writes the result in the Status Line of the Model Window. Additional messages are sent to any open Journal Windows.

The rules for evaluating Expressions may be found in Section 3.4. Expressions must be well-formed according to the rules of elementary algebra. You may use any of the arithmetic and logic operators listed in Section 3.4.

If SNAs are used in the Expression field, they are evaluated with respect to the Active Transaction. If there is no Active Transaction because no simulation has been started, an error message will be written. Names which have not been explicitly assigned values cannot be used in Expressions. To do so you must assign a value by an EQU Command before the Expression is evaluated.

### Examples

**SHOW C1**

shows the system clock in the Status Line.

**SHOW 4#(SQR(2)+SIN(C1))**

shows the result of 4 times the sum of the square root of 2 and the sine of the system clock.

**SHOW N1^W\$Chair**

shows the result of raising the number of Transactions to have entered Block 1 to the power of the number of Transactions waiting at the Block named Chair.

### Special Restrictions

- ◆ Some SNAs cannot be evaluated unless there is an Active Transaction.
- ◆ SHOW displays up to 38 characters or digits.

### Related Windows

- ◆ Any valid Expression can be viewed in an Expressions or Plot Window.

# START

**A START Command begins a simulation.**

**START A,B,C,D**

### Operands

**A** - Termination count. Required. The operand must be *PosInteger*.

**B** - Printout operand. NP for "no printout". Default is to print a standard report. Optional. The operand must be *NP* or *Null*.

**C** - Not used. Kept for compatibility with older dialects of GPSS.

**D** - Chain printout. 1 to include the CEC and FEC in the standard report. Optional. The operand must be *Null*, or *PosInteger*.

### Action

The START Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

The START Command is used to set up and begin the actual simulation. The simulation does not end until the termination count, which is set by Operand A, reaches zero or becomes negative. TERMINATE Blocks are used to reduce the termination count.

The simulation may stop short if a HALT Command is issued, or if a Stop Condition is detected.

Operands B and D are used to control automatic report generation. If Operand B is not used, a standard report is written. If Operand B is NP, which stands for "no printout", no standard report is written. If Operand D is nonzero, the Current Events Chain (CEC) and the Future Events Chain (FEC) are reported. Otherwise they are not reported. A further discussion of the control and contents of the standard report may be found in Chapters 11 and 12.

Operand C is kept for compatibility with older versions of GPSS. It was used as a "snap" count to put out a report periodically. This Function is available by using more than one START and REPORT Command.

When a START Command is performed:

- ◆ The Termination Count is set.
- ◆ Any generate Blocks marked "not started" are primed with a single Transaction.
- ◆ Random number generators are not reset.
- ◆ If time is 0, a RESET of the statistics accumulators is performed. See the discussion of the RESET Command in this chapter.

## Example

**START 1000,,,1**

In this example, the Termination Count is set to 1000 and the simulation is started. When the Termination Count reaches zero or becomes negative (by reduction due to TERMINATE Blocks in the simulation), a standard report is written which includes information on the Current Events Chain (CEC) and the Future Events Chain (FEC).

## Related SNA

- ◆ TG1 - Termination count.

## Related Blocks

- ◆ TERMINATE - destroy Transaction and optionally reduce the termination count.

# STEP

**A STEP Command causes the simulation to proceed a specified number of Block entries.**

**STEP A**

## Operand

**A** - Block entry count. Required. A must be a positive integer, more formally, *PosInteger*.

## Example

### STEP 1

This Command causes the simulation to proceed exactly one Block entry and then to stop. The simulation is then said to be in the "Halted" state.

## Action

The STEP Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

The STEP Command causes the Simulation Object to simulate the specified number of Block entries. When a simulation reaches the required number of Block entries, the Simulation Object sends a message to the Status Line, and any Journal Windows that are open. The message gives the time, the Active Transaction number, the current Block of the Active Transaction, and the next scheduled Block of the Active Transaction.

A simulation started by a STEP Command does not end when the termination count goes to zero, it ends when the required number of Block entries have occurred.

When a STEP Command is performed:

- ◆ The termination count is not set.
- ◆ Any generate Blocks marked "not started" are primed.
- ◆ Random number generators are not reset.

## Hot Key

A STEP 1 Command can be sent to the simulation by pressing the **[Ctrl] + [Alt] + [1]** key combination. A GPSS World window must have the input focus.

## Related Windows

- ◆ The stepping of simulations can be viewed dynamically in the Blocks Window.

# STOP

**A STOP Command sets or removes a Stop Condition.**

## STOP A,B,C

### Operands

**A** - Transaction number. A must be a positive integer. If the A Operand is omitted, any Transaction number will satisfy the condition. Optional. The operand must be *Null*, or *PosInteger*.

**B** - Block number. If the B Operand is omitted, any Block will satisfy the condition. Optional. The operand must be *Null*, *Name*, or *PosInteger*.

**C** - ON or OFF. If the C Operand is omitted, ON is assumed. Optional. The operand must be ON, OFF or *Null*.

## Example

## **STOP 100,52**

This Command inserts a Stop Condition which will cause a running simulation to stop when Transaction number 100 attempts to enter Block number 52.

### **Action**

The STOP Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

The STOP Command with the ON option inserts a Stop Condition into the simulation but does not cause the simulation to start. A subsequent START, STEP, or CONTINUE must be entered.

When a simulation reaches a Stop Condition, the Simulation Object sends a message to the Status Line, and any Journal Windows that are open. The message gives the time, the Active Transaction number, the current Block of the Active Transaction, and the next scheduled Block of the Active Transaction.

When a Stop Condition is detected, established Stop Conditions remain in force. If a CONTINUE Command is issued, the CONTINUE Command skips the original Stop Condition but does not remove it. If the same condition occurs again, the simulation will stop again. Stop Conditions must be removed explicitly by entering a new STOP Command containing the OFF option. They are also removed when a model is Translated.

STOP Commands can be sent by using mousing operations in the Blocks Window. This is discussed in more detail in Chapter 5.

If you skip the A Operand, any Transaction will satisfy the Stop Condition. If you skip the B Operand, any Block will satisfy the Stop Condition. A STOP Command with no operands will cause the simulation to stop immediately.

Any number of Stop Conditions may be established.

The STOP Command with the OFF option removes any existing Stop Conditions that satisfy the Transaction and Block conditions specified by operands A and B.

### **Examples**

#### **STOP**

With no operands, the STOP Command will cause any subsequent simulation to stop immediately.

#### **STOP „OFF**

This Command removes all Stop Conditions from the simulation.

#### **STOP 2**

This Command will cause the simulation to stop when Transaction 2 becomes the Active Transaction.

#### **STOP ,Chair**

This Command will cause the simulation to stop when any Transaction attempts to enter the Block at location CHAIR.

#### **STOP ,Chair,OFF**

This Command will remove all Stop Conditions that specify Chair as the Block name.

### **Related Windows**

- ◆ Mouse operations can be used in the Blocks Window to place and remove Stop Conditions.

# STORAGE

A STORAGE Command defines a Storage Entity.

**NAME STORAGE A**

## Label / Operand

**NAME** - Entity Label for this entity. Required. The field must be *Name*.

**A** - Total storage capacity. Required. The operand must be *PosInteger*.

## Example

**MotorPool STORAGE 20**

This Command defines a Storage Entity named MotorPool with a total capacity of 20 units.

## Action

The STORAGE Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

A STORAGE Command defines a Storage Entity in the simulation. When a Transaction attempts to enter an ENTER Block, its storage demand is compared with the available storage at the Storage Entity. If the demand can be granted, the Transaction is allowed to enter the ENTER Block and the available storage capacity of the Storage Entity is reduced. If the Transaction's demand cannot be satisfied, the Transaction comes to rest in the simulation on the Delay Chain of the Storage Entity. Storage Entities are explained in more detail in Chapter 4.

A Storage Entity may be redefined by entering a new STORAGE Command with the same label as the old one.

If you must refer to Storage Entities by number, not name, an EQU Command should precede the STORAGE definition. This is necessary if you wish to reference a range of STORAGES in a SELECT or COUNT Block.

## Related SNAs

- ◆ **REnnum** - Unused storage capacity. The storage content (or "token" spaces) available for use by entering Transactions at storage *Ennum*.
- ◆ **SEnnum** - Storage in use. *SEnnum* returns the amount of storage content (or "token" spaces) currently in use by entering Transactions at storage *Ennum*.
- ◆ **SAEnnum** - Average storage in use. *SAEnnum* returns the time weighted average of storage capacity (or "token" spaces) in use at storage *Ennum*.
- ◆ **SCEnnum** - Storage use count. Total number of storage units that have been entered in (or "token" spaces that have been used at) storage *Ennum*.
- ◆ **SEEnnum** - Storage empty. *SEEnnum* returns 1 if storage *Ennum* is completely unused, 0 otherwise.
- ◆ **SFEnnum** - Storage full. *SFEnnum* returns 1 if storage *Ennum* is completely used, 0 otherwise.

- ◆ **SREntnum** - Storage utilization. The fraction of total usage represented by the average storage in use at storage *Entnum*. **SREntnum** is expressed in parts-per-thousand and therefore returns an real value 0-1000, inclusively.
- ◆ **SMEntnum** - Maximum storage in use at storage *Entnum*. The "high water mark".
- ◆ **STEntnum** - Average holding time per unit at storage *Entnum*.
- ◆ **SVEntnum** - Storage in available state. **SVEntnum** returns 1 if storage *Entnum* is in the available state, 0 otherwise.

## Related Blocks

- ◆ **ENTER** - take or wait for available storage space.
- ◆ **LEAVE** - release storage space for use by other Transactions.

## Related Windows

- ◆ GPSS storage entities are visible in the Storages Window.
- ◆ Any SNA can be viewed in an Expressions or Plot Window.

# TABLE

**A TABLE Command initializes a frequency distribution table.**

**NAME TABLE A,B,C,D**

### Label / Operands

**NAME** - Entity Label for this entity. Required. The field must be *Name*. The length of a Table name is limited to 32 characters.

**A** - Table argument. Optional. The data item whose frequency distribution is to be tabulated. The operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, or *SNA*. Ignored by ANOVA, but must be specified when used by TABULATE Blocks.

**B** - Upper limit of first frequency class. The maximum argument which causes the first frequency class to be updated. Required. The operand must be *Number* or *String*.

**C** - Size of frequency classes. The difference between the upper limit and lower limit of each frequency class. Required. The operand must be *Number* or *String*.

**D** - Number of frequency classes. Required. The operand must be *PosInteger*.

### Action

The TABLE Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

A TABLE Command is used to define a GPSS Table Entity for collecting a frequency distribution, its cumulative relative frequencies, its mean, and its standard deviation.

In addition to general purpose data collection, a Table Entity can be used to tabulate the residuals in an Analysis of Variance. This specialized use is discussed below.

To collect a data item, a Transaction must enter a TABULATE Block which refers to this TABLE. When a TABULATE Block is entered, the table argument (Operand A in the TABLE Command) is evaluated. If it is less than or equal to the B Operand of the TABLE Command, the first frequency class of the table is chosen. If the table argument does not fit into the first frequency class, the class is chosen by dividing the argument value by the C Operand of the TABLE Command. The lower limit of a frequency class is included in the previous class. If the table is not large enough to accommodate this value, the last frequency class in the table is chosen.

Then the integer in the chosen frequency class and the accumulated count is incremented by the B Operand of the TABULATE Command. The default increment is 1.

Finally, accumulators for the mean and standard deviation of the table are updated.

A Table Entity can be redefined or reinitialized by a second TABLE Command with the same label as the first.

## Example

**SalesTable TABLE P\$Price,9.95,10,10**

In this simple example, the distribution of the Price Parameter of Transactions is entered into the table named SalesTable. The TABLE Command creates a table with a total of 10 frequency classes.

All Price values at or below 9.95 cause the first frequency class of the table to be updated. Normally, this means that the integer for the first frequency class is increased by 1. However, a weighting factor is available in the B Operand of the TABULATE Block. This has the effect of adding the weighting factor to the current integer value in the appropriate frequency class. The weighting factor applies to the mean and standard deviation as well, having the same effect as multiple entries into the TABULATE Block.

If the value of the Price Parameter is greater than 89.95, the tenth, and last, frequency class will be updated. If the value of the Price Parameter falls in neither the first nor the last frequency class, it is used to choose from the equally spaced frequency classes from the second to the ninth.

For example, if the Price value is 29.49, the third frequency class would be updated.

The statistics collected in a GPSS Table Entity are printed in the standard report. An example is in Chapter 11.

## ANOVA Residuals

When you pass a named Results Matrix to the ANOVA Library Procedure, you can cause residuals of the Analysis to be tabulated automatically. All that you need to do is to have defined a Table Entity with the same name as the Matrix Entity with a suffix of "\_RESIDUALS". For example, if you intend to use

**ANOVA( MainResult, 3, 1 )**

you can have the residuals tabulated automatically by defining a Table Entity such as

**MainResult\_Residuals TABLE ,-5,.5.20**

prior to the invocation of ANOVA.

When you do this, the ANOVA Procedure ignores Operand A of TABLE and tabulates residuals arising from the analysis of the Result Matrix. The ANOVA Table and the descriptive cell statistics appear in the Journal Window. Then, you can examine the residuals by opening a Table Window on the MainResult\_Residuals Table.

## Memory Restriction

◆ Table Entities are limited to the maximum memory request in the Model Settings Notebook. This is discussed in Chapter 2.

## Related SNAs

- ◆ **TB $E$ ntrum** - Nonweighted average of entries in table  $Entnum$ .
- ◆ **TCE $ntrum$**  - Count of nonweighted table entries in table  $Entnum$ .
- ◆ **TDE $ntrum$**  - Standard deviation of nonweighted table entries in table  $Entnum$ .

## Related Blocks

- ◆ **TABULATE** - register statistics for a data item in a Table Entity.

## Related Windows

- ◆ A Table can be viewed in the online Table Window.
- ◆ Any SNA can be viewed in an Expressions or Plot Window.

# VARIABLE

**A VARIABLE Command defines an arithmetic Variable Entity.**

### NAME VARIABLE X

#### Label / Operand

**NAME** - Entity Label for this entity. Required. The field must be *Name*.

**X** - Expression. Required. Expressions are discussed in Section 3.4.

#### Example

**Var1 VARIABLE 5#LOG(Q\$WaitingLine)**

This Command defines a Variable Entity which is to be evaluated when a V\$Var1 is encountered. When this SNA is evaluated, the Expression defining the Variable Entity named Var1 is evaluated and returned as the result.

#### Action

The VARIABLE Command is a *Queued Command*. When the Simulation Object receives one, it places it at the end of the simulation's Command Queue.

When a VARIABLE Command is performed, the Simulation Object creates or redefines a GPSS Variable Entity. The Variable Entity so created is evaluated when a System Numerical Attribute in the V class, referring to this particular entity, is encountered by the Simulation Object.

The Expression in a VARIABLE Command is evaluated according to the rules in Chapter 3, and may include calls to PLUS Procedures or to Library Procedures. The evaluation proceeds differently in GPSS/PC Compatibility Mode.

Expressions must be well-formed according to the rules of elementary algebra. A formal definition can be found in the Appendix. You may use any of the arithmetic and logic operators listed in Section 3.4. If SNAs are used in the Expression field, they are evaluated with respect to the Active Transaction. A Named Value which has not been explicitly assigned a value cannot be used as an item in an Expression. To do so, you must assign a value to it before the Expression is evaluated. Assignments to User Variables are done by EQU Commands or in PLUS Procedures.

Once a Variable Entity is created in a simulation, it is never destroyed. However, it may be redefined later by an interactive VARIABLE Command.

If the simulation is not run in GPSS/PC Compatibility Mode, FVARIABLE and VARIABLE Commands are treated the same.

## GPSS/PC Compatibility

- ◆ All SNAs are truncated in GPSS/PC Compatibility Mode.
- ◆ In Variable Entity evaluation in GPSS/PC Compatibility Mode, the intermediate results are truncated.

## Related SNA

- ◆ *VEnnum* - Returns the result of evaluating an arithmetic Variable Entity *Ennum*.

## Related Windows

- ◆ Any SNA can be viewed in an Expressions or Plot Window.

[\[Table of Contents\]](#)

# Chapter 7 - Block Statements

You use Block Statements to create GPSS Block Entities. Block Statements which are part of the Initial Model Translation create permanent Blocks in the simulation. A Block Statement sent to an existing simulation creates a one-time temporary Block to be used in a mode called *Manual Simulation*. This is discussed in more detail in Section 2.3.

A model is simply a sequence of Model Statements. A GPSS World Model Statement is either a GPSS Statement or a PLUS Procedure definition. The GPSS Statements, in turn, are either Block Statements, which cause a Block to be created, or Commands, which do not. Any Model Statement can be sent to an existing simulation as an Interactive Statement.

This chapter contains reference information for each Block Statement supported by GPSS World.

The GPSS Block statements are:

**ADOPT** - Change Assembly Set.

**ADVANCE** - Place Transaction on Future Events Chain.

**ALTER** - Test and modify Transactions in a Group.

**ASSEMBLE** - Wait for and destroy related Transactions.

**ASSIGN** - Modify Transaction Parameter.

**BUFFER** - Place Transaction at end of the Current Events Chain.

**CLOSE** - End the Data Stream.

**COUNT** - Place count of entities into a Transaction Parameter.

**DEPART** - Decrement content of a Queue Entity.

**DISPLACE** - Change the Next Sequential Block of a Transaction.

**ENTER** - Occupy or wait for storage units in a Storage Entity.

**EXAMINE** - Test group membership.

**EXECUTE** - Perform action specified by a different Block.

**FAVAIL** - Change status of a Facility Entity to "available".

**FUNAVAIL** - Change status of a Facility Entity to "not available".

**GATE** - Test entity and modify Transaction flow.

**GATHER** - Wait for related Transactions.

**GENERATE** - Create Transaction and place on Future Events Chain.

**INDEX** - Modify Transaction Parameter.

**INTEGRATION** - Turn the integration of a User Variable On or Off.

**JOIN** - Place a member into a Numeric or Transaction Group.

**LEAVE** - Release storage units of a Storage Entity.

**LINK** - Move Transaction to Userchain Entity.

**LOGIC** - Modify Logicswitch Entity.

**LOOP** - Decrement Parameter, jump to different Block if result is nonzero.

**MARK** - Place value of system clock into Transaction Parameter.

**MATCH** - Wait for related Transaction to reach conjugate MATCH Block.

**MSAVEVALUE** - Assign value to Matrix Entity element.

**OPEN** - Initialize a Data Stream.

**PLUS** - Evaluate PLUS Expression and save result in Parameter.

**PREEMPT** - Displace Facility owner.

**PRIORITY** - Modify Transaction priority.

**QUEUE** - Increment content of a Queue Entity.

**READ** - Bring the next line of data from a Data Stream.

**RELEASE** - Free Facility Entity.

**REMOVE** - Take a member out of Numeric or Transaction Group.

**RETURN** - Free Facility Entity.

**SAVAIL** - Change status of Storage Entity to "available".

**SAVEVALUE** - Assign a value to Savevalue Entity.

**SCAN** - Test Transaction group, place value in Parameter.

**SEEK** - Change the line pointer in a Data Stream.

**SEIZE** - Assume ownership of or wait for a Facility Entity.

**SELECT** - Place selected entity number into Transaction Parameter.

**SPLIT** - Create related Transaction.

**SUNAVAIL** - Change status of Storage Entity to "not available".

**TABULATE** - Update Table Entity.

**TERMINATE** - Destroy Transaction, decrement Termination Count.

**TEST** - Test arithmetic condition and modify Transaction flow.

**TRACE** - Set Trace Indicator of the Active Transaction.

**TRANSFER** - Move to specified Block.

**UNLINK** - Remove Transaction from Userchain Entity.

**UNTRACE** - Turn off Trace Indicator in the Active Transaction.

**WRITE** - Send a value to a Data Stream.

## Operands

Statements usually have one or more operands which you must fill in. Most operands have several different forms which are valid. In the descriptions which follow, a valid class of operands may be described by an italicized word. You must choose a member of the class and type it into the operand field. For example, if one of the valid forms of an operand is given as *PosInteger*, you could type:

**21**

The italicized words are usually suggestive, but you may need to refer to the formal definition in the Appendix.

## Windows

A wide variety of windows are available for you to observe the state of Blocks and their effects on the other entities in your simulations. In general, windows are specialized by the entity type.

◆ Model Window - Text View ◆ Full screen textual model editor.

◆ Block Input Window - Drag and Drop model building.

- ◆ Journal Window - Record session events.
- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Expressions Window - Online view of values of expressions.
- ◆ Facilities Window - Online view of Facility Entity dynamics.
- ◆ Logicswitches Window - Online view of Logicswitch Entity dynamics.
- ◆ Matrix Window - Online view of the dynamics of a Matrix cross-section.
- ◆ Plot Window - Online view of a plot of up to 8 expressions.
- ◆ Queues Window - Online view of Queue Entity dynamics.
- ◆ Savevalues Window - Online view of Savevalue Entity dynamics.
- ◆ Storages Window - Online view of Storage Entity dynamics.
- ◆ Table Window - Online view of Table Entity dynamics.
  
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.
- ◆ Numeric Groups Snapshot - Picture of the state of the Numeric Groups in the simulation.
- ◆ Userchains Snapshot - Picture of the state of the User Chains in the simulation.
- ◆ Transaction Groups Snapshot - Picture of the state of the Transaction Groups in the simulation.

# ADOPT

**ADOPT Blocks are used to change the Assembly Set of the Active Transaction.**

## ADOPT A

### Operand

**A** - Assembly Set. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

#### ADOPT 2000

The numerical Assembly Set value of the Active Transaction is given the value 2000. In effect, the Active Transaction becomes a member of Assembly Set 2000.

## Action

When a Transaction enters an ADOPT Block, Operand A is evaluated numerically, and truncated. If the result is less than or equal to zero, an Error Stop occurs. Otherwise, the result is assigned to the Assembly Set value of the Active Transaction.

Every Transaction is assigned to an Assembly Set when it is created. For Transactions created in GENERATE Blocks, the initial assignment uses the same number for the Assembly Set as was used for the Transaction Number. For Transactions created in SPLIT Blocks, the offspring Transactions are placed in the same Assembly Set as their parent Transaction.

Assembly Sets are used to combine related Transactions in ASSEMBLE and GATHER Blocks. The ADOPT Block provides for easy control of Assembly Set assignments.

## Special Restriction

- ◆ A must be positive.

## Refuse Mode

A Transaction is never refused entry to an ADOPT Block.

## Related Blocks

- ◆ ASSEMBLE - Wait for and destroy Assembly Set members.
- ◆ GATHER - Wait for Assembly Set members.
- ◆ MATCH - Wait for Assembly Set member.
- ◆ SPLIT - Create Transaction(s) in the same Assembly Set.

## Related SNAs

- ◆ A1 - Assembly Set. Return the Assembly Set of the Active Transaction.
- ◆ MBEnignum - Match at Block. *MBEnignum* returns a 1 if there is a Transaction at Block *Enignum* which is in the same Assembly Set as the Active Transaction. *MBEnignum* returns a 0 otherwise.

## Related Window

- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.

# ADVANCE

An ADVANCE Block delays the progress of a Transaction for a specified amount of simulated time.

## ADVANCE A,B

### Operands

A - The mean time increment. Required. The operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**B** - The time half-range or, if a function, the function modifier. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Example

### ADVANCE 101.6,50.3

This example creates a Block which chooses a random number between 51.3 and 151.9, inclusively (i.e. 101.6 plus or minus 50.3), and delays the entering Transaction that amount of simulated time.

## Action

An ADVANCE Block calculates a time increment and places the entering Transaction on the Future Events Chain (FEC) for that amount of simulated time.

The time increment can be calculated in several ways. If only the A Operand is specified, it is evaluated and used as the time increment. If the A and B operands are present, and B does not specify a function, both A and B are evaluated numerically and a random number between A-B and A+B, inclusively, is used as the time increment. You can select which random number generator number is to be used as the source of the random number. This is set in the "Random" page of the Model Settings Notebook.

### CHOOSE View / Settings / Model

then select the **Random** page. Then fill in the desired random number stream entity number in the entry box marked "ADVANCE". The installation default is to use Random Number Stream number 1.

If B is an FN class SNA, called a Function Modifier, the evaluating B is multiplied by the result of evaluating the A Operand; the product is used as the time increment.

If zero is calculated as the time increment (ADVANCE 0), the entering Transaction is placed on the Current Events Chain in front of priority peers. Such a Block then behaves as a null operation. A further discussion of the Current Events Chain can be found in Chapter 9.

## Special Restriction

- ◆ If a negative number is calculated as the time increment, an Error Stop occurs.

## Refuse Mode

Normally, Transactions are not refused entry to an ADVANCE Block. However, since preempted Transactions are not permitted to exist on the Future Events Chain, a preempted Transaction will not be permitted to enter an ADVANCE Block if the time increment is positive (nonzero).

When a Transaction is refused entry, its Delay Indicator is set and remains so until the Transaction enters a "Simultaneous" Mode TRANSFER Block. To coordinate the state of multiple entities, it is better to use a TEST Block and BOOLEAN VARIABLE instead of a TRANSFER SIM Block.

## Related Windows

- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# ALTER

## An ALTER Block changes the priority or a Parameter of selected members of a Transaction Group.

**ALTER O A,B,C,D,E,F,G**

### Operands

**O** - Conditional operator. Relationship of E to F for the alteration to occur. These choices are explained below. Optional. The operator must be *Null*, E, G, GE, L, LE, MAX, MIN, or NE.

**A** - Transaction Group. Group whose members will be tested for alteration. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Limit. The maximum number of Transactions to be altered. The default is ALL. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**C** - Altered attribute. The member Transaction Parameter to be altered, or PR to alter the member Transaction priority. The operand must be PR, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**D** - Replacement value. The value which will replace attribute C. Required. The operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**E** - Test value. PR or Parameter number. The member Transaction Parameter which determines whether each Group member Transaction should be altered, or PR to use the Transaction priority for the determination. It is evaluated with respect to the Transaction Group member. Optional. The operand must be PR, *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**F** - Reference value. The value against which the E Operand is compared. It is evaluated with respect to the Active Transaction. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**G** - Alternate Block number. The alternate destination for the Active Transaction. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Examples

**ALTER Inventory,ALL,Price,49.95**

In this simple example, all Transactions in the Transaction Group named Inventory have their Parameter named Price set equal to 49.95.

**ALTER NE Bin7,10,Price,49.95,PartNum,99.95,Out**

In this example, when a Transaction enters the ALTER Block, Transaction Group named Bin7 is scanned for Transactions which do not have a value of 99.95 in their Parameter named PartNum. The first 10 Transactions which meet the test have the value of their Parameter named Price set to 49.95. If 10 Transactions cannot be found which pass the test, the entering Transaction attempts to enter the Block labeled Out. Otherwise, it proceeds to the Next Sequential Block.

### Action

An ALTER Block selects Transactions from a Transaction Group and alters one of the attributes of each of these Transactions. When a Transaction Group member is altered, its Transaction attribute specified by the C Operand is given the value specified by the D Operand. Altered Transactions are not displaced from their context. However, the Transaction entering the ALTER Block may be redirected according to the G Operand.

If you do not use a conditional operator, or operands E or F, all Transactions up to the limit (Operand B) are altered. In this case, no priority or Parameter test is made to determine whether or not to alter the attribute of the member Transaction.

If you use operands E, F, or a conditional operator, each Group member must pass a test before it is altered. Operand E specifies which attribute of the member Transactions is to be tested. It may be compared to the minimum or the maximum of all such Group member attributes by using MIN or MAX

as the conditional operator. All Transactions which are tested and have the maximum or minimum attribute are altered. In this case, you must not use Operand F.

You may use a conditional operator to specify the relationship between the Transaction attribute (Operand E) and the reference value (Operand F) which will initiate the alteration of the Transaction. The default for the conditional operator is E for equality. If you use no conditional operator, but you use operand E and Operand F, the values must be equal for the member Transaction attribute to be altered.

You may compare the Group member attribute to Operand F, with or without a conditional operator. In this case, the conditional operator must not be MIN or MAX. Operand E always refers to the Group member under test. However, if any other operand is a Transaction related SNA, it is evaluated with respect to the entering Transaction.

The B Operand cuts off the Group scan when it equals the number of Transactions that have been altered. The default is ALL. If there is no attribute test, that is, if E is not specified, Transactions are altered until the alteration count equals B or until the entire Group has been tested.

The G Operand indicates an alternate destination Block to be taken by the entering Transaction when a special condition occurs. The G Operand is used for the next Block under the following conditions:

- ◆ No Transaction is altered.
- ◆ The count of altered Transactions specified by B cannot be reached.

If the G Operand is not used, the entering Transaction always goes to the Next Sequential Block.

## Conditional Operators

The conditional operator may be E, G, GE, L, LE, MAX, MIN, or NE. If no conditional operator is used, E (equality) is assumed. When the condition is true, the Transaction being tested is altered. The conditions are defined as follows:

- ◆ E - The member Transaction attribute specified by Operand E must be equal to the reference value specified by Operand F for the member Transaction to be altered.
- ◆ G - The member Transaction attribute specified by Operand E must be greater than the reference value specified by Operand F for the member Transaction to be altered.
- ◆ GE - The member Transaction attribute specified by Operand E must be greater than or equal to the reference value specified by Operand F for the member Transaction to be altered.
- ◆ L - The member Transaction attribute specified by Operand E must be less than the reference value specified by Operand F for the member Transaction to be altered.
- ◆ LE - The member Transaction attribute specified by Operand E must be less than or equal to the reference value specified by Operand F for the member Transaction to be altered.
- ◆ MAX - The member Transaction attribute specified by Operand E must be equal to the largest such attribute of all Transactions in the Group for the member Transaction to be altered.
- ◆ MIN - The member Transaction attribute specified by Operand E must be equal to the smallest such attribute of all Transactions in the Group for the member Transaction to be altered.
- ◆ NE - The member Transaction attribute specified by Operand E must be unequal to the reference value specified by Operand F for the member Transaction to be altered.

If no conditional operator is used, E is assumed.

## Special Restrictions

- ◆ If Operand E is used, then you must use Operand F or else you must use the conditional operator MIN or MAX.
- ◆ If Operand F is used, you must use Operand E, and not MIN or MAX.

- ◆ If MIN or MAX is used for the conditional operator, Operand E must be used and Operand F must not be used.

## Refuse Mode

A Transaction is never refused entry to an ALTER Block.

## Related Blocks

Transactions are added to Transaction Groups, and numbers are added to Numeric Groups by JOIN Blocks. Transactions in Groups can be referenced by ALTER, EXAMINE, REMOVE, and SCAN Blocks. Numbers in Numeric Groups can be referenced by EXAMINE and REMOVE Blocks.

## Related SNA

- ◆ *GTEntnum* - Transaction Group count. *GTEntnum* returns the membership count of Transaction Group *Entnum*.

## Related Windows

- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ Transaction Groups Snapshot - Picture of the state of the Transaction Groups in the simulation.

# ASSEMBLE

**Wait for and destroy related Transactions.**

## ASSEMBLE A

### Operand

**A** - Transaction count. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

## ASSEMBLE 2

This is the simplest way to use the ASSEMBLE Block. The first Transaction of an Assembly Set (see Section 9.3) is caused to wait when it enters an ASSEMBLE Block. When another Transaction in the same Assembly Set enters the Block, it is destroyed and the waiting Transaction is allowed to continue.

### Action

When a Transaction enters an ASSEMBLE Block, the Match Chain of the Block is searched for a waiting Transaction of the same Assembly Set. If there are no other members of the same Assembly Set present, the A Operand is evaluated, truncated, decremented by one, and saved in a storage location in the Transaction. If this number is zero, the Transaction immediately attempts to enter the Next Sequential Block. Otherwise the Transaction is placed on a queue attached to the ASSEMBLE Block called the Match Chain to await the arrival of other members of its Assembly Set.

When a Transaction enters an ASSEMBLE Block, if a waiting Transaction is found, the entering Transaction is destroyed and the Transaction count that was saved in the chained Transaction is

reduced by one. When this count becomes 0, the waiting Transaction is removed from the Match Chain. If this Transaction has not been preempted at any Facility, it attempts to enter the Next Sequential Block. When it does so, it is scheduled behind active Transactions of the same priority.

Preempted Transactions which complete an assembly at an ASSEMBLE Block are not permitted to leave the Block until all preemptions have been cleared. More discussion of the preemption mechanism can be found in Section 9.4. Preempted Transactions which have been removed from the Match Chain do not participate in later assemblies even though they remain in the ASSEMBLE Block.

ASSEMBLE Blocks differ from GATHER Blocks in that succeeding Transactions are destroyed at an ASSEMBLE.

## Special Restriction

- ◆ Transactions which are currently preempted are not permitted to leave ASSEMBLE Blocks.

## Refuse Mode

A Transaction is never refused entry to an ASSEMBLE Block.

## Related Blocks

- ◆ ADOPT - Set the Assembly Set of the Active Transaction.
- ◆ GATHER - Wait for members of an Assembly Set.
- ◆ MATCH - Wait for Assembly Set member.
- ◆ SPLIT - Create Transactions in the same Assembly Set.

## Related SNAs

- ◆ A1 - Assembly Set. Return the Assembly Set of the Active Transaction.
- ◆ MBEnignum - Match at Block. MBEnignum returns a 1 if there is a Transaction at Block Enignum which is in the same Assembly Set as the Active Transaction. MBEnignum returns a 0 otherwise.

## Related Windows

- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# ASSIGN

**ASSIGN Blocks are used to place or modify a value in a Transaction Parameter.**

## ASSIGN A,B,C

### Operands

A - Parameter number of the Active Transaction. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*, followed by +, -, or Null.

**B** - Value. Required. the operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**C** - Function number. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

## Examples

### ASSIGN 2000,150.6

This is the simplest way to use the ASSIGN Block. The value 150.6 is assigned to Parameter number 2000 of the entering Transaction. If no such Parameter exists, it is created.

### ASSIGN TEXT,"Look on my works, ye Mighty, and despair."

In this example, a string is assigned to the Parameter of the Active Transaction named TEXT. If no such Parameter exists, it is created.

### ASSIGN 2000+,-3

In this example, the [+] following Operand A indicates that the value of Operand B is to be added to the original Parameter value. This Block adds a -3 to the value contained in Transaction Parameter 2000. If there is no such Transaction Parameter, one is created and initialized to 0 before the addition. In this case, the value of the Transaction Parameter becomes -3.

### ASSIGN 2000-,-3

In this example, the [-] following Operand A indicates that the value of Operand B is to be subtracted from to the original Parameter value. This Block subtracts a -3 from the value contained in Transaction Parameter 2000. If there is no such Transaction Parameter, one is created and initialized to 0 before the subtraction. Then value of the Transaction Parameter becomes 3.

## Action

When a Transaction enters an ASSIGN Block, the value of the Transaction Parameter identified in the A Operand is set according to the B and C operands. A Transaction Parameter is created if necessary.

You may assign, add to, or subtract from the numeric equivalent of the Transaction Parameter's value. If there is no C Operand, Operand B is evaluated and is used as the new value, or its numeric equivalent is used as the increment or decrement. Addition and subtraction are specified by a + or - suffix immediately following the A Operand. If there is no such suffix, Operand B is evaluated and the result is given to the value of the Transaction Parameter.

Optionally, Operand C may be used to determine the number of a function, here called a "Function Modifier". If specified, the function is evaluated, multiplied by the numerical equivalent of the evaluated B Operand, and the result is added, subtracted, or assigned to the value of the Transaction Parameter depending on the optional suffix of the A Operand. Notice that Operand C specifies a Function Entity number or name (do not precede it with an FN or FN\$). If an FN class SNA is used, the GPSS Function is evaluated and the result is used to specify a second GPSS Function which is then evaluated.

## Special Restriction

◆ A must be positive, but may be followed by a + or - suffix.

## Refuse Mode

A Transaction is never refused entry to an ASSIGN Block.

## Related SNA

◆ PParameter or \*Parameter - Parameter value. PParameter returns the value of Parameter *Parameter*. (note: e.g. P1 or \*1 or P\$NAME will yield the value in the Parameter 1 in the first two cases and the Parameter called NAME in the final case.)

## Related Window

- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.

# BUFFER

**A BUFFER Block places the Active Transaction on the Current Events Chain behind its priority peers.**

## BUFFER

### Operands

None.

### Example

## BUFFER

This example creates a Block which gives Transactions other than the Active Transaction a chance to be scheduled.

### Action

When a Transaction enters a BUFFER Block, it is placed on the Current Events Chain behind Transactions of equal priority.

The Transaction scheduler tries to move the Active Transaction as far as it can in the simulation. In effect, the Transaction scheduler removes the Active Transaction from the CEC, calls the routine for the Next Sequential Block (NSB), and unless something extraordinary occurs, replaces the Transaction on the CEC *IN FRONT* of its peers (i.e. same priority) on the CEC. This replacement is modified by PRIORITY and BUFFER Blocks. After a Transaction enters a BUFFER Block, it is replaced *BEHIND* its peers on the CEC. A more detailed discussion of Transaction scheduling is in Chapter 9.

BUFFER Blocks are used to allow newly reactivated Transactions to get ahead of the Active Transaction. It is a common occurrence that the Active Transaction enters a Block which triggers an event which must proceed to completion before the Active Transaction should proceed. It may be necessary to follow such Blocks by BUFFER Blocks in order to allow reactivated Transaction(s) to proceed immediately in the simulation. Alternately, the reactivated Transactions could be given a higher priority.

### Refuse Mode

A Transaction is never refused entry to a BUFFER Block.

### Related Windows

- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# CLOSE

**A CLOSE Block terminates a Data Stream and retrieves its error code.**

**CLOSE A,B,C**

## Operands

**A** - Transaction Parameter. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**B** - Data Stream. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*. The default is 1.

**C** - Alternate Destination. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Example

**CLOSE Error\_Parm,2**

In this example, the CLOSE Block terminates the operation of Data Stream 2 and frees all the resources associated with it. The internal error code of Data Stream 2 is placed in Parameter Error\_Parm of the Active Transaction.

## Action

A CLOSE Block shuts down a Data Stream and retrieves its error code.

If Operand A is used, it is evaluated numerically, truncated, and used as the number of a Parameter of the Active Transaction. If no such Parameter exists, one is created. The error code of the Data Stream is placed in this Parameter.

If Operand B is used, it is evaluated numerically, truncated, and used as the entity number of the Data Stream. The result must be a positive integer. If Operand B is not used, Data Stream number 1 is closed.

If Operand C is used, it is evaluated numerically, truncated, and used as the Block Entity number of an Alternate Destination Block. When the error code of the Data Stream is nonzero, the Active Transaction proceeds to the Alternate Destination Block after it enters the CLOSE Block.

Chapter 4 (4.16) contains a full discussion of Data Streams, including the error code descriptions, under the Section entitled [Data Streams](#).

## Blocking Condition

The simulation is blocked while CLOSE retrieves the error code.

## Special Restrictions

- ◆ A and B, if specified, must be positive.
- ◆ C, if specified, must be a valid Block location in the simulation.

## Refuse Mode

A Transaction is never refused entry to a CLOSE Block.

## Related Blocks

- ◆ OPEN - Create a Data Stream.

- ◆ READ - Retrieve a text line from a Data Stream.
- ◆ WRITE - Send a text line to a Data Stream.
- ◆ SEEK - Set the Current Line Position of a Data Stream.

# COUNT

**A COUNT Block places an entity count into a Parameter of the Active Transaction.**

**COUNT O A,B,C,D,E**

## Operands

**O** - Conditional operator or logical operator. These choices are explained below. Required. The operator must be FNV, FV, I, LS, LR, NI, NU, SE, SF, SNE, SNF, SNV, SV, U, E, G, GE, L, LE, MIN, MAX, or NE.

**A** - Parameter number to receive count. Required. The operand must be *Name, PosInteger, ParenthesizedExpression, SNA, or SNA\*Parameter*.

**B** - Number or name of entity at lower end of range. The entity number of the first entity to be tested. The entity type is implicitly specified by the logical operator or by Operand E. Required. The operand must be *Name, PosInteger, ParenthesizedExpression, SNA, or SNA\*Parameter*.

**C** - Number or name of entity at upper end of range. The entity number of the last entity to be tested. Required. The operand must be *Name, PosInteger, ParenthesizedExpression, SNA, or SNA\*Parameter*.

**D** - Reference value for E Operand. Required only when in Conditional Mode. Optional. The operand must be *Null, Name, Number, String, ParenthesizedExpression, SNA, or SNA\*Parameter*.

**E** - SNA class name. Entity attribute specifier for Conditional Mode tests. Required only for Conditional Mode. The type of SNA implies the entity type. You do not specify the entity number in Operand E. This is done automatically as the entity number range is searched. You may use any entity SNA class. The operand must be *Null or entitySNAclass*.

## Examples

**COUNT SF FullCount,Warehouse1,Warehouse13**

In this example, the number of full Storage Entities whose entity numbers fall between Warehouse1 and Warehouse13 and will be stored in the Transaction Parameter named FullCount. Normally, the Storage Entity Labels should be assigned contiguous integers in a set of EQU Commands.

**COUNT E EmptyCount,FirstQueue,LastQueue,0,Q**

In this example, the COUNT Block operates in Conditional Mode. Operand E specifies SNA class Q, which refers to a Queue Entity. Each Queue Entity with entity number between that of FirstQueue and LastQueue is tested. Any such queue entity whose current content is 0 is counted. EmptyCount is the name of the Parameter of the Active Transaction to receive the count of "empty" Queue Entities in the specified range. Normally, the Queue Entity Labels should be assigned contiguous integers in a set of EQU Commands.

## Action

When the COUNT Block is entered, the entity specified by Operand B is tested. If the entity does not exist and does not require a separate Command for its definition, a new entity is created. Thereafter, each entity in the range indicated by operands B and C is tested. An SNA is built automatically for

each entity. The SNA class used to build the SNA is taken from Operand E or is specified by the logical operator.

A COUNT Block operates in either Logical Mode or in Conditional Mode, depending on whether a logical operator or a conditional operator is used.

When a logical operator is used (defined below), Operands A, B, and C are used. The condition specified by the logical operator is tested for the entities whose numbers fall between B and C. The count of entities in that condition is placed in the Parameter of the entering Transaction whose number or name is given by Operand A. If the Parameter does not exist, it is created. The entity type is implied by the logical operator.

When a conditional operator is used, Operands A, B, C, D, and E are used. Operands A, B, C, are used to specify the target Parameter, and the range of entity numbers, as above. But now the conditional operator specifies the relationship between operands D and E that must hold for the entity to be counted.

In Conditional Mode, the SNA class is combined with the entity specifications in order to build an SNA. The entity type implied by each SNA class is given in Section 3.4. The complete SNA is built from this class and the number of the entity being tested. Each such SNA is evaluated for each entity and compared to the reference value in Operand D. If the condition set up in the conditional operator is met, the entity is counted.

## Logical Operators

Either a conditional operator or a logical operator is required. The logical operator may be FNV, FV, I, LS, LR, NI, NU, SE, SF, SNE, SNF, SNV, SV, or U. When the logical operator is true, the entity being tested is counted. The conditions are defined as follows:

- ◆ FNV - The Facility must be unavailable in order to be counted.
- ◆ FV - The Facility must be available in order to be counted.
- ◆ I - The Facility must be currently interrupted (preempted) in order to be counted.
- ◆ LS - The Logicswitch Entity must be set (in the "on" state) in order to be counted.
- ◆ LR - The Logicswitch Entity must be reset (in the "off" state) in order to be counted.
- ◆ NI - The Facility must NOT be currently interrupted (preempted) in order to be counted.
- ◆ NU - The Facility must not be in use in order to be counted.
- ◆ SE - The Storage must be empty in order to be counted.
- ◆ SF - The Storage must be full in order to be counted.
- ◆ SNE - The Storage must NOT be empty in order to be counted.
- ◆ SNF - The Storage must NOT be full in order to be counted.
- ◆ SNV - The Storage must NOT be available in order to be counted.
- ◆ SV - The Storage must be available in order to be counted.
- ◆ U - The Facility must be in use in order to be counted.

## Conditional Operators

Either a conditional operator or a logical operator is required. The conditional operator may be E, G, GE, L, LE, MAX, MIN, or NE. The conditions are defined as follows:

- ◆ E - The value of the automatic SNA must be equal to the reference value specified by Operand D for the entity to be counted.
- ◆ G - The value of the automatic SNA must be greater than the reference value specified by Operand D for the entity to be counted.

- ◆ GE - The value of the automatic SNA must be greater than or equal to the reference value specified by Operand D for the entity to be counted.
- ◆ L - The value of the automatic SNA must be less than the reference value specified by Operand D for the entity to be counted.
- ◆ LE - The value of the automatic SNA must be less than or equal to the reference value specified by Operand D for the entity to be counted.
- ◆ MAX - The value of the automatic SNA must be equal to the greatest of all such SNAs, for the entity to be counted.
- ◆ MIN - The value of the automatic SNA must be equal to the least of all such SNAs, for the entity to be counted.
- ◆ NE - The value of the automatic SNA must be unequal to the reference value specified by Operand E for the entity to be counted.

## Special Restrictions

- ◆ D and E are required if O is a conditional operator.
- ◆ When evaluated, C must be greater than or equal to B.
- ◆ The number of tested entities must be less than 32768.

## Refuse Mode

A Transaction is never refused entry to a COUNT Block.

## Related Windows

- ◆ Facilities Window - Online view of Facility Entity dynamics.
- ◆ Logicswitches Window - Online view of Logicswitch Entity dynamics.
- ◆ Storages Window - Online view of Storage Entity dynamics.

# DEPART

**A DEPART Block registers statistics which indicate a reduction in the content of a Queue Entity.**

## DEPART A,B

### Operands

**A** - Queue Entity name or number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**B** - Number of units by which to decrease content of the Queue Entity. Default value is 1. Optional. The operand must be *Null*, *Name*, *PosInteger*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

**DEPART WaitingLine**

In this example the content of the Queue Entity named WaitingLine is reduced by one and the associated statistics accumulators are updated.

## Action

When a Transaction enters a DEPART Block, Operand A is evaluated, truncated, and used to find the Queue Entity with that number. The Queue Entity is created if necessary. If a Queue Entity is created, the attempt to decrement the content of the Queue Entity will cause an Error Stop. Manual Simulation can be used to alter the Queue Entity content interactively.

Operand B specifies the value to be used to decrease the content of the Queue Entity. If B was specified, Operand B is evaluated numerically, truncated, and used as the result. If B was not specified, the value of 1 is used.

Finally, the statistics accumulated on behalf of the Queue Entity are updated. If Qtable Entities have been defined for this same Queue Entity, they are also updated.

## Special Restrictions

- ◆ A and B must be positive, if specified.
- ◆ If the content of the Queue Entity is about to become negative, an Error Stop occurs.

## Refuse Mode

A Transaction is never refused entry to a DEPART Block.

## Related SNAs

- ◆ *QEntnum* - Current content of queue entity. The current count value of Queue Entity *Entnum*.
- ◆ *QAEntnum* - Average queue content. The time weighted average count for Queue Entity *Entnum*.
- ◆ *QCEntnum* - Total queue entries. The sum of all queue entry counts for Queue Entity *Entnum*.
- ◆ *QMEntnum* - Maximum queue content. The maximum count (high water mark) of Queue Entity *Entnum*.
- ◆ *QEEntnum* - Average queue residence time. The time weighted average of the count for Queue Entity *Entnum*.
- ◆ *QXEntnum* - Average queue residence time excluding zero entries. The time weighted average of the count for Queue Entity *Entnum* not counting entries with a zero residence time.
- ◆ *QZEntnum* - Queue zero entry count. The number of entries of Queue Entity *Entnum* with a zero residence time.

## Related Window

- ◆ Queues Window - Online view of Queue Entity dynamics.

# DISPLACE

**A DISPLACE Block moves any Transaction.**

# DISPLACE A,B,C,D

## Operands

**A** - Transaction number. Required. The operand must be *Name, PosInteger, ParenthesizedExpression, SNA* or *SNA\*Parameter*.

**B** - Displaced Transaction destination. Block name or number. Required. The operand must be *Name, PosInteger, ParenthesizedExpression, SNA*, or *SNA\*Parameter*.

**C** - Parameter number. Parameter of displaced Transaction to receive residual time if preempted Transaction is removed from FEC. Optional. The operand must be *Null, Name, PosInteger, ParenthesizedExpression, SNA*, or *SNA\*Parameter*.

**D** - Alternate destination for the Active Transaction. Block name or number. Optional. The operand must be *Null, Name, PosInteger, ParenthesizedExpression, SNA*, or *SNA\*Parameter*.

## Example

**DISPLACE X\$Culprit,Compensate,Residual,NotCaught**

In this example, the DISPLACE Block moves the Transaction whose number is kept in the Savevalue named Culprit. If the target Transaction exists, it is sent to the Block labeled Compensate as its Next Sequential Block. If it was residing on the Future Events Chain, any time left until its reentry into the simulation is calculated and stored in the Parameter named Residual of the displaced Transaction. If the Transaction was on the FEC and no such Parameter exists, it is created. If the target Transaction does not exist, the Active Transaction moves to the alternate destination Block labeled NotCaught, after entering the DISPLACE Block.

## Action

A DISPLACE Block moves any Transaction in the simulation to any Block. The displaced Transaction is removed from Transaction chains, as discussed below, and is scheduled to enter the destination Block.

When a Transaction enters a DISPLACE Block, Operand A is evaluated numerically, truncated, and used to find the Transaction to be displaced. If that Transaction does not exist, processing ceases, and the Active Transaction is sent to the Alternate Destination specified by Operand D, if any.

If the Transaction exists, it is given the new Block destination specified by Operand B.

If the displaced Transaction is on the FEC, it is removed from it and the residual time duration is calculated as the time which the Transaction is scheduled to come off the FEC minus the current time. If the C Operand is used, the residual time is saved in a Transaction Parameter. If no such Parameter exists and Operand C is used, a new Parameter is created.

When a Transaction is displaced, it is given a new Block destination and is dequeued from:

- ◆ FEC
- ◆ PENDING (INTERRUPT-MODE PREEMPT) CHAINS
- ◆ DELAY (MAJOR PRIORITY) CHAINS
- ◆ USER CHAINS
- ◆ RETRY CHAINS

and not dequeued from:

- ◆ CEC
- ◆ INTERRUPT (PREEMPTED) CHAINS
- ◆ GROUP CHAINS

When a Transaction is displaced, preemptions at Facilities are not cleared.

## Refuse Mode

A Transaction is never refused entry to a DISPLACE Block.

## Related SNA

- ◆ XN1 - Transaction number of the Active Transaction.

## Related Windows

- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

## Related Library Procedures

- ◆ QueryXNExist - determine the existence of a Transaction.
- ◆ QueryXNParameter - retrieve the value of a Transaction Parameter.
- ◆ QueryXNAssemblySet - retrieve the Assembly Set of a Transaction.
- ◆ QueryXNPriority - retrieve the priority of a Transaction.
- ◆ QueryXNM1 - retrieve the Mark Time of a Transaction.

# ENTER

**When a Transaction attempts to enter an ENTER Block, it either takes or waits for a specified number of storage units.**

## ENTER A,B

### Operands

**A** - Storage Entity name or number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**B** - Number of units by which to decrease the available storage capacity. Default value is 1. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

#### ENTER Toolkit,2

In this example the Active Transaction demands 2 storage units from the storage units available at the Storage Entity named Toolkit. If there are not enough storage units remaining in the Storage Entity, the Transaction comes to rest on the Delay Chain of the Storage Entity.

### Action

When a Transaction enters an ENTER Block, Operand A is evaluated and used to find the Storage Entity with that number. If the Storage Entity does not exist, an Error Stop occurs. Storage entities must be defined by a STORAGE Command.

If the Storage Entity exists, Operand B is used to determine the storage demand. If B was specified, Operand B is evaluated, truncated, and used as the result. If B was not specified, the value of 1 is used.

If the Storage Entity is in the available state, and if there are enough storage units to satisfy the demand, the Transaction is allowed to enter the ENTER Block and the demand is granted by reducing the current storage units by the amount of the demand. Otherwise, the Transaction comes to rest on the Delay Chain of the Storage Entity in priority order.

When storage units are freed by a LEAVE Block, a "first-fit-with-skip" discipline is used to choose the next Transaction(s) to be allowed entry into the ENTER Block. Any such entry is permitted before the Active Transaction leaves the LEAVE Block. This prevents "line-bucking".

When a Transaction enters an ENTER or LEAVE Block, the statistics accumulated on behalf of the Storage Entity are updated.

## Special Restrictions

- ◆ A must be positive.
- ◆ B must be nonnegative.
- ◆ A request for more storage than exists will result in an Error Stop. (e.g. a B Operand that is larger than the defined size of the STORAGE named in Operand A.)

## Refuse Mode

- ◆ The Active Transaction is refused entry to the ENTER Block if its storage demand cannot be met.
- ◆ The Active Transaction is refused entry to the ENTER Block if the Storage Entity is in the unavailable state.

When a Transaction is refused entry, its Delay Indicator is set and remains so until the Transaction enters a "Simultaneous" Mode TRANSFER Block. Simultaneous Mode TRANSFER Blocks are rarely used. A BOOLEAN VARIABLE can more efficiently control the coordination of the state of a number of resources when used in a TEST Block.

## Related Command

A Storage Entity must be defined in a STORAGE Command before it can be updated by an ENTER Block. The STORAGE Command must exist in the model, or must be sent to the Simulation Object interactively, before a Transaction can enter the ENTER Block. Any attempt to do so before the Storage Entity is defined, cases an Error Stop.

A Storage Entity can be redefined by an interactive STORAGE Command.

## Related SNAs

- ◆ *REntrnum* - Unused storage capacity. The storage content (or "token" spaces) available for use by entering Transactions at Storage Entity *Entrnum*.
- ◆ *SEntrnum* - Storage in use. *SEntrnum* returns the amount of storage content (or "token" spaces) currently in use by entering Transactions at Storage Entity *Entrnum*.
- ◆ *SAEntrnum* - Average storage in use. *SAEntrnum* returns the time weighted average of storage capacity (or "token" spaces) in use at Storage Entity *Entrnum*.
- ◆ *SCEntrnum* - Storage use count. Total number of storage units that have been entered in (or "token" spaces that have been used at) Storage Entity *Entrnum*.
- ◆ *SEEntrnum* - Storage empty. *SEEntrnum* returns 1 if Storage Entity *Entrnum* is completely unused, 0 otherwise.

- ◆ *SFEnnum* - Storage full. *SFentnum* returns 1 if Storage Entity *Entnum* is completely used, 0 otherwise.
- ◆ *SREnnum* - Storage utilization. The fraction of total usage represented by the average storage in use at Storage Entity *Entnum*. *SREnnum* is expressed in parts-per-thousand and therefore returns an real value 0-1000, inclusively.
- ◆ *SMEntnum* - Maximum storage in use at Storage Entity *Entnum*. The "high water mark".
- ◆ *STEnnum* - Average holding time per unit at Storage Entity *Entnum*.
- ◆ *SVEnnum* - Storage in available state. *SVEnnum* returns 1 if Storage Entity *Entnum* is in the available state, 0 otherwise.

## Related Window

- ◆ *Storages Window* - Online view of Storage Entity dynamics.

# EXAMINE

**An EXAMINE Block may be used to test for membership in a Numeric Group or a Transaction Group.**

## EXAMINE A,B,C

### Operands

**A** - Group number. Group whose members will be examined. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Numeric Mode only. The value to be tested for membership in the Numeric Group. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**C** - Block number. Alternate destination for Active Transaction if no membership is found. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Examples

**EXAMINE ValidColors,P\$Color,NotCorrectColor**

In this simple example, if the Numeric Group named ValidColors does not include the value contained in the Transaction Parameter named Color, the Active Transaction proceeds to the Block location NotCorrectColor. If the value is a Numeric Group member, the Active Transaction proceeds to the Next Sequential Block (NSB).

**EXAMINE ValidXNs,,NotValidXN**

In this example, if the Active Transaction is not a member of the Transaction Group named ValidXNs, the Active Transaction proceeds to the Block location NotValidXN. If the Transaction is a Transaction Group member, it proceeds to the Next Sequential Block (NSB).

### Action

An EXAMINE Block operates in either "Numeric Mode" or "Transaction Mode". If the B Operand is used, the EXAMINE Block operates in Numeric Mode. In this case the value of Operand B is tested

for membership in the Numeric Group. If the B Operand is not used, the EXAMINE operates in Transaction Mode. Then, It is the Active Transaction which is tested for membership in a Transaction Group.

When a Transaction enters an EXAMINE Block, Operand A is evaluated and the appropriate Group entity is found. If there is no such Group, one is created.

If the B Operand is used, it is evaluated numerically and the result is tested for membership in the Numeric Group. If Operand B is not used, the Active Transaction is tested for membership in the Transaction Group specified by the A Operand.

There is some loss of efficiency when non-integers are used in Numeric Groups.

If the membership test fails, Operand C is evaluated, truncated, and used as the destination Block number for the Active Transaction. If the membership test is successful, the Active Transaction proceeds to the Next Sequential Block (NSB).

## Special Restrictions

- ◆ A and C must be positive.
- ◆ C must be a Block location in the simulation.

## Refuse Mode

A Transaction is never refused entry to an EXAMINE Block.

## Related Blocks

Transactions and numbers are added to Groups by JOIN Blocks. Transactions in Transaction Groups can be referenced by ALTER, EXAMINE, REMOVE, and SCAN Blocks. Numbers in Numeric Groups can be referenced by EXAMINE and REMOVE Blocks.

## Related SNAs

- ◆ *GNEntnum* - Numeric Group Count. *GNEntnum* returns the membership count of Numeric Group *Entnum*.
- ◆ *GTEntnum* - Transaction Group Count. *GTEntnum* returns the membership count of Transaction Group *Entnum*.

## Related Windows

- ◆ Numeric Groups Snapshot - Picture of the state of the Numeric Groups in the simulation.
- ◆ Transaction Groups Snapshot - Picture of the state of the Transaction Groups in the simulation.

# EXECUTE

**The EXECUTE Block can cause any Block routine in the simulation to be done on behalf of the Active Transaction.**

## EXECUTE A

### Operand

**A** - Block number. The name or number of the Block to be "performed". required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Example

### EXECUTE P\$ActiveBlock

This EXECUTE Block will cause the Block, whose number is in the Parameter named ActiveBlock, to be executed with respect to the Active Transaction.

## Action

When a Transaction enters an EXECUTE Block, the A Operand is evaluated and is used to find the Block with that number or name. Then the Block routine associated with the target Block is done on behalf of the Active Transaction.

If the Active Transaction is refused entry to the target Block, it remains in the EXECUTE Block.

After the target Block routine is performed, the Active Transaction is normally scheduled for the Block following the EXECUTE Block. However, if the target Block schedules an alternate destination for the Active Transaction, that Block, and not the EXECUTE Block's next sequential Block, is scheduled.

## Special Restrictions

- ◆ A must be a Block location in the simulation.
- ◆ An EXECUTE Block cannot act on another EXECUTE Block.

## Refuse Mode

An EXECUTE Block never refuses entry to a Transaction.

## Related Window

- ◆ Blocks Window - Online view of Block dynamics.

# FAVAIL

**A FAVAIL Block ensures that a Facility Entity is in the available state.**

### FAVAIL A

## Operand

A - Facility number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Example

### FAVAIL Teller1

This Block, when entered by an Active Transaction, will ensure that the Facility Entity named Teller1 is in the available state.

## Action

A FAVAIL Block ensures that a Facility Entity is in the available state. If the Facility Entity was previously idle, the FAVAIL Block tries to give ownership to a waiting Transaction. This is discussed in more detail in Chapter 9.

If the Facility Entity was previously available, FAVAIL has no effect.

FAVAIL cancels the affects of FUNAVAIL on the Facility Entity, but does not affect displaced Transactions.

## Refuse Mode

A Transaction is never refused entry to a FAVAIL Block.

## Related Block

- ◆ FUNAVAIL - Place Facility Entity in the unavailable state.

## Related SNAs

The SNAs associated with Facilities are:

- ◆  $FEnignum$  - Facility busy. If Facility Entity  $Entnum$  is currently busy,  $FEnignum$  returns 1. Otherwise  $FEnignum$  returns 0.
- ◆  $FCEnignum$  - Facility capture count. The number of times Facility Entity  $Entnum$  has become owned by a Transaction.
- ◆  $FIEnignum$  - Facility  $Entnum$  interrupted. If Facility Entity  $Entnum$  is currently preempted,  $FIEnignum$  returns 1. Otherwise  $FIEnignum$  returns 0.
- ◆  $FREnignum$  - Facility utilization. The fraction of time Facility Entity  $Entnum$  has been busy.  $FREnignum$  is expressed in parts-per-thousand and therefore returns an real value 0-1000, inclusively.
- ◆  $FTEnignum$  - Average Facility holding time. The average time Facility Entity  $Entnum$  is owned by a capturing Transaction.
- ◆  $FVEnignum$  - Facility in available state.  $FVEnignum$  returns 1 if Facility Entity  $Entnum$  is in the available state, 0 otherwise.

## Related Window

- ◆ Facilities Window - Online view of Facility Entity dynamics.

# FUNAVAIL

**FUNAVAIL Blocks are used to make a Facility Entity unavailable for ownership by Transactions.**

## FUNAVAIL A,B,C,D,E,F,G,H

### Operands

**A** - Facility name or number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**B** - REmove or COntinue Mode for owning Transaction. Optional. The operand must be *RE*, *CO*, or *Null*.

**C** - Block number. New Block for Transaction owning the Facility Entity. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**D** - Parameter number. Parameter to receive residual time if owning Transaction is removed from FEC. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**E** - REmove or COninue Mode for preempted Transactions. Optional. The operand must be RE, CO or Null.

**F** - Block number. New Block for Transactions preempted at the Facility Entity. Optional. The operand must be Null, Name, PosInteger, ParenthesizedExpression, SNA or SNA\*Parameter.

**G** - REmove or COninue Mode for Transactions pending or delayed at the Facility Entity. Optional. The operand must be RE, CO or Null.

**H** - Block number. New Block for Transactions pending or delayed at the Facility Entity. Optional. The operand must be Null, Name, PosInteger, ParenthesizedExpression, SNA or SNA\*Parameter.

## Examples

### **FUNAVAIL Teller1**

This is the simplest way to use the FUNAVAIL Block. The Facility Entity named Teller1 is "frozen" during the unavailable period. Several types of Transactions other than the Active Transaction are affected. If an owning Transaction is in an ADVANCE Block, it will be rescheduled on the FEC automatically when the Facility again becomes available. The time remaining in the ADVANCE Block (i.e. the "residual time") is calculated and saved automatically when the Transaction is removed from the FEC. The residual time becomes the time increment when the Transaction is rescheduled on the FEC. Other Transactions delayed or pending at TELLER1 will not be able to move during the unavailable period.

### **FUNAVAIL TELLER,RE,TLR2,300,RE,MGR,CO**

In this example, the Facility Entity TELLER is made unavailable. This means that newly arriving Transactions will be delayed. The Transactions at or preempted at the Facility TELLER are dealt with according to operands B-H. If the Transaction which owns Facility TELLER is on the FEC, it is dequeued and the residual time is saved in its Parameter number 300. The owning Transaction is displaced and sent to the Block named TLR2. Transactions currently preempted at Facility TELLER are removed from contention and sent to Block named MGR. Since Operand G is CO, Transactions currently delayed at Facility TELLER are allowed to own the Facility, even though it is in the unavailable state.

## Action

The complexity of the FUNAVAIL Block is due to the three classes of Transactions which must be dealt with:

1. the owning Transaction (operands B-D),
2. preempted Transactions on the Interrupt Chain (operands E-F), and
3. delayed Transactions on the Delay Chain or Pending Chain (operands G-H).

A FUNAVAIL Block allows you to put a Facility "out of action" and to control the fate of Transactions waiting for, using, or preempted at the Facility Entity. Transactions arriving during the unavailable period will be delayed and not be granted ownership. A FUNAVAIL Block has no effect if the Facility is already unavailable.

When the REmove option is used, the Transactions are removed from contention for the Facility. If the REmove option is used for pending and delayed Transactions, i. e. if G is RE, then H must be used to redirect the Transactions.

When the COninue option is used, the Transactions on the specific Facility chain may continue to own the Facility, even during the unavailable period. In this case, Facility utilization statistics are adjusted to include this time.

When an alternate destination Block is used, the Transactions are displaced from their current context and redirected to the new Block. Delayed and pending Transactions, which are controlled by operands G and H, cannot be redirected without also using the REmove option. The owning Transaction, controlled by operands B through D, and preempted Transactions, controlled by operands E and F, can remain in contention for the Facility and yet be displaced to a new destination. This is done by specifying an alternate destination without using the corresponding RE option.

When RE is not used in Operand B, any owning Transaction becomes preempted at this Facility. Such Transactions cannot leave ASSEMBLE, GATHER, or MATCH Blocks or enter (nonzero) ADVANCE Blocks until all preemptions are cleared.

When a Transaction is displaced by using the C, F, or H Operand, it is given a new Block destination and is dequeued from:

- ◆ FEC
- ◆ PENDING (waiting to preempt) CHAINS
- ◆ DELAY (waiting to seize) CHAINS
- ◆ USER CHAINS
- ◆ RETRY CHAINS

and not dequeued from:

- ◆ CEC
- ◆ INTERRUPT (preempted) CHAINS of other Facilities
- ◆ GROUP CHAINS

When a Transaction is displaced from its present context anywhere in the model, by using an alternate destination, it is removed from any Blocking conditions but preemptions are not cleared. Such a displaced Transaction is scheduled to enter the Block specified in Operand C of the FUNAVAIL Block. When an owning Transaction RELEASEs or RETURNS a Facility, the next owner is chosen from the head of the Pending Chain, the Interrupt Chain, or the Delay Chain, in that order.

## TheOwning Transaction

Operands B-D are used to control the owning Transaction.

If B is CO, the owning Transaction is not removed from ownership. It may, however, be given a new destination with the C Operand.

If B is RE, the owning Transaction is removed from contention for the Facility. This means that the Transaction may continue to circulate in the simulation without restrictions due to a preemption at this Facility (there may be outstanding preemptions at other Facilities, however). It also means that the owning Transaction must not attempt to RETURN or RELEASE the Facility. The C Operand must be used to redirect the course of such a Transaction.

If B is *Null*, the owning Transaction is preempted and placed on the Interrupt Chain of the Facility. If it was taken from the FEC and the C Operand is not used, it will be automatically restored to the FEC using the automatically saved residual time when the Facility again becomes available.

The C Operand may be used regardless of Operand B. It causes the owning Transaction to be displaced, and gives it a new destination Block. If you choose to return the Transaction to the FEC, having used the C Operand, you must explicitly route the Transaction to an ADVANCE Block. The D Operand causes the residual time to be saved in a Parameter of the owning Transaction. The residual time value is then available for explicit rescheduling when you use the Parameter value as Operand A of an ADVANCE Block.

## Preempted Transactions

Operands E and F are provided to control the fates of Transactions currently preempted at this Facility.

If E is CO, preempted Transactions are not removed from contention for the Facility, and may own the Facility during any unavailable period. Preempted Transactions may be given a new destination with the F Operand.

If E is RE, preempted Transactions are removed from contention for the Facility. This means that the Transaction may continue to circulate in the simulation without restrictions due to a preemption at this Facility (there may be outstanding preemptions at other Facilities, however). It also means that preempted Transactions must not attempt to RETURN or RELEASE the Facility. Optionally, the F Operand is available to redirect the course of such a Transaction.

If E is *Null*, preempted Transactions are left on the Interrupt Chain of the Facility, and cannot be granted ownership of the Facility during the unavailable period.

The F Operand may be used regardless of Operand E. It causes preempted Transactions to be displaced, and gives them a new destination Block. Preempted Transactions may not exist on the FEC, so no residual time options are relevant. If E is *Null*, the preemption is not cleared for displaced Transactions.

## Pending Interrupt Mode Transactions

Operands G and H are provided to control the fates of Transactions on the Pending Chain (i.e. pending Interrupt Mode PREEMPTs) or the Delay Chain.

If G is CO, delayed Transactions are not removed from contention for the Facility, and may own the Facility during any unavailable period.

If G is RE, delayed Transactions are removed from contention for the Facility and allowed to circulate in the simulation. These Transactions must not attempt to RETURN or RELEASE the Facility. The H Operand must be used to redirect the course of such Transactions.

If G is *Null*, delayed Transactions are left on the Delay Chain or the Pending Chain of the Facility, and cannot be granted ownership of the Facility during the unavailable period. The use of Operand H is invalid when G is *Null*.

## Special Restrictions

- ◆ If B is RE, C must be used.
- ◆ If H is used, G must be RE.
- ◆ If G is RE, H must be used.

## Refuse Mode

A Transaction is never refused entry to a FUNAVAIL Block.

## Related Block

- ◆ FAVAIL - Place Facility in the available state.

## Related SNAs

- ◆ *FEnignum* - Facility busy. If Facility *Entnum* is currently busy, *FEnignum* returns 1. Otherwise *FEnignum* returns 0.
- ◆ *FCEnignum* - Facility capture count. The number of times Facility *Entnum* has become owned by a Transaction.
- ◆ *FIEnignum* - Facility *Entnum* interrupted. If Facility *Entnum* is currently preempted, *FIEnignum* returns 1. Otherwise *FIEnignum* returns 0.
- ◆ *FREnignum* - Facility utilization. The fraction of time Facility *Entnum* has been busy. *FREnignum* is expressed in parts-per-thousand and therefore returns an real value 0-1000, inclusively.
- ◆ *FTEnignum* - Average Facility holding time. The average time Facility *Entnum* is owned by a capturing Transaction.
- ◆ *FVEnignum* - Facility in available state. *FV Enignum* returns 1 if Facility *Entnum* is in the available state, 0 otherwise.

## Related Windows

- ◆ Facilities Window - Online view of Facility Entity dynamics.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.

# GATE

**A GATE Block alters Transaction flow based on the state of an entity.**

## **GATE O A,B**

### **Operands**

**O** - Conditional operator. Condition required of entity to be tested for successful test. Required. The operator must be FNV, FV, I, LS, LR, M, NI, NM, NU, SE, SF, SNE, SNF, SNV, SV, or U.

**A** - Entity name or number to be tested. The entity type is implied by the conditional operator. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Destination Block number when test is unsuccessful. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, *SNA\*Parameter*.

### **Examples**

#### **GATE SNF MotorPool**

In this example of a "Refuse Mode" GATE Block, the Active Transaction enters the GATE Block if the Storage Entity named MotorPool is not full (i. e. if at least 1 unit of storage is available). If the Storage is full, the Active Transaction is blocked until 1 or more storage units become available.

#### **GATE SNE MotorPool,CupboardIsBare**

In this example of an "Alternate Exit Mode" GATE Block, the Active Transaction always enters the GATE Block. If the Storage Entity named MotorPool is not empty (i. e. if at least 1 unit of storage is in use) the Transaction proceeds to the NSB. If the Storage is empty (unsuccessful test), the Active Transaction is scheduled to enter the Block at the location named CupboardIsBare.

### **Action**

A GATE Block operates in either "Refuse Mode" or "Alternate Exit Mode".

If Operand B is not used, the GATE Block operates in Refuse Mode. When a Transaction attempts to enter a Refuse Mode GATE Block, and the test is unsuccessful, the Transaction is blocked until the test is repeated and is successful. If the test is successful, the Active Transaction enters the GATE Block and then proceeds to the Next Sequential Block.

Blocked Transactions are placed on the Retry Chain of the tested entity. When the state of any of the entity changes, the blocked Transaction is reactivated, the test specified by the GATE block is retried, and if successful, the Transaction is permitted to enter the GATE Block. However, the integration of User Variables does not cause blocked Transactions to be reactivated. You should use the thresholds in an INTEGRATE Command if you need to be notified about the level of one or more continuous variables. This is discussed further in Chapter 1, in the Section entitled Continuous Variables.

If Operand B is used, the GATE Block operates in Alternate Exit Mode. When a Transaction attempts to enter such a GATE Block, and the test is unsuccessful, the Transaction enters the GATE Block, is scheduled for the alternate destination Block specified by the B Operand, and is placed on the Current Events Chain in front of its priority peers. If the test is successful, the Active Transaction enters the GATE Block and then proceeds to the Next Sequential Block.

### **Conditional Operators**

The conditional operator is required. It may be FNV, FV, I, LS, LR, M, NI, NM, NU, SE , SF, SNE, SNF, SNV, SV, or U. When the condition is true, the Transaction enters the GATE Block and proceeds to the Next Sequential Block. The conditions are defined as follows:

- ◆ FNV - The Facility specified implicitly by Operand A must be unavailable for a successful test.
- ◆ FV - The Facility specified implicitly by Operand A must be available for a successful test.
- ◆ I - The Facility specified implicitly by Operand A must be currently interrupted for a successful test.
- ◆ LS - The Logicswitch entity specified implicitly by Operand A must be in the "set" state for a successful test.
- ◆ LR - The Logicswitch entity specified implicitly by Operand A must be in the "reset" state for a successful test.
- ◆ M - The Match Block specified implicitly by Operand A must have a related (to the Active Transaction) Transaction waiting for a Match Condition.
- ◆ NI - The Facility specified implicitly by Operand A must be not interrupted for a successful test.
- ◆ NM - The Match Block specified implicitly by Operand A must not have a related (to the Active Transaction) Transaction waiting for a Match Condition.
- ◆ NU - The Facility specified implicitly by Operand A must not be in use for a successful test.
- ◆ SE - The Storage Entity specified implicitly by Operand A must be empty for a successful test. All storage units must not be in use.
- ◆ SF - The Storage Entity specified implicitly by Operand A must be full for a successful test. All storage units must be being used.
- ◆ SNE - The Storage Entity specified implicitly by Operand A must be not empty for a successful test. At least one storage unit must be in use.
- ◆ SNF - The Storage Entity specified implicitly by Operand A must be not full for a successful test. There must be at least one storage unit that can be used.
- ◆ SNV - The Storage Entity specified implicitly by Operand A must be in the "unavailable" state for a successful test.
- ◆ SV - The Storage Entity specified implicitly by Operand A must be in the "available" state for a successful test.
- ◆ U - The Facility specified implicitly by Operand A must be in use for a successful test.

## Special Restrictions

- ◆ B, if specified, must be the location of a Block in the simulation.
- ◆ GATE Blocks are extremely powerful, but they can cause a lot of computer time to be used in unsuccessful tests. You may need to arrange your simulation to reduce the frequency of unsuccessful tests. This can be done by placing Transactions with no chance of a successful test on a User Chain using LINK and UNLINK Blocks.
- ◆ The MB class of SNA should not be used alone to specify a blocking condition in a GATE Block. You should use MATCH Blocks instead.

## Refuse Mode

A GATE Block operating in Refuse Mode will refuse entry to a Transaction when the test fails. The refused Transaction will be blocked until the test is successful.

When a Transaction is refused entry, its delay indicator is set and remains so until the Transaction enters a "Simultaneous" Mode TRANSFER Block. However, since the advent of the Boolean Variable in the GPSS language, it is more efficient to use a TEST Block and a Boolean Variable when it is necessary to coordinate the state of multiple entities.

## Related Windows

- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Facilities Window - Online view of Facility Entity dynamics.
- ◆ Logicswitches Window - Online view of Logicswitch Entity dynamics.
- ◆ Storages Window - Online view of Storage Entity dynamics.

# GATHER

Wait for related Transactions.

## GATHER A

### Operand

A - Transaction count. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

#### GATHER 2

This is the simplest way to use the GATHER Block. The first Transaction of an Assembly Set (see Section 9.3) is caused to wait when it enters an GATHER Block. When another Transaction in the same Assembly Set enters the Block, both related Transactions are released and put on the Current Events Chain.

### Action

When a Transaction enters a GATHER Block, the Match Chain of the Block is searched for a waiting Transaction of the same Assembly Set. If there are no other members of the same Assembly Set present, the A Operand is evaluated, truncated, decremented by one, and saved in a storage location in the Active Transaction. If this number is less than or equal to zero, the Transaction immediately attempts to enter the Next Sequential Block. Otherwise, the Transaction is placed on a special chain in the ASSEMBLY Block, called the Match Chain, to await the arrival of other members of its Assembly Set.

If the Active Transaction arrives to find other members of its Assembly Set already on the Match Chain, the Active Transaction is also placed on the chain and the Transaction count saved in the first chained Transaction is reduced by one. When this count becomes 0, all related Transactions are removed from the Match Chain. All Transactions which have not been preempted at any Facility are then placed on the CEC behind their priority peers.

Preempted Transactions which have completed an assembly at a GATHER Block are not permitted to leave the Block until all preemptions have been cleared. More discussion of the preemption mechanism can be found in Section 9.4. Preempted Transactions which have been removed from the Match Chain do not participate in later gatherings even though they remain in the GATHER Block.

GATHER Blocks differ from ASSEMBLE Blocks in that Transactions after the first are destroyed at an ASSEMBLE Block.

## Special Restrictions

- ◆ A must be positive.
- ◆ Transactions which are currently preempted are not permitted to leave GATHER Blocks.

## Refuse Mode

A Transaction is never refused entry to a GATHER Block.

## Related Blocks

- ◆ ADOPT - Set the Assembly Set of the Active Transaction.
- ◆ ASSEMBLE - Wait for and destroy Assembly Set members.
- ◆ MATCH - Wait for Assembly Set member.
- ◆ SPLIT - Create Transactions in the same Assembly Set.

## Related SNAs

- ◆ A1 - Assembly Set. Return the Assembly Set of the Active Transaction.
- ◆ MBE<sub>n</sub> - Match at Block. MBE<sub>n</sub> returns a 1 if there is a Transaction at Block <sub>n</sub> which is in the same Assembly Set as the Active Transaction. MBE<sub>n</sub> returns a 0, otherwise.

## Related Windows

- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# GENERATE

**A GENERATE Block creates Transactions for future entry into the simulation.**

**GENERATE A,B,C,D,E**

## Operands

**A** - Mean inter generation time. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, or *DirectSNA*. You may not use Transaction Parameters.

**B** - Inter generation time half-range or Function Modifier. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, or *DirectSNA*. You may not use Transaction Parameters.

**C** - Start delay time. Time increment for the first Transaction. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, or *DirectSNA*. You may not use Transaction Parameters.

**D** - Creation limit. The default is no limit. Optional. The operand must be *Null*, *Name*, *PosInteger*, *String*, *ParenthesizedExpression*, or *DirectSNA*. You may not use Transaction Parameters.

**E** - Priority level. Optional. Zero is the default. The operand must be *Null*, *Name*, *integer*, *String*, *ParenthesizedExpression*, or *DirectSNA*. You may not use Transaction Parameters.

## Example

### GENERATE 0.1

This is the simplest way to use the GENERATE Block. This Block causes a priority zero Transaction to enter the simulation every tenth of a time unit.

## Action

When a simulation is begun, or an interactive Command is performed, any GENERATE Block which has not been "primed" is called upon to schedule its first Transaction. Such Transactions are scheduled to enter the GENERATE Block and placed on the Future Events Chain if they have a positive time increment. Operand C can be used to specify a positive time increment for the first Transaction. Otherwise, the first time increment is calculated from operands A and B.

You cannot use Parameters in GENERATE Block operands. Newly GENERATEd Transactions do not have Parameters, and their entry into such a GENERATE Block would cause an Error Stop.

Before the new Transaction is created, Operand D is evaluated numerically to see if all the Transactions desired have been created. If the creation limit has not been exceeded, processing continues. The GENERATE Block then creates the new Transaction assigning it the next Transaction number, the priority from the E Operand, and the Transaction Mark Time is assigned the value in the absolute system clock. The new Transaction represents a new Assembly Set with one member.

The inter arrival time for the new Transaction is calculated from the A, B, and C operands. If only the A Operand is specified, it is evaluated numerically and used as the time increment. If the A and B operands are present, and B does not specify a function, both A and B are evaluated numerically and a random number between A-B and A+B, inclusively, is used as the time increment. You can select which random number generator number is to be used as the source of the random number. This is set in the "Random" page of the Model Settings Notebook.

### CHOOSE **Edit / Settings**

then select the **Random** page. Then fill in the desired random number stream entity number in the entry box marked "GENERATE". The installation default is to use random number stream number 1.

When Operand B is an FN class SNA, it is a special case called a "function modifier". In this case, the time increment is calculated by multiplying the result of the function by the evaluated A Operand.

If the C Operand is specified, it is evaluated numerically and used as the time increment for the first Transaction. If you wish to cause the first transaction(s) to arrive at time 0, you must use a separate GENERATE block with a creation limit in Operand D and null values for Operands A and B. If Operand A and / or B is used and C=0, C will be interpreted as null. The following example will create three transactions at time 0.

### GENERATE „0,3

If the time increment is strictly positive (nonzero), the Transaction is placed on the FEC, if it is zero the Transaction goes to the CEC behind its priority peers, if it is negative an Error Stop occurs.

## Special Restrictions

- ◆ Time values must not be negative.
- ◆ Either Operand A or Operand D must be used.
- ◆ GENERATE is the only Block which cannot be used in Manual Simulation Mode.  
Use a SPLIT Block to create Transactions interactively.

## Refuse Mode

A Transaction is never refused entry to a GENERATE Block.

## Related Windows

- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# INDEX

**An INDEX Block updates a Parameter of the Active Transaction.**

## INDEX A,B

### Operands

**A** - Parameter number. Parameter with source value. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Numeric value. Number to be added to contents of Parameter. The result goes into Parameter 1. Required. The operand must be *Name*, *Number*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

#### INDEX 2,11.7

In this example, when a Transaction enters the INDEX Block its Parameter number 1 is given the sum of 11.7 and the value of Parameter 2 of the Active Transaction.

### Action

The INDEX Block adds the numeric equivalent of Operand B to the numeric equivalent of the value of any Transaction Parameter and puts the result into Parameter 1.

If the source Parameter does not exist, an Error Stop occurs. Otherwise, if Parameter number 1 does not exist for the Active Transaction, it is created.

### Special Restriction

- ◆ A must be positive.

### Refuse Mode

A Transaction is never refused entry to an INDEX Block.

### Related SNA

- ◆ *PParameter* or *\*Parameter* - Parameter value. Returns the value of Parameter *Parameter*.

### Related Window

- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.

# INTEGRATION

An INTEGRATION Block disables or enables the integration of a User Variable.

## INTEGRATION A,B

### Operands

A - User Variable. Required. The operand must be *Name*.

B - Integration state. Must be *Null*, ON, or OFF.

### Examples

#### INTEGRATION Population

This is the simplest way to use the INTEGRATION Block. When a Transaction enters this INTEGRATION Block, the integration of the User Variable, Population, is set to the enabled state. This causes the User Variable to be integrated automatically with respect to simulated time. The default of Operand B is ON.

#### INTEGRATION Population,OFF

In this example, the entry of the Active Transaction into the INTEGRATION Block assures that the integration of the User Variable Population is disabled. The automatic update of Population will cease until the integration is again enabled.

### Action

A INTEGRATION Block sets the state of an integration to either ON, or OFF, that is, enabled or disabled.

The default of Operand B is ON, and need not be specified when the integration is to be activated.

Integrations are defined by INTEGRATE Commands, and are automatically begun in the active state. For integrations that are never interrupted, there is no need for INTEGRATION Blocks in the simulation.

Each INTEGRATE Command can also define one or two threshold expressions. When the value of an integrated variable crosses from one side of a threshold to the other, a new Transaction is created and sent to a destination Block specified in the INTEGRATE Command. In this manner, Continuously integrated variables can be closely incorporated into the discrete side of your simulation. You can use them to perform important duties related to the state of the integrated variable, or simply to move the thresholds.

User Variables can be assigned new values discretely, as well as through integration. You can do so using an EQU Command, or a PLUS Assignment Statement. If you want such assignments to occur within the running of the simulation, you must define a PLUS Procedure that makes the assignment. For example, if you defined a PLUS Procedure as follows:

```
PROCEDURE Setpop(Pop_Level) BEGIN  
    Population = Pop_Level ;  
END ;
```

you could reinitialize the Population User Variable by entering a PLUS Block, such as

**PLUS (Setpop(200))**

or by using a parenthesized expression that invokes setpop( ) in some other Block.

See Chapter 4 for a detailed description of Continuous Simulation.

## Refuse Mode

A Transaction is never refused entry to an INTEGRATION Block.

## Related Command

- ◆ INTEGRATE - Define the derivative of a user variable for integration, and activate the integration.

## Related Windows

- ◆ Expressions Window - Online view of values of expressions.
- ◆ Plot Window - Online view of a plot of up to 8 expressions.

# JOIN

**A JOIN Block adds the Active Transaction to a Transaction Group, or adds a number to a Numeric Group.**

**JOIN A,B**

## Operands

**A** - Group entity number. Group to which a member will be added. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Numeric value. Number to be added to numeric Group. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Example

**JOIN SoldItems**

This is the simplest way to use the JOIN Block. The Transaction entering the JOIN Block becomes a member of the Transaction Group SoldItems.

## Action

JOIN Blocks operate in "Numeric Mode" when a number is specified by the B Operand, and otherwise in "Transaction Mode".

In Numeric Mode, operands A and B are evaluated numerically, and the number specified by B is included in the Numeric Group specified by A. If this number is already a member of the Numeric Group, no operation is performed. There is some loss of efficiency when non-integers are used in Numeric Groups.

In Transaction Mode, the entering Transaction is included in the Transaction Group specified by the A Operand. If the Transaction is already a member of the Transaction Group, no operation is performed.

Numeric Groups are distinct from Transaction Groups even if they have the same Group number.

Transactions are in no way restricted because of their membership in a Transaction Group. After membership is achieved, the entering Transaction proceeds to the Next Sequential Block. The only

way that a Transaction can be removed from a Group is to be terminated in a TERMINATE or ASSEMBLE Block, or to be chosen for removal in a REMOVE Block.

## Special Restriction

- ◆ A must be positive.

## Refuse Mode

A Transaction is never refused entry to a JOIN Block.

## Related Blocks

Transactions in Transaction Groups can be referenced by ALTER, EXAMINE, REMOVE, and SCAN Blocks. Numbers in numeric Groups can be referenced by EXAMINE and REMOVE Blocks.

## Related SNAs

- ◆ *GNEntnum* - Numeric Group count. *GNEntnum* returns the membership count of Numeric Group *Entnum*.
- ◆ *GTEntnum* - Transaction Group count. *GTEntnum* returns the membership count of Transaction Group *Entnum*.

## Related Windows

- ◆ Numeric Groups Snapshot - Picture of the state of the Numeric Groups in the simulation.
- ◆ Transaction Groups Snapshot - Picture of the state of the Transaction Groups in the simulation.

# LEAVE

**A LEAVE Block increases the accessible storage units at a Storage Entity.**

## LEAVE A,B

### Operands

**A** - Storage Entity name or number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Number of storage units. The default is 1. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

**LEAVE RepairMen,10**

In this example, when a Transaction enters the LEAVE Block, the available storage units at the Storage Entity named RepairMen is increased by 10.

## Action

When a Transaction enters a LEAVE Block, Operand A is evaluated and truncated, and the corresponding Storage Entity is found. If no such entity exists, an Error Stop occurs. Storage entities must be defined in STORAGE Commands.

The number of storage units to be freed is found by evaluating and truncating Operand B. If B was not used, then it is assumed to be 1.

Then the available storage at the Storage Entity is increased by the value of the B Operand. If the result exceeds the original storage capacity of the Storage Entity, an Error Stop occurs.

If no error occurs, the Transaction is scheduled for the Next Sequential Block and is placed on the Current Events Chain ahead of its priority peers.

When storage becomes available, the Delay Chain of the Storage Entity is examined in decreasing priority for Transactions whose demands can now be met. A "first-fit-with-skip" discipline is used. Successful Transactions are allowed to enter the ENTER Block which refused them and then are placed on the CEC behind their priority peers. This is done before the current Active Transaction in the LEAVE Block proceeds in the simulation. In this way, no other Transaction can buck the line of Transactions waiting on the Delay Chain of the Storage Entity. You can see this in the Blocks Window. You will see the Transaction enter the ENTER Block and come to rest. The current Active Transaction in the LEAVE Block will then resume its movement.

## Special Restrictions

- ◆ A must refer to a previously defined Storage Entity defined by a STORAGE Statement.
- ◆ B must be nonnegative.
- ◆ An attempt to free more storage than was defined will cause an Error Stop.

## Refuse Mode

A Transaction is never refused entry to a LEAVE Block.

## Related Command

A Storage Entity must be defined in a STORAGE Command before it can be updated by an LEAVE Block. The STORAGE Command must exist in the model, or must be sent to the Simulation Object interactively, before a Transaction can enter the LEAVE Block. Any attempt to do so before the Storage Entity is defined, cases an Error Stop.

A Storage Entity can be redefined by an interactive STORAGE Command.

## Related SNAs

- ◆ *REnnum* - Unused storage capacity. The storage content (or "token" spaces) available for use by entering Transactions at Storage Entity *Ennum*.
- ◆ *SEnnum* - Storage in use. *SEnnum* returns the amount of storage content (or "token" spaces) currently in use by entering Transactions at Storage Entity *Ennum*.
- ◆ *SAEnnum* - Average storage in use. *SAEnnum* returns the time weighted average of storage capacity (or "token" spaces) in use at Storage Entity *Ennum*.
- ◆ *SCEnnum* - Storage use count. Total number of storage units that have been entered in (or "token" spaces that have been used at) Storage Entity *Ennum*.
- ◆ *SEEnnum* - Storage empty. *SEEnnum* returns 1 if Storage Entity *Ennum* is completely unused, 0 otherwise.
- ◆ *SFEnnum* - Storage full. *SFEnnum* returns 1 if Storage Entity *Ennum* is completely used, 0 otherwise.
- ◆ *SREnnum* - Storage utilization. The fraction of total usage represented by the average storage in use at Storage Entity *Ennum*. *SREnnum* is expressed in parts-per-thousand and therefore returns an real value 0-1000, inclusively.

- ◆ *SME<sub>n</sub>* - Maximum storage in use at Storage Entity *En<sub>n</sub>*. The "high water mark".
- ◆ *STE<sub>n</sub>* - Average holding time per unit at Storage Entity *En<sub>n</sub>*.
- ◆ *SVE<sub>n</sub>* - Storage in available state. *SVE<sub>n</sub>* returns 1 if Storage Entity *En<sub>n</sub>* is in the available state, 0 otherwise.

## Related Window

- ◆ Storages Window - Online view of Storage Entity dynamics.

# LINK

**A LINK Block controls the placement of the Active Transaction on the User Chain of a Userchain Entity.**

## LINK A,B,C

### Operands

**A** - Userchain number. The Userchain Entity which may receive the entering Transaction. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Chain ordering. The placement of new Transactions on the Userchain. Required. The operand must be LIFO, FIFO, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**C** - Next Block location. The destination Block for Transactions which find the Link Indicator of the Userchain in the off state (reset). Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

### Example

#### LINK OnHold,FIFO

In this example, the Active Transaction is placed at the end of the User Chain Entity named OnHold. It is removed from all chains except Transaction Groups and Interrupt Chains. In other words, preemptions are not cleared. The Transaction remains on the User Chain until some other Transaction enters an UNLINK Block and specifically removes it. In the present example, the Transaction is placed at the end of the User Chain named OnHold.

### Action

The simplest operation of a LINK Block occurs when the C Operand is not used. In this case, the entering Transaction is always placed on the User Chain specified by Operand A. It is removed from all other chains except transaction groups and interrupt chains.

The placement of the Transaction in the User Chain is controlled by Operand B. If FIFO (First-In-First-Out) is used for Operand B, newly arriving Transactions are placed at the tail of the User Chain. If LIFO (Last-In-First-Out) is used for Operand B, newly arriving Transactions are placed at the head of the User Chain. Any other item used in Operand B is evaluated for the Active Transaction and again for each Transaction on the User Chain, beginning at the front, until the value for the Active Transaction is greater. When a Transaction whose value is less than that of the Active Transaction is found, the Active Transaction is placed on the User Chain immediately in front of it. If the value for the Active Transaction never exceeds the value for an occupant of the User Chain, the Active Transaction is placed at the end of the chain. This leads to a descending order. P class SNAs are an exception. They are queued in ascending order of Parameter value.

If you do not use LIFO or FIFO in Operand B, you will normally use a Transaction oriented SNA such as PR, M1 or a class P SNA. However, indirect addressing may also be useful. If PR is used, the

Transactions are placed in priority order. And finally, if a Parameter number is specified, the Transaction is inserted into the chain immediately behind those Transactions whose Parameter value is less than that of the entering Transaction.

The situation is more complicated when the C Operand is used.

A flag called a "Link Indicator" is part of each Userchain Entity. The Link Indicator is useful for using a User Chain to control the queuing of Transactions on a resource. It is on (set) when the hypothetical resource is "busy" and off (reset) when the hypothetical resource is "not busy". When the Link Indicator of a Userchain Entity is off, if the C Operand is used, the LINK Block will not place the Transaction on the User Chain. The Link Indicator is useful for letting the first of several Transactions avoid the User Chain and for causing following Transactions to be placed on the User Chain.

If the C Operand is used, the entering Transaction will NOT be placed on the User Chain if the Link Indicator is off. Instead, the Transaction will proceed to the Block specified by C, and then the Link Indicator will be set. Following Transactions entering the LINK Block will go onto the User Chain.

The Link Indicator is manipulated by both LINK and UNLINK Blocks. It is turned off (reset) when an UNLINK Block finds the User Chain empty. It may be useful to think of the Link Indicator as representing the "busy condition" of a hypothetical resource. When the Link Indicator is set, the resource is "busy". When a Transaction finds the resource idle it should not have to wait (on the User Chain). The Transaction would proceed to the Block specified by Operand C and the LINK Block then sets the Link Indicator of the Userchain Entity.

Consider two Transactions arriving at a LINK Block, one after the other. If, for example, the first Transaction does not enter an UNLINK Block before the second Transaction arrives, the second Transaction would find the Link Indicator on (set), and would be placed on the User Chain. In this example, when the first Transaction enters an UNLINK Block, the second Transaction is removed from the User Chain. At this time, the Link Indicator remains on (set). Then, when the second Transaction enters an UNLINK Block, and no waiting Transactions are found on the User Chain, the Link Indicator is finally turned off (reset).

User Chains allow you manipulate the queuing mechanisms of Transactions to a much more detailed level than do Facilities or Storages. It is possible to implement very complicated scheduling algorithms using LINK and UNLINK Blocks.

User Chains can be used to reduce the amount of computer time wasted on unsuccessful tests associated with GATE, TEST, TRANSFER BOTH, and TRANSFER ALL Blocks. You can create faster simulations by placing blocked Transactions on User Chains when there is no possibility of a successful condition test. Then, you must introduce each Transaction back into its test Block when an event occurs which might unblock the Transaction.

## Special Restrictions

- ◆ A, B, and C, if specified, must be positive.
- ◆ C, if specified, must be a Block location in the simulation.

## Refuse Mode

A Transaction is never refused entry to a LINK Block.

## Related SNAs

- ◆ *CAEnnum* - Average Userchain content. The time weighted average number of chained Transactions for Userchain *Ennum*.
- ◆ *CCEnnum* - Total Userchain entries. The cumulative count of all Transactions chained to Userchain *Ennum*.
- ◆ *CHEnnum* - Current Userchain content. The current number of Transactions chained to Userchain *Ennum*.
- ◆ *CMEnnum* - Maximum Userchain content. The maximum number of Transactions chained to Userchain *Ennum*. The "high water mark".
- ◆ *CTEnnum* - Average Userchain residence time. The average duration of a Transaction at Userchain *Ennum*.

## Related Window

❖ Userchains Snapshot - Picture of the state of the Userchain Entities in the simulation.

# LOGIC

A LOGIC Block changes the state of a Logicswitch entity.

## LOGIC O A

### Operands

O - Logic operator. Required. The operator must be S, R, or I.

A - Logicswitch Entity number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

#### LOGIC S PowerSwitch

In this example, the Logicswitch Entity named PowerSwitch is left in the true or "set" state.

### Action

A LOGIC Block is used to set, reset, or invert the state of a Logicswitch Entity. A Logicswitch Entity has two states, on (set, or 1) and off (reset, or 0). If the logic operator is S or R, the Logicswitch Entity specified by the A Operand is left in the set or reset state, respectively.

If the logic operator is I, the state of the Logicswitch Entity specified by the A Operand is inverted. This means that if it is found to be set, it will be reset. If it is found to be reset, it will be set.

### Logical Operators

A logical operator is required. It may be S, R, or I with the following effect:

- ❖ S - The logic switch is left in the "set" or on state.
- ❖ R - The logic switch is left in the "reset" or off state.
- ❖ I - The logic switch is inverted.

### Refuse Mode

A Transaction is never refused entry to a LOGIC Block.

### Related SNA

- ❖ LSEnnum - Logic switch set. LSEnnum returns 1 if Logic Switch Entity *Entnum* is in the "set" state, 0 otherwise.

### Related Window

- ❖ Logicswitches Window - Online view of Logicswitch Entity dynamics.

# LOOP

**A LOOP Block modifies a Parameter and controls the destination of the Active Transaction based on the result.**

## LOOP A,B

### Operands

**A** - Parameter containing count. required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Block number. Next Block if count nonzero after decrement. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

#### LOOP Customer\_Count,Start\_Over

Let us assume in this example that the Start\_Over Block precedes the LOOP Block in the simulation. In this example, when a Transaction enters the LOOP Block its Parameter named Customer\_Count is decremented by 1. If the result is nonzero, the Transaction proceeds to Block location Start\_Over. This causes the Transaction to continue to loop until the value of the Parameter named Customer\_Count is 0. The Transaction then proceeds to the Next Sequential Block.

### Action

When a Transaction enters a LOOP Block, Operand A is evaluated, truncated, and used to find the Transaction Parameter with that number. If there is no such Parameter, an Error Stop occurs. Otherwise the value of the Parameter is decreased by 1.

If the new value of the Parameter is greater than zero and the B Operand is specified, the Transaction is scheduled for the location specified in the B Operand. Otherwise, the Transaction proceeds to the Next Sequential Block.

### Special Restriction

- ◆ A must be positive.

### Refuse Mode

A Transaction is never refused entry to a LOOP Block.

### Related SNA

*PParameter* or *\*Parameter* - Parameter value. *PParameter* returns the value of Parameter *Parameter*.

### Related Windows

- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.

# MARK

**A MARK Block places an absolute clock time stamp into the Active Transaction or into its Parameter.**

## MARK A

### Operand

**A** - Parameter number. Parameter to receive value of system clock. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Examples

#### MARK Beginning

In this example, when a Transaction enters the MARK Block, its Transaction Parameter named Beginning is given a value equal to the value of the absolute system clock, AC1.

#### MARK

In this example, when a Transaction enters the MARK Block, its Mark Time is set equal to the value of the absolute system clock.

### Action

When a Transaction enters a MARK Block and Operand A was not specified, its Mark Time is set equal to the absolute system clock.

If Operand A was specified, it is evaluated numerically, and truncated. The Parameter of the Active Transaction with that number is found and given a value equal to the value of the absolute system clock. If the Parameter does not exist, it is created.

The Active Transaction then proceeds to the Next Sequential Block (NSB).

The time stamps set up by a MARK Block can be retrieved by M1 and MP class SNAs. M1 returns the "transit time", which is the absolute system clock minus the Transaction's Mark Time. SNAs in the class MP return a value equal to the absolute system clock minus the value of a Transaction Parameter.

### Special Restriction

◆ A, if specified, must be positive.

### Related SNAs

◆ *MPEntnum* - Transit time, Parameter. Current absolute system clock value minus value in Parameter *Entnum*.

◆ M1 - Transit time. M1 returns the absolute clock minus the "mark" time of the Transaction.

### Refuse Mode

A Transaction is never refused entry to a MARK Block.

### Related Windows

- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# MATCH

**A pair of MATCH Blocks cause Transactions to wait for each other.**

## MATCH A

### Operand

**A** - Block name or number. Conjugate MATCH Block to be tested for a matching (same Assembly Set) Transaction. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

```
A_Is_Done MATCH B_Is_Done
B_Is_Done MATCH A_Is_Done
```

This example shows two conjugate MATCH Blocks. They would normally be placed apart in the simulation, and one would eventually receive a Transaction in the same Assembly Set. If each MATCH Block were placed after a set of Blocks representing some process, the pair of related Transactions would proceed past the MATCH Blocks only when both Transactions had completed their respective processes.

### Action

When a Transaction enters a MATCH Block, Operand A is evaluated numerically, truncated, and the conjugate MATCH Block is found. If there is no such Block, an Error Stop occurs.

If the conjugate MATCH Block contains a Transaction (on its Match Chain) of the same Assembly Set as the Active Transaction, the related Transaction is removed from the Match Chain. If it is not currently preempted at any Facility Entity, it is placed on the Current Events Chain behind its priority peers. Similarly, if the Active Transaction is not currently preempted at any Facility it is placed on the CEC, but ahead of its peers.

If either matching Transaction is currently preempted at a Facility, it is not permitted to leave its MATCH Block until all preemptions have been cleared on its behalf.

If, when the Active Transaction enters the MATCH Block, no matching Transaction is found, it comes to rest on the Match Chain of the MATCH Block.

### Special Restrictions

- ◆ Operand A must be the location of a MATCH Block in the simulation.
- ◆ Transactions which are currently preempted are not permitted to leave MATCH Blocks.

### Refuse Mode

A Transaction is never refused entry to a MATCH Block.

## Related Blocks

- ◆ ADOPT - Set the Assembly Set of the Active Transaction.
- ◆ ASSEMBLE - Wait for and destroy Assembly Set members.
- ◆ GATHER - Wait for Assembly Set members.
- ◆ SPLIT - Create Transactions in the same Assembly Set.

## Related SNAs

- ◆ A1 - Assembly Set. Return the Assembly Set of the Active Transaction.
- ◆ MBE<sub>n</sub> - Match at Block. MBE<sub>n</sub> returns a 1 if there is a Transaction at Block <sub>n</sub> which is in the same Assembly Set as the Active Transaction. MBE<sub>n</sub> returns a 0 otherwise.

## Related Windows

- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# MSAVEVALUE

An MSAVEVALUE Block updates an element of a Matrix Entity.

### MSAVEVALUE A,B,C,D

#### Operands

**A** - Matrix Entity name or number, with optional + or -. Required. the operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*, followed by +, -, or *Null*.

**B** - Row number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**C** - Column number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**D** - Value to be stored, added, or subtracted. Required. The operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

#### Examples

**MSAVEVALUE Sales+,Part23,Cust77,6.234**

When a Transaction enters the MSAVEVALUE Block in this example, the element, of the Matrix Entity named Sales, with row equal to the value of Part23 and the column equal to the value of Cust77 is increased by 6.234. Normally, the row and column names would have appeared earlier in EQU Statements.

**MSAVEVALUE Parts,Part23,Description,"Zippo lighter"**

In this example, a string is assigned to a Matrix element. The Matrix Entity must be defined elsewhere in the Model, or interactively, by a MATRIX Command that defines the Parts Matrix Entity.

## Action

When a Transaction enters a MSAVEVALUE Block, Operand A is evaluated numerically and the Matrix Entity with that number or name is found. If no such Matrix Entity is found, an Error Stop occurs. Matrix Entities must be defined in MATRIX Commands.

Operands B and C are evaluated numerically to find the proper element in the Matrix Entity. If such an element does not exist, an Error Stop occurs.

Operand D is evaluated and used in the update operation. If Operand A has a + suffix, the operation will be addition to the numerical equivalent of Operand D; if -, it will be subtraction. If Operand A has no suffix, the matrix entity element will be assigned a new value equal to D.

A matrix is defined in a MATRIX Command. It can have up to 6 dimensions. However, only the first two dimensions can be accessed in an MSAVEVALUE Block. In this case, all missing coordinates are presumed to be equal to 1.

PLUS Procedures can access all elements of any matrix. If you need to use matrices of more than 2 dimensions, you will have to create one or more PLUS Procedures to access them. Matrices defined in a MATRIX Command have global scope and are known to all PLUS Procedures. In addition, temporary matrices with local scope can be created for the duration of a PLUS Procedure invocation. This is discussed further in Chapter 8.

## Refuse Mode

A Transaction is never refused entry to an MSAVEVALUE Block.

## Special Restriction

- ◆ MSAVEVALUE Blocks can only access the first cross section of a higher dimensional matrix. Indices of the third and higher dimensions are given the value 1. PLUS assignment statements can be used to access and modify all elements.

## Related SNA

- ◆ MXEnignum( $m,n$ ) - Matrix element value. The value in row  $m$ , column  $n$  of matrix Enignum is returned. The row and column values can only be Name, PosInteger, or P class SNA

## Related Windows

- ◆ Expressions Window - Online view of values of expressions.
- ◆ Matrix Window - Online view of the dynamics of a Matrix cross-section.
- ◆ Plot Window - Online view of a plot of up to 8 expressions.

# OPEN

An OPEN Block initializes a Data Stream.

**OPEN A,B,C**

## Operands

**A** - Data Stream descriptor. Required. The operand must be *Name*, *PosInteger*, *String*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**B** - Data Stream number. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*. Default is 1.

**C** - Alternate Destination Block name or number. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Example

```
OPEN ("MYFILE.TXT"),2,Error_Block
```

In this example, the OPEN Block creates type of Data Stream known as an I/O Stream, and gives it number 2 for identification. If an error occurs during the OPEN, an error code is stored internally and the Active Transaction proceeds to the Block labeled Error\_Block.

## Action

An OPEN Block causes a Data Stream to be created, and sets the Current Line Position to 1.

A Data Stream is a sequence of text lines used by a GPSS World simulation. Each Data Stream is identified by a unique number. There are 3 types of Data Stream:

1. Input/Output (I/O) Streams,
2. In-Memory Streams, and
3. Pipe Streams.

Operand A is evaluated as a string. If it is a null string, an In-Memory Stream is created. If it is a pipe name, such as "*pipe\mypipe*", a Pipe Stream is created. Otherwise an I/O Stream is created, and Operand A is presumed to be a file specification. If a path is not included in the file specification, it is assumed that the Project's Report directory is to be used. Projects are discussed in Chapter 1 of this manual.

If Operand B is used, it is evaluated numerically and used as the Data Stream number, for later reference by READ, WRITE, and CLOSE Blocks. The default is 1.

If Operand C is used, any error occurring during the OPEN causes the Active Transaction to proceed to the Block with that number.

In any case, if an error is detected, the error code is stored internally. A CLOSE Block can be used to retrieve the Error Code.

Chapter 4 (4.16) contains a full discussion of Data Streams, including the Error Code descriptions..

## Blocking Condition

The simulation is blocked while the Data Stream is initialized.

## Refuse Mode

A Transaction is never refused entry to a CLOSE Block.

## Related Blocks

- ◆ CLOSE - Shut down a Data Stream.
- ◆ READ - Retrieve a text line from a Data Stream.
- ◆ WRITE - Send a text line to a Data Stream.
- ◆ SEEK - Set the Current Line Position of a Data Stream.

# PLUS

**A PLUS Block evaluates an expression and, optionally, places the result into a Parameter.**

**PLUS A,B**

## Operands

**A** - Expression. Required. The operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**B** - Parameter number. Parameter to receive result. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Example

**PLUS (normal(2,100,2)+100.26),Result\_Parm**

## Action

When a Transaction enters a PLUS Block, Operand A is evaluated normally.

If Operand B is specified, it is evaluated numerically, truncated, and used to identify a Parameter of the Active Transaction. If no such Parameter exists, it is created. Then, the result from the evaluation of Operand A is placed into it.

PLUS Procedures without an explicit return value return integer 0.

## Refuse Mode

A Transaction is never refused entry to a PLUS Block.

## Related Windows

- ◆ Expressions Window - Online view of values of expressions.
- ◆ Plot Window - Online view of a plot of up to 8 expressions.

# PREEMPT

**A PREEMPT Block displaces a Transaction from ownership of a Facility Entity.**

**PREEMPT A,B,C,D,E**

## Operands

**A** - Facility name or number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**B** - Priority Mode. PR, for Priority Mode, or Interrupt Mode, if omitted. Optional. The operand must be *PR* or *Null*.

**C** - Block name or number. New destination for Transaction presently owning the Facility. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**D** - Parameter number. Parameter of preempted Transaction to receive residual time if preempted Transaction is removed from FEC. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**E** - REmove Mode. Removes preempted Transaction from contention for the Facility. Optional. The operand must be *RE*, or *Null*. If *RE*, you must specify a destination in Operand C.

## Examples

### **PREEMPT Teller1**

This is the simplest way to use the PREEMPT Block. When a Transaction enters this PREEMPT Block, it is given ownership of the Facility Entity unless it (the Facility) is currently preempted. If a Transaction must be displaced from ownership of the Facility, it is not permitted to reside on the FEC and its movement in the simulation is restricted.

### **PREEMPT Teller1,,Teller2,101,RE**

In this example, the PREEMPT Block operates in Interrupt Mode because the B Operand is omitted. This means that a Transaction can enter the Block and own the Facility if the Facility is not already owned by a preempting Transaction. Transaction priorities are ignored. When a Transaction is preempted, any remaining time in an ADVANCE Block is recorded in Parameter 101 of the preempted Transaction. The preempted Transaction is removed from contention for the Facility and sent to the Block labeled Teller2. An error will result if such a Transaction later tries to RELEASE or RETURN the Facility named Teller1, without owning it again.

### **PREEMPT Teller1,PR,Teller2**

In this example, the PREEMPT operates in Priority Mode. This means that a Transaction can enter the Block and own the Facility Entity if the Facility is not already owned by a Transaction of equal or higher priority. Any preempted Transaction is not removed from contention for the Facility. This means that such a Transaction can no longer exist on the FEC or leave ASSEMBLE, GATHER, or MATCH Blocks. Under these restrictions, preempted Transactions may continue to circulate in the simulation, and in this case are sent to the Block location Teller2. Such Transactions remain in contention for the Facility named Teller1, and normally will regain ownership of it. A preempted Transaction may RETURN or RELEASE a Facility even if it does not own it. This removes the Transaction from contention for the Facility by removing it from the Interrupt Chain of the Facility.

## Action

A PREEMPT Block behaves like a SEIZE Block when the Facility is idle. Otherwise, the PREEMPT Block operates in either Priority Mode or Interrupt Mode. This is determined by Operand B.

In Priority Mode, only a higher priority Transaction can displace the Transaction which owns the Facility. If a would-be preemptor is of insufficient priority, it is placed on the Delay Chain of the Facility in priority order.

In Interrupt Mode, if the Facility is already preempted, the Active Transaction is placed on the Pending Chain. Transactions on the Pending Chain are given ownership of the Facility in preference to preempted Transactions or to Transactions on the Delay Chain.

Operands C to E are concerned with what to do with the current owner of the Facility which is about to become preempted. Preempted Transactions are not permitted to exist on the FEC. Preempted Transactions which have unfinished time in an ADVANCE Block may be replaced on the FEC automatically by omitting the C and E operands. Alternately, if you choose to replace the Transaction on the FEC yourself, to resume the unfinished time, you must use Operand D and eventually send the preempted Transaction to an ADVANCE Block.

A preempted Transaction may be removed from contention for the Facility (i.e. removed from all chains of the Facility) by using the RE option in the E Operand. The RE option removes the restrictions placed on preempted Transactions due to preemption at this Facility, and makes any subsequent attempt to RELEASE or RETURN the Facility an Error Stop condition.

A preempted Transaction cannot exist on the FEC. A more detailed discussion of preemption is in Section 9.4. Any newly preempted Transaction in an ADVANCE Block which is on the FEC is removed from the FEC and the residual time duration is saved. If the D Operand is used, the residual time is also saved in a Transaction Parameter. If no such Parameter exists, one is created. The residual time is used to reschedule the Transaction on the FEC when it regains ownership of all Facilities for which it contends. Alternately, you may give a preempted Transaction a new Block to attempt to enter by using the C Operand.

A preempted Transaction remains in contention for a Facility even if it was displaced by the C Operand, unless RE is used in Operand E. If a preempted Transaction, which is still in contention for a Facility, attempts to enter a TERMINATE Block an Error Stop occurs. Such Transactions must enter a RELEASE or RETURN Block before they are permitted to TERMINATE. Alternately, if you intend to TERMINATE the preempted Transaction you could use the RE option to make sure it doesn't inadvertently regain ownership before termination. such a Transaction could not RELEASE or RETURN the Facility.

When a Transaction is displaced by using the C Operand, it is given a new Block destination and is dequeued from:

- ◆ FEC
- ◆ PENDING (INTERRUPT-MODE PREEMPT) CHAINS
- ◆ DELAY (MAJOR PRIORITY) CHAINS
- ◆ USER CHAINS
- ◆ RETRY CHAINS

and not dequeued from:

- ◆ CEC
- ◆ INTERRUPT (PREEMPTED) CHAINS
- ◆ GROUP CHAINS

When a Transaction is displaced from its present context, by using an alternate destination, it is removed from blocking conditions, but preemptions at other Facilities are not cleared.

When the C Operand is not used, a preempted Transaction taken off the FEC will be returned to it automatically. Preempted Transactions which have not been displaced using the C Operand are expected eventually to enter a RETURN or RELEASE Block in order to give up ownership of the Facility. If such a Transaction arrives at the RETURN or RELEASE before regaining ownership of the Facility, the Transaction is removed from contention for the Facility. No error condition occurs.

A Transaction can be preempted from any number of Facilities and continue to circulate in the simulation. However, it is subject to two restrictions:

- ◆ It will be refused entry to ADVANCE Blocks with positive time arguments.
- ◆ It will not be allowed to leave an ASSEMBLE, GATHER, or MATCH Block until all its preemptions have been cleared.

A Facility can be preempted any number of times. However, once a Transaction has been preempted, it cannot attempt to seize the Facility from which it has been preempted. A Transaction can be preempted from any number of Facilities.

## Special Restrictions

- ◆ If E is RE, C must be used.
- ◆ A Transaction may not preempt itself.

## Refuse Mode

A Transaction is refused entry to a PREEMPT Block if, in Interrupt Mode, the Facility is currently preempted. Such Transactions are placed at the end of the Facility's Pending Chain.

A Transaction is refused entry to a PREEMPT Block if, in Priority Mode, the Facility is currently owned by a Transaction of priority equal to or greater than that of the Active Transaction. The Active Transaction is placed in priority order on the Facility's Delay Chain.

A Transaction is refused entry to a PREEMPT Block if the Facility is in the unavailable state. Such Transactions are placed on the Facility's Delay Chain, in priority order, FIFO within priority.

## Related Blocks

- ◆ DISPLACE - Move any Transaction.
- ◆ FAVAIL - Place Facility in the available state.
- ◆ FUNAVAIL - Place Facility in the unavailable state.
- ◆ RELEASE - Give up ownership and remove contention for a Facility.
- ◆ RETURN - Give up ownership and remove contention for a Facility.
- ◆ SEIZE - Acquire or wait for ownership of a Facility.

## Related SNAs

- ◆  $FEnignum$  - Facility busy. If Facility  $Enignum$  is currently busy,  $FEnignum$  returns 1. Otherwise  $FEnignum$  returns 0.
- ◆  $FCEnignum$  - Facility capture count. The number of times Facility  $Enignum$  has become owned by a Transaction.
- ◆  $FIEnignum$  - Facility  $Enignum$  interrupted. If Facility  $Enignum$  is currently preempted,  $FIEnignum$  returns 1. Otherwise  $FIEnignum$  returns 0.
- ◆  $FREnignum$  - Facility utilization. The fraction of time Facility  $Enignum$  has been busy.  $FREnignum$  is expressed in parts-per-thousand and therefore returns an real value 0-1000, inclusively.
- ◆  $FTEnignum$  - Average Facility holding time. The average time Facility  $Enignum$  is owned by a capturing Transaction.
- ◆  $FVEnignum$  - Facility in available state.  $FV Enignum$  returns 1 if Facility  $Enignum$  is in the available state, 0 otherwise.

## Related Windows

- ◆ Facilities Window - Online view of Facility Entity dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# PRIORITY

**A PRIORITY Block sets the priority of the Active Transaction.**

## PRIORITY A,B

### Operands

**A** - New priority value. Required. The operand must be *Name*, *integer*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Buffer option. Places Active Transaction behind priority peers on CEC. Optional. The operand must be *BU* or *Null*.

## Example

### PRIORITY 10

In this example any entering Transaction is made to have a priority of 10.

## Action

When a Transaction enters a PRIORITY Block, Operand A is evaluated numerically, truncated, and assigned to the priority of the Active Transaction.

The Transaction is scheduled for the Next Sequential Block and is placed on the CEC according to its new priority. If the BU option was used in Operand B, the Transaction is placed behind its priority peers on the CEC. Otherwise, it is placed in front of its priority peers.

Transaction priorities are integers. When a Transaction is created without an explicit priority, it is given a priority of 0, by default. GPSS World is most efficient when priorities used in the simulation are contiguous. For example, use 0, 1, 2, instead of -200, 0, 23.

## Refuse Mode

A Transaction is never refused entry to a PRIORITY Block.

## Related SNA

- ◆ PR - Transaction priority. The value of the priority of the Active Transaction.

## Related Windows

- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# QUEUE

**A QUEUE Block updates Queue Entity statistics to reflect an increase in content.**

## QUEUE A,B

### Operands

**A** - Queue Entity name or number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Number of units by which to increase the content of the Queue Entity. Default value is 1. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Example

### QUEUE WaitingLine

In this example the content of the Queue Entity named WaitingLine is increased by one and the associated statistics accumulators are updated.

## Action

When a Transaction enters a QUEUE Block, Operand A is evaluated, truncated, and used to find the Queue Entity with that number. The Queue Entity is created if necessary.

Operand B specifies the value to be used to increase the content of the Queue Entity. If B was specified, Operand B is evaluated, truncated, and used as the increment. If B was not specified, the value of 1 is used.

Finally, the statistics accumulated on behalf of the Queue Entity are updated.

## Special Restriction

- ◆ B, if specified, must be positive.

## Refuse Mode

A Transaction is never refused entry to a QUEUE Block.

## Related SNAs

- ◆ *QEnignum* - Current queue content. The current count value of Queue Entity *Enignum*.
- ◆ *QAEnignum* - Average queue content. The time weighted average count for Queue Entity *Enignum*.
- ◆ *QCEnignum* - Total queue entries. The sum of all queue entry counts for Queue Entity *Enignum*.
- ◆ *QMEnignum* - Maximum queue content. The maximum count (high water mark) of Queue Entity *Enignum*.
- ◆ *QTEEnignum* - Average queue residence time. The time weighted average of the count for Queue Entity *Enignum*.
- ◆ *QXEnignum* - Average queue residence time excluding zero entries. The time weighted average of the count for Queue Entity *Enignum* not counting entries with a zero residence time.
- ◆ *QZEnignum* - Queue zero entry count. The number of entries of Queue Entity *Enignum* with a zero residence time.

## Related Window

- ◆ Queues Window - Online view of Queue Entity dynamics.

# READ

**A READ Block retrieves a text line from a Data Stream.**

### READ A,B,C

## Operands

**A** - Transaction Parameter. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**B** - Data Stream number. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*. Default is 1.

**C** - Alternate Destination Block name or number. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Example

**READ Text\_Parm,1,Done**

In this example, the READ Block retrieves a text line from Data Stream number 1 and places a copy in the Transaction Parameter Text\_Parm. If no such Parameter exists, it is created. If an error occurs the Active Transaction proceeds to the Block labeled Done, otherwise it proceeds to the Next Sequential Block.

## Action

When a Transaction enters a READ Block, Operand A is evaluated numerically, truncated, and used to identify the Transaction Parameter which will receive the text line. If a text line is received successfully and no Parameter exists for the Active Transaction, one is created.

If Operand B is used, it is evaluated numerically, truncated, and used as the Data Stream Entity number. This must be a positive integer. If Operand B is not used, Data Stream number 1 is assumed.

If Operand C is used, any error occurring during the READ causes the Active Transaction to proceed to the Block with that number.

In any case, if an error is detected, the error code is stored internally. A CLOSE Block can be used to retrieve the Error. Chapter 4 (4.16) contains a full discussion of Data Streams, including the Error Code descriptions.

If the Data Stream is a Pipe Stream, the text line is read from the named pipe and returned to the Transaction Parameter. The simulation is blocked while this occurs.

If the Data Stream is not a Pipe Stream, the text line is determined by the Current Line Position, a 1-relative line index associated with the Data Stream. In this case, the line indicated by the Current Line Position, even if the line is a null string, is returned to the Transaction Parameter as a result of the READ. Then, the Current Line Position is incremented, i.e. move to the next line number. If there is no line to be read, no data is returned and the Active Transaction is sent to the Alternate Destination Block without any error code being stored.

Reads can be computationally expensive. You can speed processing by using a large amount of data on a small number of text lines.

## Further Discussion

Chapter 4 (4.16) contains a full discussion of Data Streams under the Section entitled, Data Streams.

## Refuse Mode

A Transaction is never refused entry to a READ Block.

## Blocking Condition

The simulation is blocked while READ retrieves the text line.

## Related Blocks

◆ OPEN - Create a Data Stream.

- ◆ CLOSE - Shut down a Data Stream.
- ◆ WRITE - Send a text line to a Data Stream.
- ◆ SEEK - Set the Current Line Position of a Data Stream.

# RELEASE

**A RELEASE Block releases ownership of a Facility, or removes a preempted Transaction from contention for a Facility.**

## RELEASE A

### Operand

A - Facility number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

#### RELEASE Teller1

In this example, when a Transaction which owns the Facility Entity named Teller1 enters the RELEASE Block, it gives up ownership to the Facility.

### Action

When a Transaction enters a RELEASE Block it removes itself from contention for the Facility. This can happen in two ways.

If the Active Transaction owns the Facility, it gives up ownership and proceeds to the Next Sequential Block.

If the Active Transaction has been preempted from ownership of the Facility, it is removed from the Interrupt Chain of the Facility. Ownership is not affected, because some other Transaction is the owner. If the Active Transaction is presently clear of all preemptions, it may now move normally in the simulation.

In any case, the Active Transaction is removed from ownership of or contention for the Facility and attempts to enter the Next Sequential Block. If it neither owns, nor is preempted at the Facility, an Error Stop occurs.

If the Active Transaction gives up ownership of the Facility, the next owner is taken from the Pending Chain, the Interrupt Chain, and finally the Delay Chain. If there are now Interrupt Mode PREEMPTS pending at this Facility, the first is given ownership of the Facility. Otherwise, ownership is returned to the most recently preempted Transaction. If both the Pending Chain (waiting Interrupt Mode PREEMPTS) and the Interrupt Chain (preempted Transactions) are empty, the highest priority Transaction on the normal Delay Chain is given ownership. If there are no Transactions waiting, the Facility becomes idle.

When a new owner is chosen from the Delay Chain or the Pending Chain, it enters the SEIZE or PREEMPT Block immediately, and then is scheduled by placing it on the CEC behind its priority peers. After this entry, the current Active Transaction in the RELEASE Block continues its movement.

### Special Restriction

- ◆ An entering Transaction must own, or currently be preempted at, the Facility. Otherwise an Error Stop occurs.

## Refuse Mode

A RELEASE Block never refuses entry to a Transaction.

## Related SNAs

- ◆  $FEnignum$  - Facility busy. If Facility  $Entnum$  is currently busy,  $FEnignum$  returns 1. Otherwise  $FEnignum$  returns 0.
- ◆  $FCEnignum$  - Facility capture count. The number of times Facility  $Entnum$  has become owned by a Transaction.
- ◆  $FIEnignum$  - Facility  $Entnum$  interrupted. If Facility  $Entnum$  is currently preempted,  $FIEnignum$  returns 1. Otherwise  $FIEnignum$  returns 0.
- ◆  $FREnignum$  - Facility utilization. The fraction of time Facility  $Entnum$  has been busy.  $FREnignum$  is expressed in parts-per-thousand and therefore returns an real value 0-1000, inclusively.
- ◆  $FTEnignum$  - Average Facility holding time. The average time Facility  $Entnum$  is owned by a capturing Transaction.
- ◆  $FVEnignum$  - Facility in available state.  $FV Enignum$  returns 1 if Facility  $Entnum$  is in the available state, 0 otherwise.

## Related Window

- ◆ Facilities Window - Online view of Facility Entity dynamics.

# REMOVE

**A REMOVE Block removes members from a Numeric Group or a Transaction Group.**

**REMOVE O A,B,C,D,E,F**

## Operands

**O** - Conditional operator. Relationship of D to E for removal to occur. These choices are explained below. Optional. The operator must be *Null*, E, G, GE, L, LE, MAX, MIN, or NE.

**A** - Group number. Group from which a member or members will be removed. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Removal limit. The maximum number of Transactions to be removed. Optional. The operand must be ALL, *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**C** - Numeric value. Numeric value to be removed from a Numeric Group. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**D** - Test value. PR or Parameter number. The member attribute which determines whether each Group member Transaction should be removed, or PR to use the Transaction priority for the determination. It is evaluated with respect to the Transaction Group member. Optional. The operand must be PR or *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**E** - Reference value. The value against which the D Operand is compared. The reference value is evaluated with respect to the Active Transaction. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**F** - Block number. The alternate destination for the entering Transaction. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

## Example

### REMOVE Inventory

This is the simplest way to use the REMOVE Block. The Transaction entering the REMOVE Block is tested for membership in the Transaction Group named Inventory. If the Transaction is a member it is removed from the Group.

### REMOVE G 3,10,,300,11.6,Jump\_Block

In this example, Transaction Group 3 is scanned for Transactions which have a value in their Parameter 300 which is greater than 11.6. The first 10 Transactions which meet the test are removed from the Transaction Group. If 10 Transactions cannot be found which pass the test, the entering Transaction attempts to enter the Block labeled Jump\_Block. Otherwise, it proceeds to the Next Sequential Block.

## Action

A REMOVE Block excludes numeric values from membership in a Numeric Group, or Transactions from membership in a Transaction Group. Transactions are not displaced from their context. However, there is one exception. The Transaction entering the REMOVE Block may be redirected according to the F Operand.

A REMOVE Block operates in Numeric Mode if Operand C is used. In Numeric Mode, operands A and C are evaluated numerically and the number specified by C is tested for membership in the numeric Group specified by A, which is truncated. If the result from Operand C is a member of the Numeric Group, it is removed from the Group. If the numeric value was not a member of the numeric Group and the F Operand is used, the entering Transaction proceeds to the Block specified by F. Otherwise, the entering Transaction proceeds to the next sequential Block. Only operands A, C, and F can be used in Numeric Mode. There is some loss of efficiency when non-integers are used in Numeric Groups.

A REMOVE Block operates in Transaction Mode if Operand C is not used. In Transaction Mode, there are several options available to select the Transaction(s) to be removed from the Transaction Group. If you do not specify operands B, D, or E, then only the Transaction entering the REMOVE Block is removed. This mode of operation is called self removal. Otherwise, a "group scan" is performed.

In a group scan, you may test each Transaction for removal by using the conditional operator and/or operands D and E. Also you may limit the number of Transactions to be removed by the B Operand.

Operand D always refers to the Transaction Group member under test. Notice that any SNA may be used in Operand D. Any SNA which requires a Transaction for its evaluation uses the current Transaction Group member under test. The result returned by any SNA other than PR is used as a Parameter number whose value is returned as the final result.

In a group scan you may use a conditional operator to specify the relationship between the Transaction attribute (Operand D) and the reference value (Operand E) which will initiate a removal of the Transaction. Both are evaluated numerically. The default for the conditional operator is E for equality. If you use no conditional operator, but you use Operand D and Operand E, the values must be equal for the Transaction to be removed from the Transaction Group. If MIN or MAX is used as the conditional operator, all Transactions with the greatest or least attribute (Operand D) are removed, up to the limit count (Operand B).

The B Operand cuts off the group scan when it equals the number of Transactions that have been removed. The default is ALL. If there is no attribute test, that is, if D is not specified, Transactions are removed until the removal count equals B or until the Group is empty.

The F Operand is used in both Numeric Mode and Transaction Mode. It indicates an alternate destination to be taken by the entering Transaction when an exception condition occurs. The F Operand is used for the destination Block under the following conditions:

- ◆ In Numeric Mode, if the numeric value (C Operand) was not a member of the Group.
- ◆ In Self Removal Mode, if the Transaction entering the REMOVE Block was not a member of the Transaction Group.
- ◆ In Group Scan Mode, if no Transaction is removed.
- ◆ In Group Scan Mode, if the removal count specified by B cannot be reached.

If the F Operand is not used, the entering Transaction always goes to the Next Sequential Block.

## Conditional Operators

The conditional operator may be E, G, GE, L, LE, MAX, MIN, or NE. If no conditional operator is used, E (equality) is assumed. When the condition is true, the Transaction being tested is removed from the Group. The conditions are defined as follows:

- ◆ E - The Transaction attribute specified by Operand D must be equal to the reference value specified by Operand E for the Transaction to be removed from the Group.
- ◆ G - The Transaction attribute specified by Operand D must be greater than the reference value specified by Operand E for the Transaction to be removed from the Group.
- ◆ GE - The Transaction attribute specified by Operand D must be greater than or equal to the reference value specified by Operand E for the Transaction to be removed from the Group.
- ◆ L - The Transaction attribute specified by Operand D must be less than the reference value specified by Operand E for the Transaction to be removed from the Group.
- ◆ LE - The Transaction attribute specified by Operand D must be less than or equal to the reference value specified by Operand E for the Transaction to be removed from the Group.
- ◆ MAX - The Transaction attribute specified by Operand D must be equal to the largest such attribute of all Transactions in the Group for the Transaction to be removed from the Group.
- ◆ MIN - The Transaction attribute specified by Operand D must be equal to the smallest such attribute of all Transactions in the Group for the Transaction to be removed from the Group.
- ◆ NE - The Transaction attribute specified by Operand D must be unequal to the reference value specified by Operand E for the Transaction to be removed from the Group.

If no conditional operator is used in Group Scan Mode, E is assumed.

## Special Restrictions

- ◆ If a Numeric Group is referenced, you must not use a conditional operator.
- ◆ If a Numeric Group is referenced, you must not use Operand B, D, or E.
- ◆ If Operand D is used, then you must use Operand E or else you must use the conditional operator MIN or MAX.
- ◆ If Operand E is used, you must use Operand D.
- ◆ If MIN or MAX is used for the conditional operator, Operand D must be used and Operand E must not be used.

## Refuse Mode

A Transaction is never refused entry to a REMOVE Block.

## Related Blocks

Transactions and numbers are added to Groups by JOIN Blocks. Transactions in Groups can be referenced by ALTER, EXAMINE, REMOVE, and SCAN Blocks. Numbers in Numeric Groups can be referenced by EXAMINE Blocks.

## Related SNAs

◆ **GNEntnum** - Numeric Group count. *GNEntnum* returns the membership count of Numeric Group *Entnum*.

◆ **GTEntnum** - Transaction Group count. *GTEntnum* returns the membership count of Transaction Group *Entnum*.

## Related Windows

◆ Numeric Groups Snapshot - Picture of the state of the Numeric Groups in the simulation.

◆ Transaction Groups Snapshot - Picture of the state of the Transaction Groups in the simulation.

# RETURN

**A RETURN Block releases ownership of a Facility, or removes a preempted Transaction from contention for a Facility.**

## RETURN A

### Operand

**A** - Facility number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

#### RETURN Teller1

In this example, when a Transaction which owns the Facility named Teller1 enters the RETURN Block, it gives up ownership of the Facility.

### Action

When a Transaction enters a RETURN Block it removes itself from contention for the Facility. This can happen in two ways.

If the Active Transaction owns the Facility Entity, it gives up ownership and proceeds to the Next Sequential Block.

If the Active Transaction has been preempted from ownership of the Facility, it is removed from the Interrupt Chain of the Facility. Ownership is not affected, because some other Transaction is the owner. If the Active Transaction is presently clear of all preemptions, it may now move normally in the simulation.

In any case, the Active Transaction is removed from ownership of or contention for the Facility and attempts to enter the Next Sequential Block. If it neither owns, nor is preempted at the Facility, an Error Stop occurs.

If the Active Transaction gives up ownership of the Facility, the next owner is taken from the Pending Chain, the Interrupt Chain, and finally the Delay Chain. If there are now Interrupt Mode PREEMPTS pending at this Facility, the first is given ownership of the Facility. Otherwise, ownership is returned to the most recently preempted Transaction. If both the Pending Chain (waiting Interrupt Mode PREEMPTS) and the Interrupt Chain (preempted Transactions) are empty, the highest priority Transaction on the normal Delay Chain is given ownership. If there are no Transactions waiting, the Facility becomes idle.

When a new owner is chosen from the Delay Chain or the Pending Chain, it enters the SEIZE or PREEMPT Block immediately, and then is scheduled by placing it on the CEC behind its priority peers. After this entry, the current Active Transaction in the RETURN Block continues its movement.

## Special Restriction

- ◆ An entering Transaction must own, or currently be preempted at, the Facility. Otherwise an Error Stop occurs.

## Refuse Mode

A RETURN Block never refuses entry to a Transaction.

## Related SNAs

- ◆  $FEnignum$  - Facility busy. If Facility  $Entnum$  is currently busy,  $FEnignum$  returns 1. Otherwise  $FEnignum$  returns 0.
- ◆  $FCEnignum$  - Facility capture count. The number of times Facility  $Entnum$  has become owned by a Transaction.
- ◆  $FIEnignum$  - Facility  $Entnum$  interrupted. If Facility  $Entnum$  is currently preempted,  $FIEnignum$  returns 1. Otherwise  $FIEnignum$  returns 0.
- ◆  $FREnignum$  - Facility utilization. The fraction of time Facility  $Entnum$  has been busy.  $FREnignum$  is expressed in parts-per-thousand and therefore returns an real value 0-1000, inclusively.
- ◆  $FTEnignum$  - Average Facility holding time. The average time Facility  $Entnum$  is owned by a capturing Transaction.
- ◆  $FVEnignum$  - Facility in available state.  $FV Entnum$  returns 1 if Facility  $Entnum$  is in the available state, 0 otherwise.

## Related Window

- ◆ Facilities Window - Online view of Facility Entity dynamics.

# SAVAIL

**A SAVAIL Block ensures that a Storage Entity is in the available state.**

## SAVAIL A

### Operand

A - Storage name or number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

#### SAVAIL MotorPool

In this example, when a Transaction enters the SAVAIL Block, the Storage Entity MotorPool is assured to be in the available state.

### Action

An SAVAIL Block ensures that a Storage Entity is in the available state. If any Transactions are waiting on the Delay Chain of the Storage, they are given a chance to have their storage requests satisfied by the Storage Entity according to the first-fit-with-skip discipline. Those Transactions whose storage requests cannot be satisfied remain on the Delay Chain of the Storage Entity.

If the Storage Entity is already in the available state, the SAVAIL Block has no effect.

## Refuse Mode

A Transaction is never refused entry to a SAVAIL Block.

## Related Command

A Storage Entity must be defined in a STORAGE Command before it can be updated by an SAVAIL Block. The STORAGE Command must exist in the model, or must be sent to the Simulation Object interactively, before a Transaction can enter the SAVAIL Block. Any attempt to do so before the Storage Entity is defined, cases an Error Stop.

A Storage Entity can be redefined by an interactive STORAGE Command.

## Related SNAs

- ◆ *REntrnum* - Unused storage capacity. The storage content (or "token" spaces) available for use by entering Transactions at Storage Entity *Entnum*.
- ◆ *SEntrnum* - Storage in use. *SEntrnum* returns the amount of storage content (or "token" spaces) currently in use by entering Transactions at Storage Entity *Entnum*.
- ◆ *SAEntrnum* - Average storage in use. *SAEntrnum* returns the time weighted average of storage capacity (or "token" spaces) in use at Storage Entity *Entnum*.
- ◆ *SCEntrnum* - Storage use count. Total number of storage units that have been entered in (or "token" spaces that have been used at) Storage Entity *Entnum*.
- ◆ *SEEntrnum* - Storage empty. *SEEntrnum* returns 1 if Storage Entity *Entnum* is completely unused, 0 otherwise.
- ◆ *SFEntrnum* - Storage full. *SFEntrnum* returns 1 if Storage Entity *Entnum* is completely used, 0 otherwise.
- ◆ *SREntrnum* - Storage utilization. The fraction of total usage represented by the average storage in use at Storage Entity *Entnum*. *SREntrnum* is expressed in parts-per-thousand and therefore returns an real value 0-1000, inclusively.
- ◆ *SMEntrnum* - Maximum storage in use at Storage Entity *Entnum*. The "high water mark".
- ◆ *STEntrnum* - Average holding time per unit at Storage Entity *Entnum*.
- ◆ *SVEntrnum* - Storage in available state. *SVEntrnum* returns 1 if Storage Entity *Entnum* is in the available state, 0 otherwise.

## Related Window

- ◆ *Storages Window* - Online view of Storage Entity dynamics.

# SAVEVALUE

**A SAVEVALUE Block changes the value of a Savevalue Entity.**

## SAVEVALUE A,B

## Operands

**A** - Savevalue Entity number. Required. May be followed by + or - to indicate addition or subtraction to existing value. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*, followed by +, -, or *Null*.

**B** - The value to be stored, added, or subtracted. Required. The operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Examples

**SAVEVALUE Account,99.95**

In this example, the Savevalue Entity named Account takes on the value 99.95.

**SAVEVALUE The\_Bard,"A rose by any other name ..."**

In this example, the Savevalue Entity named The\_Bard is assigned a string. If the Savevalue Entity does not exist, it is created.

## Action

An SAVEVALUE Block is used to assign, increment, or decrement the value of a Savevalue Entity.

The A Operand is evaluated numerically, truncated, and used as the Savevalue Entity number.

Operand B is evaluated and used to determine the new value for the Savevalue Entity. If Operand A is followed by +, then the numeric equivalent of Operand B is added to the numeric equivalent of the old value. If Operand A is followed by -, then the numeric equivalent of Operand B is subtracted from the numeric equivalent of the old value. If Operand A is not followed by a sign, the old value of the SAVEVALUE is replaced by Operand B.

## Refuse Mode

A Transaction is never refused entry to a SAVEVALUE Block.

## Related SNA

◆ *XEntnum* - Savevalue. The value of Savevalue *Entnum* is returned.

## Related Windows

◆ Expressions Window - Online view of values of expressions.

◆ Plot Window - Online view of a plot of up to 8 expressions.

◆ Savevalues Window - Online view of Savevalue Entity dynamics.

# SCAN

**A SCAN Block passes information from a Transaction Group member to the Active Transaction.**

**SCAN O A,B,C,D,E,F**

## Operands

**O** - Conditional operator. Relationship of B to C for the Transaction Group member to be chosen. These choices are explained below. Optional. The operator must be *Null*, E, G, GE, L, LE, MAX, MIN,

or NE.

**A** - Transaction Group number. Group whose members will be scanned. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Test value. PR or Parameter number. The member attribute which determines whether the Group member Transaction should be selected. It is evaluated with respect to the Transaction Group member. Optional. The operand must be PR or Null, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**C** - Reference value. The value against which the B Operand is compared. The default is 0. Optional. The operand must be Null, *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**D** - Retrieved value. PR or Parameter number. The member attribute which is to be assigned to a Parameter of the Active Transaction. It is evaluated with respect to the Transaction Group member. Required. The operand must be PR, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**E** - Receiving Parameter number. The Parameter number of the entering Transaction which will receive the value retrieved from Operand D. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**F** - Alternate Block number. The alternate destination for the entering Transaction. Optional. The operand must be Null, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Examples

**SCAN MAX Inventory,P\$Price,,P\$PartNumber,100**

In this example, all Transactions in the Transaction Group Inventory are examined in order to find the Parameter named Price with the greatest value. The first Transaction with a maximal Price Parameter value is then selected for evaluation of the D Operand. The value of the PartNumber Parameter of the selected Transaction Group member is assigned to Parameter number 100 of the Active Transaction. If the Transaction Group is empty, no action occurs.

**SCAN E Lot11,PartNum,127,Price,Price,Phone**

In this example, when a Transaction enters the SCAN Block, the Transaction Group named Lot11 is scanned for the first Transaction which has a value of 127 in its Parameter named PartNum. If such a Transaction is found, the value in its Parameter named Price is transferred to the corresponding Parameter of the Transaction entering the SCAN Block. The entering Transaction then proceeds to the Next Sequential Block. If no such Transaction Group member is found, the entering Transaction proceeds to the Block labeled Phone. Operands D and E need not specify the same Transaction Parameter.

## Action

A SCAN Block finds the first Transaction in a Group which passes all tests, and stores one of its attributes in a Parameter of the Active Transaction. If no such Parameter exists for the Active Transaction, one is created. If an appropriate Transaction Group member cannot be found, no value is stored.

The Transaction Group member may be chosen on the basis of a test of one of its attributes. This is done by using operands B, C, and/or a conditional operator. If no such test is used, the first Transaction in the Group, if any, is selected.

When a Transaction Group member is found to satisfy a test, its attribute, which is specified by Operand D, is copied into the Parameter of the Active Transaction, specified by the E Operand. In this case, the Active Transaction always proceeds to the Next Sequential Block.

If no Transaction is found to satisfy the requirements, the Active Transaction may optionally be directed to the Block specified by the F Operand. Otherwise, it proceeds to the Next Sequential Block.

If you use operands B, C, or a conditional operator, the first Transaction Group member to pass the test is selected. Operand B specifies which attribute of the member Transactions is to be tested. It

may be compared to the minimum or the maximum of all such Group member attributes by using MIN or MAX as the conditional operator. The first Transaction in the Transaction Group which has the maximum or minimum attribute is selected. If you use MIN or MAX, you must not use Operand C.

You may compare the Group member attribute to Operand C, with or without a conditional operator. In this case, the conditional operator must not be MIN or MAX. The default for Operand C is 0.

Operands B and D always refer to the Transaction Group member under test. Notice that any SNA may be used. Any SNA which requires a Transaction for its evaluation uses the current Transaction Group member under test. The result returned by any SNA, other than PR, is used as a Parameter number, whose value is returned as the final result.

The default for the conditional operator is E for equality. If you use no conditional operator, but you use Operand B and Operand C, the values must be equal for the member Transaction attribute to be selected.

The F Operand indicates an alternate destination Block to be taken by the entering Transaction when no Transaction is found which satisfies the conditions specified. If F is not used, the entering Transaction always proceeds to the Next Sequential Block.

## Conditional Operators

The conditional operator may be E, G, GE, L, LE, MAX, MIN, or NE. If no conditional operator is used, E (equality) is assumed. When the condition is true, the Transaction being tested is selected. The conditions are defined as follows:

- ◆ E - The member Transaction attribute specified by Operand B must be equal to the reference value specified by Operand C for the member Transaction to be selected.
- ◆ G - The member Transaction attribute specified by Operand B must be greater than the reference value specified by Operand C for the member Transaction to be selected.
- ◆ GE - The member Transaction attribute specified by Operand B must be greater than or equal to the reference value specified by Operand C for the member Transaction to be selected.
- ◆ L - The member Transaction attribute specified by Operand B must be less than the reference value specified by Operand C for the member Transaction to be selected.
- ◆ LE - The member Transaction attribute specified by Operand B must be less than or equal to the reference value specified by Operand C for the member Transaction to be selected.
- ◆ MAX - The member Transaction attribute specified by Operand B must be equal to the greatest such attribute of all Transactions in the Group for the member Transaction to be selected.
- ◆ MIN - The member Transaction attribute specified by Operand B must be equal to the smallest such attribute of all Transactions in the Group for the member Transaction to be selected.
- ◆ NE - The member Transaction attribute specified by Operand B must be unequal to the reference value specified by Operand C for the member Transaction to be selected.

## Special Restrictions

- ◆ If Operand C is used, you must use Operand B.
- ◆ If MIN or MAX is used for the conditional operator, Operand C must not be used.

## Refuse Mode

A Transaction is never refused entry to a SCAN Block.

## Related Blocks

Transactions and numbers are added to Groups by JOIN Blocks. Transactions in Groups can be referenced by ALTER, EXAMINE, REMOVE, and SCAN Blocks.

## Related SNA

- ◆ *GTEntnum* - Transaction Group count. *GTEntnum* returns the membership count of Transaction Group *Entnum*.

## Related Windows

- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ Transaction Groups Snapshot - Picture of the state of the Transaction Groups in the simulation.

# SEEK

A SEEK Block sets the Current Line Position of a Data Stream.

### SEEK A,B

#### Operands

**A** - New Current Line Position. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**B** - Data Stream Entity. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

#### Example

##### SEEK 20,Data\_Base

In this example, the SEEK Block changes the Current Line Pointer of the Data Stream Data\_Base to 20.

#### Action

A Data Stream is a sequence of text lines used by a GPSS World simulation. Each Data Stream is identified by a unique number. Chapter 4 (4.16) contains a full discussion of Data Streams, including the error code descriptions under the Section entitled [Data Streams](#).

Each Data Stream has a *Current Line Position*. This is a 1-relative index to the next line position to be read or written. When a Transaction enters a SEEK Block, Operand A is evaluated numerically and used as next Current Line Position.

If Operand B is used, it is evaluated numerically and used as the Data Stream Entity number. It must be a positive integer. If Operand B is not specified, the SEEK is applied to Data Stream number 1.

If an error is detected, the error code is stored internally. A CLOSE Block can be used later to retrieve the Error Code.

#### Refuse Mode

A Transaction is never refused entry to a SEEK Block.

## Related Blocks

- ◆ OPEN - Create a Data Stream.
- ◆ CLOSE - Shut down a Data Stream.

◆ READ - Retrieve a text line from a Data Stream.

◆ WRITE - Send a text line to a Data Stream.

# SEIZE

**When the Active Transaction attempts to enter a SEIZE Block, it waits for or acquires ownership of a Facility Entity.**

## SEIZE A

### Operand

A - Facility name or number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

#### SEIZE Teller1

In this example, when a Transaction attempts to enter the SEIZE Block, the state of the Facility named Teller1 is tested. If it is idle, ownership is given to the Active Transaction, which is allowed to enter the SEIZE Block and proceed to the Next Sequential Block (NSB). If the Facility is busy (owned), the Active Transaction comes to rest on the Delay Chain of the Facility.

### Action

A SEIZE Block enables a Transaction to acquire ownership of a Facility. If the Facility is idle (not owned), the Transaction immediately acquires ownership of it, enters the SEIZE Block, and attempts to enter the next Block. If the Facility is already owned, the Transaction comes to rest, last within its priority, on the Delay Chain of the Facility and does not enter the SEIZE Block.

### Refuse Mode

A Transaction is refused entry to a SEIZE Block if it cannot immediately receive ownership of the Facility Entity.

The Active Transaction is refused entry to the SEIZE Block if the Facility entity is in the unavailable state.

When a Transaction is refused entry, its Delay Indicator is set and remains so until the Transaction enters a "Simultaneous" Mode TRANSFER Block. Simultaneous Mode TRANSFER Blocks are rarely used because a BOOLEAN VARIABLE can more efficiently control the coordination of the state of a number of resources when used in a TEST Block.

### Related SNAs

◆ *FEntnum* - Facility busy. If Facility *Entnum* is currently busy, *FEntnum* returns 1. Otherwise *FEntnum* returns 0.

◆ *FCEntnum* - Facility capture count. The number of times Facility *Entnum* has become owned by a Transaction.

◆ *FIEntnum* - Facility *Entnum* interrupted. If Facility *Entnum* is currently preempted, *FIEntnum* returns 1. Otherwise *FIEntnum* returns 0.

- ◆ *FREntnum* - Facility utilization. The fraction of time Facility *Entnum* has been busy. *FREntnum* is expressed in parts-per-thousand and therefore returns an real value 0-1000, inclusively.
- ◆ *FTEntnum* - Average Facility holding time. The average time Facility *Entnum* is owned by a capturing Transaction.
- ◆ *FVEntnum* - Facility in available state. FV *Entnum* returns 1 if Facility *Entnum* is in the available state, 0 otherwise.

## Related Window

- ◆ Facilities Window - Online view of Facility Entity dynamics.

# SELECT

**A SELECT Block chooses an entity and places its entity number in a Parameter of the Active Transaction.**

**SELECT O A,B,C,D,E,F**

## Operands

**O** - Conditional operator or logical operator. These choices are explained below. Required. The operator must be FNV, FV, I, LS, LR, NI, NU, SE, SF, SNE, SNF, SNV, SV, U, E, G, GE, L, LE, MIN, MAX, or NE.

**A** - Parameter name or number to receive the number of the selected entity. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Lower entity number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**C** - Upper entity number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**D** - Reference value for E Operand when in Conditional Mode. Optional. The operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*. Not used with MAX or MIN.

**E** - SNA class name. Entity attribute specifier for Conditional Mode tests. Required only for Conditional Mode. The type of SNA implies the entity type. You do not specify the entity number in Operand E. This is done automatically as the entity number range is searched. You may use any entity SNA class. Optional. The operand must be *Null* or *entitySNAclass*.

**F** - Alternate Block number. The destination Block if no entity is selected. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Examples

**SELECT SNF Not\_Full,Bin1,Bin3**

In this example, the entity number of the first Storage Entity that has room whose entity numbers fall between Bin1 and Bin3 will be stored in the Transaction Parameter named Not\_Full. If the Parameter does not exist, it will be created. It is always wise to test prior to entry into a SELECT Block that a successful selection is possible. If it is not possible, a 0 would be put in the Parameter and in this case, entry into an ENTER Block would cause an Error Stop since no entity can have an entity number of 0. You can also use an alternate exit if no entity is found to meet the desired criteria.

If the range of entities to be searched have been defined with alphanumeric names as above, you must first use EQU Statements to assign contiguous numbers to the range of names. The EQU Statements must occur prior to the original entity definitions.

```
10 Bin1 EQU 1
20 Bin2 EQU 2
30 Bin3 EQU 3
40 Bin1 STORAGE 11
50 Bin2 STORAGE 1000
60 Bin3 STORAGE 150
```

### **100 SELECT SNF 3,Bin1,Bin3,,,No\_Room**

Here, we have taken the example above and given an alternate destination to Transactions that find all Storage Entities to be full. If you do not test first for a successful selection using a TEST Block and a BOOLEAN VARIABLE, you should have an alternate destination should the selection be unsuccessful.

#### **SELECT E Empty1,Queue1,Queue9,0,Q**

In this example, the SELECT Block operates in Conditional Mode. Each Queue Entity with entity number between those of Queue1 and Queue9, inclusively, is tested. The first Queue Entity whose current content is 0 is selected. EMPTY1 is the name of the Parameter of the entering Transaction to receive the entity number of the first "empty" Queue Entity in the specified range.

### **Action**

When the SELECT Block is entered, the entity specified by Operand B is tested. If the entity does not exist and does not require a separate Command for its definition, a new entity is created. Thereafter, each entity in the range indicated by operands B and C is tested. An SNA is built automatically for each entity. The SNA class used to build the SNA is taken from Operand E or is specified by the logical operator.

A SELECT Block operates in either Logical Mode or in Conditional Mode, depending on whether a logical operator or a conditional operator is used.

When a logical operator is used (defined below), operands A, B, and C are used. The condition specified by the logical operator is tested for the entities whose numbers fall between B and C. The entity number of the first entity found in that condition is placed in the Parameter of the entering Transaction whose number is given by Operand A. The entity type is implied by the logical operator. If no entity is found, 0 is placed in the Parameter of the Active Transaction. If the Parameter does not exist, it is created.

When a conditional operator is used, operands A, B, C, E, and usually D are used. Operands A, B, C, are used to specify the target Parameter, and the range of entity numbers, as above. But now the conditional operator specifies the relationship between operands D and E that must hold for the entity to be selected. Both are evaluated numerically.

In Conditional Mode, the SNA class is combined with the entity specifier in order to build an SNA. The entity type implied by each SNA class is given in Section 3.4. The complete SNA is built from this class and the number of the entity being tested. Each such SNA is evaluated for each entity and compared to the reference value in Operand D. Operand D is evaluated with respect to the entering Transaction and is the reference value for comparison to Operand E, which specifies the class of SNA (and therefore the entity type) to be evaluated. The conditional operator specifies the relation that Operand E, evaluated at each entity, must bear to Operand D, evaluated on behalf of the entering Transaction, in order for the entity to be selected. If MIN or MAX is used as the conditional operator, Operand D is ignored.

In either mode, the F Operand may be used to direct the entering Transaction to a new Block in the event that no entity can be selected. If F is not used, the entering Transaction will always proceed to

the next sequential Block. If F is used, and no entity is selected, the Active Transaction proceeds to the Block specified by F, with its Parameter, specified by Operand A, unchanged.

## Logical Operators

Either a conditional operator or a logical operator is required. The logical operator may be FNV, FV, I, LS, LR, NI, NU, SE, SF, SNE, SNF, SNV, SV, or U. When the logical operator is true, the entity being tested is selected. The conditions are defined as follows:

- ◆ FNV - The Facility Entity must be unavailable in order to be selected.
- ◆ FV - The Facility Entity must be available in order to be selected.
- ◆ I - The Facility Entity must be currently interrupted (preempted) in order to be selected.
- ◆ LS - The Logicswitch Entity must be set in order to be selected.
- ◆ LR - The Logicswitch Entity must be reset in order to be selected.
- ◆ NI - The Facility Entity must NOT be currently interrupted (preempted) in order to be selected.
- ◆ NU - The Facility Entity must not be in use in order to be selected.
- ◆ SE - The Storage Entity must be empty in order to be selected.
- ◆ SF - The Storage Entity must be full in order to be selected.
- ◆ SNE - The Storage Entity must NOT be empty in order to be selected.
- ◆ SNF - The Storage Entity must NOT be full in order to be selected.
- ◆ SNV - The Storage Entity must NOT be available in order to be selected.
- ◆ SV - The Storage Entity must be available in order to be selected.
- ◆ U - The Facility Entity must be in use in order to be selected.

## Conditional Operators

Either a conditional operator or a logical operator is required. The conditional operator may be E, G, GE, L, LE, MAX, MIN, or NE. The conditions are defined as follows:

- ◆ E - The value of the automatic SNA must be equal to the reference value specified by Operand D for the entity to be selected.
- ◆ G - The value of the automatic SNA must be greater than the reference value specified by Operand D for the entity to be selected.
- ◆ GE - The value of the automatic SNA must be greater than or equal to the reference value specified by Operand D for the entity to be selected.
- ◆ L - The value of the automatic SNA must be less than the reference value specified by Operand D for the entity to be selected.
- ◆ LE - The value of the automatic SNA must be less than or equal to the reference value specified by Operand D for the entity to be selected.
- ◆ MAX - The value of the automatic SNA must be equal to the greatest of all such SNAs, for the entity to be selected.
- ◆ MIN - The value of the automatic SNA must be equal to the least of all such SNAs, for the entity to be selected.
- ◆ NE - The value of the automatic SNA must be unequal to the reference value specified by Operand E for the entity to be selected.

## Special Restrictions

- ◆ Either a conditional operator or a logical operator is required.
- ◆ Evaluated entity numbers must be positive integers.
- ◆ D and E are required if O is a conditional operator. Other than MIN or MAX.
- ◆ When evaluated, C must be greater than or equal to B.

## Refuse Mode

A Transaction is never refused entry to a SELECT Block.

## Related Windows

- ◆ Facilities Window - Online view of Facility Entity dynamics.
- ◆ Logicswitches Window - Online view of Logicswitch Entity dynamics.
- ◆ Storages Window - Online view of Storage Entity dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.

# SPLIT

**A SPLIT Block creates Transactions in the same Assembly Set as the Active Transaction.**

## SPLIT A,B,C

### Operands

**A** - Count. Number of related Transactions to be created. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Block number. Destination for new Transactions. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**C** - Parameter number. Parameter to receive serial number. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Examples

#### SPLIT 1

This is the simplest way to use the SPLIT Block. A new copy of the parent Transaction is created which will follow the parent Transaction to the next Block. The new Transaction has the same priority, Parameter values, and Mark Time as the parent.

#### SPLIT 3,P20,20

In this example, each time the SPLIT Block is entered, 3 new Transactions are created. Each Transaction has the same priority, Mark Time, and Parameter values as the parent Transaction, with the exception of Parameter number 20.

Each offspring Transaction will have a serial number in its Parameter number 20 based on the value in Parameter number 20 of the parent Transaction. The parent's Parameter value 20 is first

incremented by 1 (in case it is 0), and then the serial number of each offspring is calculated following that value sequentially.

The destination Block found in Operand B is evaluated with respect to each newly created Transaction. If Parameter 20 of the parent Transaction contains a Block number, say, n, then the first offspring will go to Block n+2, the second to n+3 and the third to n+4.

The parent Transaction will proceed to the Next Sequential Block following the SPLIT Block.

The above example is unique because Parameter 20 is used in both the B and C operands. Therefore, the serial number is used to determine the destination Block location.

The B and C operands can be used for separate purposes, B containing a single destination of the offspring Transactions and C, a serial number which might be used to direct the individual Transactions at some location further on in the simulation.

### SPLIT 3,Pro,17

In this example, each time the split Block is entered, 3 new Transactions are created. The parent Transaction goes to the Next Sequential Block, the offspring to a Block labeled Pro. Parameter 17 will receive the serialization. If the parent Transaction's Parameter is not predefined, it will be created and initialized to 0. In this example, the parent Transaction with a value of 0 in the Parameter 17, (the Parameter to be used for serialization) will, after passing through the SPLIT Block, have a 1 in Parameter 17 and the offspring will have 2, 3, and 4 in Parameter 17.

## Action

A SPLIT Block creates new Transactions which share the attributes of their parent. Each offspring Transaction has the same priority and Mark Time of the parent, and is in the same Assembly Set. If the Trace Indicator is set in the parent Transaction, it is turned on in the offspring Transaction.

The new Transactions may be sent to an alternate destination, by using a Block number for Operand B. The new Block number is evaluated for each new Transaction.

The optional C Operand specifies the Parameter number of the newly created Transactions to receive a serial number. The numbering starts at the value of the corresponding Parameter in the parent Transaction, plus 1. For example, if 3 copies are to be created with a serial number in Parameter 120, and the entering Transaction has 15 in its Parameter 120, then the new Transactions will have 16, 17, and 18, respectively, in their Parameters numbered 120. If the parent Transaction has no such Parameter, it is created and the numbering starts at 1.

By using both B and C operands it is possible to send each new Transaction to a different destination as shown in the second example above.

The parent Transaction and all the offspring Transactions all belong to the same set of Transactions called an Assembly Set. All Transactions belong to exactly one Assembly Set, which can be changed by entry into an ADOPT Block. Members of the same Assembly Set are said to be "related". The relationships of Transactions may be tested and used in ASSEMBLE, GATHER, MATCH, and GATE Blocks for synchronization and other purposes.

## Special Restriction

- ◆ A, B, and C, if specified, must be positive.

## Refuse Mode

A Transaction is never refused entry to a SPLIT Block.

## Related Blocks

- ◆ ADOPT - Set the Assembly Set of the Active Transaction.
- ◆ ASSEMBLE - Wait for and destroy Assembly Set members.
- ◆ GATHER - Wait for Assembly Set members.
- ◆ MATCH - Wait for Assembly Set member.

## Related SNAs

- ◆ A1 - Assembly Set. Return the Assembly Set of the Active Transaction.
- ◆ *MBEnignum* - Match at Block. *MBEnignum* returns a 1 if there is a Transaction at Block *Enignum* which is in the same Assembly Set as the Active Transaction. *MBEnignum* returns a 0 otherwise.

## Related Windows

- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# SUNAVAIL

**A SUNAVAIL Block ensures that a Storage Entity is in the unavailable state.**

## SUNAVAIL A

### Operand

**A** - Storage name or number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, *SNA\*Parameter*.

### Example

#### SUNAVAIL MotorPool

In this simple example, the Storage Entity named MotorPool is made unavailable when a Transaction enters this Block.

### Action

An SUNAVAIL Block ensures that a Storage Entity is unavailable. This means that all Transactions requesting storage will be placed on the Delay Chain of the Storage Entity. No Transaction is permitted to enter any ENTER Block if the Storage Entity is in the unavailable state.

If the Storage Entity is already in the unavailable state, the SUNAVAIL Block has no effect.

### Special Restriction

- ◆ Operand A must be the name or number of a predefined Storage Entity.

### Refuse Mode

A Transaction is never refused entry to a SUNAVAIL Block.

### Related Command

A Storage Entity must be defined in a STORAGE Command before it can be updated by an SUNAVAIL Block. The STORAGE Command must exist in the model, or must be sent to the

Simulation Object interactively, before a Transaction can enter the SUNAVAIL Block. Any attempt to do so before the Storage Entity is defined, cases an Error Stop.

A Storage Entity can be redefined by an interactive STORAGE Command.

## Related SNAs

- ◆ *REnnum* - Unused storage capacity. The storage content (or "token" spaces) available for use by entering Transactions at Storage Entity *Ennum*.
- ◆ *SEnnum* - Storage in use. *SEnnum* returns the amount of storage content (or "token" spaces) currently in use by entering Transactions at Storage Entity *Ennum*.
- ◆ *SAEnnum* - Average storage in use. *SAEnnum* returns the time weighted average of storage capacity (or "token" spaces) in use at Storage Entity *Ennum*.
- ◆ *SCEnnum* - Storage use count. Total number of storage units that have been entered in (or "token" spaces that have been used at) Storage Entity *Ennum*.
- ◆ *SEEnnum* - Storage empty. *SEEnnum* returns 1 if Storage Entity *Ennum* is completely unused, 0 otherwise.
- ◆ *SFentnum* - Storage full. *SFentnum* returns 1 if Storage Entity *Ennum* is completely used, 0 otherwise.
- ◆ *SREnnum* - Storage utilization. The fraction of total usage represented by the average storage in use at Storage Entity *Ennum*. *SREnnum* is expressed in parts-per-thousand and therefore returns an real value 0-1000, inclusively.
- ◆ *SMEnnum* - Maximum storage in use at Storage Entity *Ennum*. The "high water mark".
- ◆ *STEnnum* - Average holding time per unit at Storage Entity *Ennum*.
- ◆ *SVEnnum* - Storage in available state. *SVEnnum* returns 1 if Storage Entity *Ennum* is in the available state, 0 otherwise.

## Related Window

- ◆ Storages Window - Online view of Storage Entity dynamics.

# TABULATE

**A TABULATE Block triggers the collection of a data item in a Table Entity.**

## TABULATE A,B

### Operands

**A** - Table Entity name or number. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Weighting factor. Optional. The operand must be *Null*, *Name*, *Number*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

### Example

**TABULATE Sales**

When a Transaction enters this TABULATE Block, the Table Entity named Sales is found. Sales must have been defined in a TABLE Command. Then the statistics associated with the table are updated with no weighting.

## Action

When a Transaction enters a TABULATE Block, Operand A is evaluated and used to find a Table Entity. If there is no such entity an Error Stop occurs.

The Table Entity is then updated according to the operands in the TABLE Statement. If the B Operand is used, it is evaluated and used as a weighting factor. Otherwise the factor is taken to be 1.

A further discussion of the statistics gathered by table entities may be found in Section 4.10 and Chapter12.

## Special Restrictions

- ◆ A must be positive.
- ◆ B, if specified, must be positive.
- ◆ A must be the name or number of a predefined TABLE entity.

## Refuse Mode

A Transaction is never refused entry to a TABULATE Block.

## Related Command

A Table Entity must be defined in a TABLE Command before it can be updated by a TABULATE Block. The TABLE Command must exist in the model, or must be sent to the Simulation Object interactively, before a Transaction can enter the TABULATE Block. Any attempt to do so before the Table Entity is defined, cases an Error Stop.

A Table Entity can be redefined by an interactive TABLE Command.

## Related SNAs

- ◆ *TBEntnum* - Non weighted average of entries in Table Entity *Entnum*.
- ◆ *TCEntnum* - Count of non weighted table entries in Table Entity *Entnum*.
- ◆ *TDEntnum* - Standard deviation of non weighted table entries in Table Entity *Entnum*.

## Related Window

- ◆ Table Window - Online view of Table Entity dynamics.

# TERMINATE

**A TERMINATE Block removes the Active Transaction from the simulation and optionally reduces the Termination Count.**

## TERMINATE A

### Operand

**A** - Termination Count decrement. Default is 0. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Example

### TERMINATE 1

In this example, when a Transaction enters the TERMINATE Block it is removed from the simulation. Also, the Termination Count of the simulation, which is set by a START Command is decremented by 1.

## Action

When a Transaction enters a TERMINATE Block, Operand A is evaluated, truncated, and used to decrement the Termination Count of the simulation. If Operand A was not specified, the Termination Count is not changed.

The Active Transaction is then removed from the simulation, and a new Active Transaction is chosen.

The Termination Count of the simulation is set by a prior START Command. When the termination count reaches 0, the simulation ends, and unless suppressed by Operand B of the START Command, the optional standard report is written.

## Special Restriction

- ◆ A, if specified, must be positive.

## Refuse Mode

A Transaction is never refused entry to a TERMINATE Block.

## Related SNA

- ◆ TG1 - Termination Count of the simulation. This value is initialized by a START Command and indicates completion of the simulation when it becomes less than or equal to 0.

# TEST

**A TEST Block compares values, normally SNAs, and controls the destination of the Active Transaction based on the result of the comparison.**

### TEST O A,B,C

## Operands

**O** - Relational operator. Relationship of Operand A to Operand B for a successful test. Required. The operator must be E, G, GE, L, LE, or NE.

**A** - Test value. Required. The operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Reference value. Required. The operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**C** - Destination Block number. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Action

A TEST Block operates in either "Refuse Mode" or "Alternate Exit Mode". In either case, operands A and B are evaluated numerically, and compared.

If Operand C is not used, the TEST Block operates in Refuse Mode. When a Transaction attempts to enter a Refuse Mode TEST Block, and the test is unsuccessful, the Transaction is blocked, i.e. not allowed to enter the TEST Block, until the test is repeated and is successful. When the test is successful, the Active Transaction enters the TEST Block and then proceeds to the Next Sequential Block.

Blocked Transactions are placed on the Retry Chains of all entities involved in the comparison. When the state of any of these entities changes, the blocked Transaction is reactivated, the test specified by the TEST block is retried, and if successful, the Transaction is permitted to enter the TEST Block. However, the integration of User Variables does not cause blocked Transactions to be reactivated. You should use the thresholds in an INTEGRATE Command if you need to be notified about the level of one or more continuous variables. This is discussed further in Chapter 1, in the Section entitled Continuous Variables.

If Operand C is used, the TEST Block operates in Alternate Exit Mode. When a Transaction attempts to enter such a TEST Block, and the test is unsuccessful, the Transaction enters the TEST Block, is scheduled for the alternate destination Block specified by the C Operand, and is placed on the Current Events Chain in front of its priority peers. If the test is successful, the Active Transaction enters the TEST Block and then proceeds to the Next Sequential Block.

## Example

### TEST G C1,70000

In this example of a "Refuse Mode" TEST Block, the Active Transaction enters the TEST Block if the relative system clock value is greater than 70000. Otherwise, the Transaction is blocked until the test is true.

### TEST G Q\$Teller1\_Line,Q\$Teller2\_Line,Teller1

In this example of an "Alternate Exit Mode" TEST Block, the Active Transaction always enters the TEST Block. If the content of the queue entity named Teller1\_Line is greater than the content of the queue entity named Teller2\_Line the Transaction proceeds to the NSB. Otherwise, the Active Transaction is scheduled to enter the Block at the location named TELLER1.

## Relational Operator

The relational operator is required. It may be E, G, GE, L, LE, or NE.

The successful tests are defined as follows:

- ◆ E - The value of Operand A must be equal to the value of Operand B.
- ◆ G - The value of Operand A must be greater than the value of Operand B.
- ◆ GE - The value of Operand A must be greater than or equal to the value of Operand B.
- ◆ L - The value of Operand A must be less than the value of Operand B.
- ◆ LE - The value of Operand A must be less than or equal to the value of Operand B.
- ◆ NE - The value of Operand A must be unequal to the value of Operand B.

## Special Restrictions

- ◆ C must be the location of a Block in the simulation.
- ◆ TEST Blocks are extremely powerful, however, unsuccessful testing can cause large amounts of computer time to be used. You may need to arrange your simulation to

reduce the frequency of unsuccessful tests. This can be done by placing Transactions with no chance of a successful test on a User Chain using LINK and UNLINK Blocks.

## Refuse Mode

A TEST Block operating in Refuse Mode will refuse entry to a Transaction when the test fails. The refused Transaction will be blocked until the test is successful.

When a Transaction is refused entry, its Delay Indicator is set and remains so until the Transaction enters a "Simultaneous" Mode TRANSFER Block. Simultaneous Mode TRANSFER Blocks are rarely used because a BOOLEAN VARIABLE can more efficiently control the coordination of the state of a number of resources when used in a TEST Block.

## Related Windows

- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# TRACE

**A TRACE Block turns on the Trace Indicator of the Active Transaction.**

## TRACE

### Operands

None.

### Example

#### TRACE

In this example, the Trace Indicator of the Active Transaction will be set and stay on until an UNTRACE Block is entered.

### Action

When a Transaction enters a TRACE Block, its Trace Indicator is turned on. This causes a trace message to be sent to all Journal Windows every time the Transaction enters a new Block.

## Refuse Mode

A Transaction is never refused entry to a TRACE Block.

## Related Windows

- ◆ Journal Window - Record session events.
- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.

◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.

◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# TRANSFER

**A TRANSFER Block causes the Active Transaction to jump to a new Block location.**

**TRANSFER A,B,C,D**

## Operands

**A** - Transfer Block mode. Described below. Optional. The operand must be BOTH, ALL, PICK, FN, P, SBR, SIM, *fraction*, *Name*, *PosInteger*, *ParenthesizedExpression*, SNA, SNA\**Parameter*, or Null.

**B** - Block number or location. Parameter name or number when in P Mode. Optional. The operand must be Null, *Name*, *PosInteger*, *ParenthesizedExpression*, SNA, or SNA\**Parameter*.

**C** - Block number or location. Increment value in FN or P Mode. Optional. The operand must be Null, *Name*, *PosInteger*, *ParenthesizedExpression*, SNA, or SNA\**Parameter*.

**D** - Block number increment for ALL Mode. Default is 1. Optional. The operand must be Null, *Name*, *PosInteger*, *ParenthesizedExpression*, SNA, or SNA\**Parameter*.

## Action

A TRANSFER Block may operate in one of 9 "modes", each with different properties. When a Transaction enters a TRANSFER Block, Operand A is used to determine the mode of operation of the Block. The meaning of operands B and C depend on the mode. When you do not specify an operand which corresponds to a Block location, the next sequential Block after the TRANSFER Block is used.

## Unconditional Mode

When the A Operand is omitted, the TRANSFER Block operates in "Unconditional Mode". In Unconditional Mode, the Active Transaction always jumps to the location specified by the B Operand.

**TRANSFER ,New\_Place**

When a Transaction enters this TRANSFER Block, it is immediately scheduled for the Block at location New\_Place.

## Fractional Mode

When the A Operand is not a keyword, the TRANSFER Block operates in "Fractional Mode". In Fractional Mode, the Active Transaction jumps to the location specified by the C Operand with a probability given by Operand A. If Operand A is a nonnegative integer, it is interpreted as parts-per-thousand and converted to a fractional probability. The alternate destination is specified in Operand B, or the NSB if Operand B is omitted.

**TRANSFER .75,,New\_Place**

When a Transaction enters this TRANSFER Block, it proceeds to the location named NEW\_PLACE with a probability of .75. The remaining times it proceeds to the Next Sequential Block. You can select which random number generator number is to be used as the source of the random number. This is set in the "Random" page of the Model Settings Notebook.

**CHOOSE Edit / Settings**

then select the **Random** page. Then fill in the desired random number stream entity number in the entry box marked "TRANSFER". The installation default is to use random number stream number 1.

## Both Mode

When the A Operand is BOTH, the TRANSFER Block operates in "Both Mode". In Both Mode, the Block specified by Operand B is tested. If it refuses to admit the Active Transaction, the Block specified in Operand C is tested. The first Block to admit the Transaction will be the new destination. If neither Block admits the Transaction, it stays in the TRANSFER Block until it can enter one or the other.

### **TRANSFER BOTH,First\_Place,Second\_Place**

When a Transaction enters this TRANSFER Block, the Block at location First\_Place is tested. If the Transaction can enter, it does so. If not, the Block at location Second\_Place is tested. The Transaction enters if it can. Otherwise, it remains in the TRANSFER Block until it can leave.

## All Mode

When the A Operand is ALL, the TRANSFER Block operates in "All Mode". In "All" Mode, the Block specified by Operand B is tested. If this Block refuses to admit the Active Transaction, Blocks are tested in succession until the Block specified by Operand C is passed, unless one of the Blocks tested admits the Transaction prior to reaching the Block specified in Operand C. The location of each succeeding test Block is calculated by adding Operand D to the previous test Block location. If Operand D is not used, every Block between those specified by B and C, inclusive, are tested. If Operand C is not used, only one Block is tested. No Block with a location higher than Operand C is tested. The first Block to admit the Transaction will be the new destination. If no Block admits the Transaction, it stays in the TRANSFER Block until it can enter one.

### **TRANSFER ALL,First\_Place,Last\_Place,2**

When a Transaction enters this TRANSFER Block, the Block at location First\_Place is tested. If the Transaction can enter, it does so. If not, the Blocks at every second higher location are tested. The Transaction enters if it can. If all tested Blocks refuse, the testing ends with the Block at location Last\_Place, or with the Block just before it, depending on the separation of First\_Place and Last\_Place. If no Block accepts, the Transaction remains in the TRANSFER Block until it can leave.

## Pick Mode

When the A Operand is PICK, the TRANSFER Block operates in "Pick" Mode. In Pick Mode, a destination is chosen randomly.

### **TRANSFER PICK,First\_Place,Last\_Place**

When a Transaction enters this TRANSFER Block, a location is chosen randomly which is numerically between First\_Place and Last\_Place, inclusively. The chosen location is the next destination for the Active Transaction. You can select which random number generator number is to be used as the source of the random number. This is set in the "Random" page of the Model Settings Notebook.

### **CHOOSE Edit / Settings**

Then select the **Random** page. Then fill in the desired random number stream entity number in the entry box marked "TRANSFER". The installation default is to use random number stream number 1.

## Function Mode

When the A Operand is FN, the TRANSFER Block operates in "Function Mode". In Function Mode, the destination is chosen by evaluating a function entity, specified in B, and adding an optional increment specified in C.

### **TRANSFER FN,Func1,5**

When a Transaction enters this TRANSFER Block, the function entity named FUNC1 is evaluated, and added to 5, to determine the location of the destination.

## Parameter Mode

When the A Operand is P, the TRANSFER Block operates in "Parameter Mode". In Parameter Mode, the Active Transaction jumps to a location calculated from the sum of a Parameter value and Operand C. If C is not specified, the Parameter value is the location of the new destination.

**TRANSFER P,Placemark,1**

When a Transaction enters this TRANSFER Block, it is immediately scheduled for the Block immediately after the location specified in the Transaction Parameter named Placemark.

## Subroutine Mode

When the A Operand is SBR, the TRANSFER Block operates in "Subroutine Mode". In Subroutine Mode, the Active Transaction always jumps to the location specified by the B Operand. The location of the transfer Block is placed in the Parameter specified in Operand C.

**TRANSFER SBR,New\_Place,Placemark**

When a Transaction enters this TRANSFER Block, it is immediately scheduled for the Block at location New\_Place. The location of the TRANSFER Block is placed in the Parameter named Placemark. If there is no such Parameter, it is created.

To return from the subroutine, use a TRANSFER Block in Parameter Mode as shown above.

## Simultaneous Mode

When the A Operand is SIM, the TRANSFER Block operates in "Simultaneous Mode". In Simultaneous Mode, the Active Transaction jumps to one of two locations depending on the Delay Indicator of the Transaction. If the Delay Indicator is set, the Transaction jumps to the location specified by the C Operand and the Delay Indicator is reset (turned off). If the Delay Indicator is reset (off), the Transaction jumps to the location specified by the B Operand.

The Delay Indicator of a Transaction is set when the Transaction is refused by a Block. The Delay Indicator remains set until the Transaction enters a Simultaneous Mode TRANSFER Block.

**TRANSFER SIM,Nodelay\_Place,Delay\_Place**

When a Transaction enters this TRANSFER Block, it is immediately scheduled for the Block at location DELAY\_PLACE if its Delay Indicator is set, or NODELAY\_PLACE if it is reset. After the transfer, the Delay Indicator is always reset.

TRANSFER SIM is rarely used. It is much more efficient to us a BOOLEAN VARIABLE in a Refuse Mode TEST Block when you wish to coordinate the state of a number of entities. TRANSFER SIM was originally developed before BOOLEAN VARIABLES had been added to the GPSS language.

## Special Restrictions

- ◆ In All Mode, Operand C must be greater than Operand B., and if D is used (C-B) must be an even multiple of D.
- ◆ All calculated Transaction destinations must be valid Block locations.
- ◆ In Both Mode or All Mode it is possible to waste a lot of computer time on unsuccessful testing. You may want to place Transactions on a User Chain until the test is likely to be successful. This can be done by using LINK and UNLINK Blocks.

## Refuse Mode

A Transaction is never refused entry by a TRANSFER Block. If a Transaction becomes blocked by refusal of destination Blocks, it remains in the TRANSFER Block.

## Related Windows

- ◆ Blocks Window - Online view of Block dynamics.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# UNLINK

**An UNLINK Block removes Transactions from the User Chain of a Userchain Entity.**

**UNLINK O A,B,C,D,E,F**

## Operands

**O** - Relational operator. Relationship of D to E for removal to occur. These choices are explained below. Optional. The operator must be *Null*, E, G, GE, L, LE or NE.

**A** - User Chain number. User Chain from which one or more Transactions will be removed. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**B** - Block number. The destination Block for removed Transactions. Required. The operand must be *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**C** - Removal limit. The maximum number of Transactions to be removed. If not specified, ALL is used. Optional. The operand must be ALL, *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**D** - Test value. The member Transaction Parameter name or number to be tested, a Boolean variable to be tested, or BACK to remove from the tail of the chain. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, *SNA\*Parameter* or BACK.

**E** - Reference value. The value against which the D Operand is compared. Optional. The operand must be *Null*, *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*. Operand E is not used if Operand D is a Boolean Variable.

**F** - Block number. The alternate destination for the entering Transaction. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

## Example

**UNLINK OnHold,Reentry,1**

This is the simplest way to use the UNLINK Block. The first Transaction at the head of the Userchain Entity named OnHold, if any, is taken off the chain and is directed to the Block labeled Reentry. It is put on the CEC behind Transactions of the same priority. The Transaction entering the UNLINK Block proceeds to the Next Sequential Block.

## Action

An UNLINK Block removes Transactions from a User Chain and directs them to a new Block. Transactions can be selected for removal, and a limit can be imposed on the number of Transactions removed. If there are no Transactions on the chain when the UNLINK Block is entered, the Link Indicator of the User Chain is reset. Also, the Transaction entering the UNLINK Block may be redirected according to the optional F Operand.

You may limit the number of Transactions to be removed from the User Chain by specifying the C Operand. If Operand C is omitted, ALL is assumed.

You may test each Transaction for removal by using the relational operator and/or operands D and E, both of which are evaluated numerically.

Operands D and E and the conditional operator are optional. When they are omitted, all Transactions are removed from the front of the chain, until the chain is exhausted or the limit count (Operand C) is reached.

You can use one of 3 options for Operand D. Operand D can be a Boolean variable, a Parameter number, or the word BACK. If Operand D is a Boolean variable, it is evaluated with respect to the chained Transaction, and if the result is nonzero, the chained Transaction is removed. If Operand D is BACK, Transactions are removed from the rear of the User Chain until the limit count is reached. Otherwise, the operand is evaluated with respect to the User Chain member Transaction and used as a Parameter number, whose value is returned from the User Chain member as the final result. This final value is compared to the result of evaluating Operand E.

If D specifies a Parameter and E is not used, a Parameter of the User Chain Transaction is compared to the same Parameter of the Active Transaction. If they are equal, the chained Transaction is removed from the chain.

Operand E is used if and only if the relational operator is used. In this case, Operand D is required as well. The User Chain is scanned from the front. Each Transaction, up to the limit count (Operand C), is removed if Operand D bears the same relationship to Operand E as is specified by the relational operator. If Operand E is a Transaction related SNA, it is evaluated with respect to the Active Transaction.

You may use the relational operator to specify the relationship between the Transaction attribute (Operand D) and the reference value (Operand E) which will initiate a removal of the Transaction. The default for the relational operator is E for equality. If you use no relational operator, but you use Operand D and Operand E, the values must be equal for the Transaction to be removed from the chain.

The F Operand is used to specify an alternate destination to be taken by the entering Transaction when the limit count (Operand C) cannot be reached, or when no Transactions can be removed from the User Chain. If the F Operand is not used, the entering Transaction always goes to the Next Sequential Block.

Userchain Entities have a self-contained gate called a Link Indicator. When it is off (reset), LINK Blocks which have a C Operand will not place an entering Transaction on the User Chain. The "gate" to the User Chain is "closed" when the Link Indicator is off (reset).

The Link Indicator is manipulated by both LINK and UNLINK Blocks. It is turned off when an UNLINK Block finds the User Chain empty. This condition may represent a case where the next Transaction to arrive should not wait (on the User Chain).

The Link Indicator represents the busy condition of a hypothetical resource. You can use LINK and UNLINK Blocks to handle the queuing on such a resource. A further discussion of the Link Indicator may be found in the description of LINK Blocks in this chapter.

## Relational Operators

The relational operator may be E, G, GE, L, LE, or NE. If no relational operator is used, E (equality) is assumed. When the relationship is true and the limit condition has not been reached, the Transaction being tested is removed from the User Chain. The relationships are defined as follows:

- ◆ E - The Transaction attribute specified by Operand D must be equal to the reference value specified by Operand E for the Transaction to be removed from the chain.
- ◆ G - The Transaction attribute specified by Operand D must be greater than the reference value specified by Operand E for the Transaction to be removed from the chain.
- ◆ GE - The Transaction attribute specified by Operand D must be greater than or equal to the reference value specified by Operand E for the Transaction to be removed from the chain.
- ◆ L - The Transaction attribute specified by Operand D must be less than the reference value specified by Operand E for the Transaction to be removed from the chain.
- ◆ LE - The Transaction attribute specified by Operand D must be less than or equal to the reference value specified by Operand E for the Transaction to be removed from the chain.

- ◆ NE - The Transaction attribute specified by Operand D must be unequal to the reference value specified by Operand E for the Transaction to be removed from the chain.

## Special Restrictions

- ◆ A, B, C, and F, if specified, must be positive.
- ◆ B and F, if specified, must be Block locations in the simulation.
- ◆ If Operand D is used but is neither a BV class SNA or BACK, then you must use Operand E.
- ◆ If Operand D is BACK or a Boolean variable, then you must use neither Operand E or a relational operator.
- ◆ If you use a relational operator, you must use operands D and E. D must be used, but must not be a Boolean variable.
- ◆ If you use Operand E you must use Operand D.
- ◆ Operand D cannot be the literal constant 0.

## Refuse Mode

A Transaction is never refused entry to an UNLINK Block.

## Related SNAs

- ◆ CAEnnum - Average Userchain content. The time weighted average number of chained Transactions for Userchain *Ennum*.
- ◆ CCEnnum - Total Userchain entries. The count of all Transactions chained to Userchain *Ennum*.
- ◆ CHEEnnum - Current Userchain content. The current number of Transactions chained to Userchain *Ennum*.
- ◆ CMEnnum - Maximum Userchain content. The maximum number of Transactions chained to Userchain *Ennum*. The "high water mark".
- ◆ CTEnnum - Average Userchain residence time. The average duration of Transactions at Userchain *Ennum*.

## Related Window

- ◆ Userchains Snapshot - Picture of the state of the Userchains in the simulation.

# UNTRACE

**An UNTRACE Block turns off the Trace Indicator of the Active Transaction.**

## UNTRACE

### Operands

None

## Example

### UNTRACE

In this example, all Transactions passing through an UNTRACE Block will have their Trace Indicators unset. These Transactions will no longer produce trace messages as they move from Block to Block.

## Action

When a Transaction enters a UNTRACE Block its Trace Indicator is turned off. Thereafter, no Block entry traces will be recorded on behalf of the Transaction unless it enters a TRACE Block.

Trace messages are sent to all open Journal Windows.

## Refuse Mode

A Transaction is never refused entry to an UNTRACE Block.

## Related Windows

- ◆ Journal Window - Record session events.
- ◆ Transaction Snapshot - Picture of the state of a Transaction in the simulation.
- ◆ CEC Snapshot - Picture of the state of the Current Events Chain in the simulation.
- ◆ FEC Snapshot - Picture of the state of the Future Events Chain in the simulation.

# WRITE

**A WRITE Block passes a text line to a Data Stream.**

**WRITE A,B,C,D**

## Operands

**A** - Text Line. Required. Evaluated as a string. The operand must be *Name*, *Number*, *String*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*.

**B** - Data Stream number. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA* or *SNA\*Parameter*. Default is 1.

**C** - Alternate Destination Block name or number. Optional. The operand must be *Null*, *Name*, *PosInteger*, *ParenthesizedExpression*, *SNA*, or *SNA\*Parameter*.

**D** - Insert Mode. Optional. The operand must be *Null*, ON or OFF. The default is ON.

## Example

**WRITE "New Line 20",1,Done**

In this example, the WRITE Block send a text line to Data Stream number 1. If an error occurs, the Active Transaction proceeds to the Block labeled Done. Otherwise it moves to the Next Sequential Block. In this case, if the Data Stream is an I/O Stream or an In-Memory Stream, the WRITE is processed in Insert Mode.

## Action

When a Transaction enters a WRITE Block, Operand A is evaluated as a string. Numeric values are converted to an ASCII string equivalent. Then the identity of the Data Stream is determined from Operand B.

If Operand B is used, it is evaluated numerically, truncated, and used as the Data Stream Entity number. This must be a positive integer. If Operand B is not used, Data Stream number 1 is assumed.

If Operand C is used, any error occurring during the WRITE causes the Active Transaction to proceed to the Block with that number.

Operand D sets the write mode, as discussed below.

In any case, if an error is detected, the error code is stored internally. A CLOSE Block can be used to retrieve the error. Chapter 4 (4.16) contains a full discussion of Data Streams, including the error code descriptions.

## Write Modes

A WRITE to a Data Stream is operated in either *Insert Mode* or *Replace Mode*. The Current Line Position is used slightly differently in these two modes. The write mode is set by Operand D. If it is not used, or is ON, the WRITE is processed in Insert Mode. If it is OFF, the WRITE is processed in Replace Mode.

### Insert Mode

This is the default mode for WRITE Blocks.

Action:

1. Move all text lines at, or after, the Current Line Position down one position.
2. If the Current Line Position is after the last text line, set it to just after the last text line in the Data Stream.
3. Place a copy of the new text line at the Current Line Position.
4. Increment the Current Line Position.

### Replace Mode

Action:

1. If the Current Line Position is after the last text line, fill any intervening line positions with null text lines.
2. Delete any text line at the Current Line Position.
3. Place a copy of the new text line at the Current Line Position.
4. Increment the Current Line Position.

### Further Discussion

Chapter 4 (4.16) contains a full discussion of Data Streams under the Section entitled, Data Streams.

## Refuse Mode

A Transaction is never refused entry to a WRITE Block.

## Related Blocks

- ◆ OPEN - Create a Data Stream.
- ◆ CLOSE - Shut down a Data Stream.
- ◆ READ - Retrieve a text line from a Data Stream.
- ◆ SEEK - Set the Current Line Position of a Data Stream.

[\[Table of Contents\]](#)

# **Chapter 8 - PLUS: The Programming Language Under Simulation**

A GPSS World Model is a sequence of Model Statements. A Model Statement may be either a GPSS Statement or a PLUS Procedure definition.

PLUS Expressions can exist as operands in GPSS Statements as well as within PLUS Procedures. Expressions can contain Procedure Calls which invoke either built-in or user defined PLUS Procedures. In addition, all the Stream Input/Output operations are available as built-in library procedures. This means that you can read from and write to files using GPSS Blocks or by calling the corresponding library procedures, or by using a mixture of both. Not only that, but now you can invoke external programs that exist on your system in EXE or DLL files by using the new Dynamic Call Procedures.

This chapter contains reference information on the PLUS language, and on the built-in procedure libraries you can access through it.

## **How this Chapter Describes Syntax**

The valid form of individual PLUS Statements and Procedure invocations is indicated in a syntax line at the beginning of the statement description. Completing a PLUS Statement is similar to filling in the blanks of a form. The syntax line describes the constant and variable parts of the statement. It is up to you to "instantiate" the variable part of the statement in order to make it do what you want it to.

The items in the syntax line tell you how to code the statement. The following rules are observed:

1. Keywords are shown in the syntax line as all caps bold. These words, and unitalicized parentheses, must be entered as given. However, when you code the statement, capitalization of keywords is optional. For example,

***ReturnString = LEFT( SourceString, MaxCount )***

contains the Procedure Name "Left" in all caps. Although you can change the case of the letters any way you like, you must still include the word "Left" and the parentheses when you invoke the Procedure. When used as a PLUS Assignment Statement, a semicolon terminator must be added.

2. The data type of the result of a Procedure invocation is given in the syntax line to the left of an equal (=) sign. Procedures can be invoked from a PLUS Procedure Call Statement, in which case the unassigned result is discarded. In a PLUS Assignment Statement, an "lvalue" (named value or matrix element) is required to the left of the equal sign. It receives the result of the Procedure invocation.
3. Arguments are given in the syntax line as capitalized bold italic semantic variables. They indicate that you must make a selection from a class of possibilities. The definition of each argument follows the grammar line. That's where the arguments are defined, and the possible choices for instantiation by you are given in one or more non-bold italic syntactic variables, such as in *Must be ParenthesizedExpression*. That means that you are to use a valid PLUS Expression enclosed in parentheses when you code the statement. Syntactic variables are defined formally in the Appendix.

The valid forms you are to use in the instantiation of syntactic variables is described in the next few sections. The statement and Procedure descriptions follow that.

## **8.1. Defining PLUS Procedures**

Procedures are commonly used in two different ways. They can be used for their effects on global variables and named values or to return a value to an expression.

If you simply wish to update a global variable or change a named value, no Return Statement is needed. In all cases where the result of a PLUS Procedure will be used where the Procedure was called (e.g.. in an operand or another PLUS expression), a Return Statement is necessary.

It's easy to define a PLUS Procedure. All you have to do is to place a valid PROCEDURE Statement in a Model File and Translate it with the model, or send a PROCEDURE Statement to an

existing simulation. Thereafter, you can invoke your Procedure in an Expression evaluation or a PLUS Assignment Statement just like any other Library Procedure.

As a simple example, consider the following PROCEDURE Statement: **PROCEDURE SetPop(Pop\_Level) Foxes = Pop\_Level ;**

Although most Procedures are more complex, this is all that is needed to define one. It doesn't even declare a return value, so a value of 0 would be used by default. The sole purpose of this Procedure is to use the value, Pop\_Level, that is passed to the Procedure and set the named value, Foxes, equal to that value. This, or any other PROCEDURE Statement, could even be sent to an existing simulation to define or redefine a Procedure named "SetPop". Then, any PLUS Expression in the simulation could include a Procedure call such as

**SetPop(Rabbits/10);**

to assign a value to the user variable **Foxes**. Of course, here we assume that the User Variable RABBITS has already been given a value.

You can define temporary Matrix Entities and temporary User Variables that exist only throughout an invocation of a Procedure. This is done using the TEMPORARY Statement in the Procedure Definition.

When a PLUS Procedure is invoked by a Procedure Call, one statement in the invoked Procedure is "performed" or "executed" after another. Most Procedures are defined as an outer Compound Statement containing a Statement List. Normal execution of a Statement List is to perform each Statement in succession. Compound Statements, IF Statements, IF-ELSE Statements, WHILE Statements and GOTO Statements may alter the normal sequence of execution. If a RETURN Statement is executed, the Procedure invocation is terminated and any memory used by it is released.

You do not really need a RETURN Statement in a PLUS Procedure. If, during an invocation of your Procedure, processing ever reaches the end of the Procedure's Statement List, the invocation of the Procedure is terminated and a value of 0 is returned. Something is always used as a return value, in case one is needed in an Expression evaluation. If you do not specify an Expression in your RETURN Statement, or if your last statement in the Statement List is completed, a zero result is used.

If you have PLUS Procedures you use in more than one model, you can keep them in a source file, called a User Procedure Library. You can then use the INCLUDE Command to bring in your Library into each model that needs it.

The remainder of this chapter discusses all the features of the PLUS language, and all the built-in Procedures available for your use within a simulation.

## 8.2. The Language

You can incorporate either PLUS Expressions or PLUS Procedures and Experiments, built using the PLUS Language, into your GPSS World models. To do so, you must be familiar with both the building blocks of the language, and how to put them together. These are the topics of the remainder of this chapter.

### 8.2.1. The Character Set

The GPSS World character set consists of naming characters and special characters. Naming characters include the uppercase letters **A-Z**, the Lower case letters **a-z**, the digits **0-9**, and the underscore character ( **\_** ).

The special characters are used to denote operators and punctuation. They are: **# \* & + - / \ , ;**

The **^** character is also considered as an operator.

Normally, the **#** character is used as the multiplication operator and the **\*** character as the GPSS SNA indirect addressing operator. If you prefer, you can reverse these by selecting "Switch \* and # " in page 1 of the Settings Notebook. Use Edit / Settings on the Main Menu to do this.

We recommend that you use a consistent style when you use upper or lower case letters. Your model will be much more readable if you do. For example, you could make your User Variables stand out as all caps, or you could make keywords stand out. On the other hand, capitalized words (first letter upper case, the rest lower) tend to be easiest to read.

### 8.2.2. Names

Names are character sequences created by you to identify entities, variables, and program locations. The naming characters are letters, digits, and underscore.

There are a few rules you must follow when you create a name. You must use from 1 to 250 naming characters, and you must start the name with a letter. In addition, your name must not be the same as a GPSS World keyword, System Numeric Attribute, or SNA Class. The keywords are listed in the Appendix.

GPSS World is case insensitive. The upper/lower case distinction does not matter. Only the characters in string constants and comments retain lower case. All other lower case letters are converted to upper case internally. This removes the danger of spelling two variables the same but having them refer to distinct values.

Your primary job when creating a name is to avoid keywords. You can refer to the list of keywords in the Appendix, if you like, but there's an easier way. All you have to do is to include an underscore somewhere in each name, after the first character, which must be a letter. That will guarantee that you will not clash with GPSS World keywords.

### Named Values

Named Values are names that you have placed in a Label field or PLUS Assignment Statement. You can use them to identify an entity or to hold a value. If you use them to label a GPSS Statement defining an entity, they are called Entity Labels. If you create them by assigning a value, as in an EQU Command or PLUS Assignment Statement, they are called User Variables.

Named Values normally have global scope. You can refer to them anywhere in the model. There are two kinds of Named Values, Entity Labels and User Variables.

### GPSS Entity Labels

Entity Labels are names you use in an entity creation command. Unlike User Variables, Entity Labels are automatically given a system assigned number, normally a unique integer greater than 9,999. When you refer to a labeled entity, GPSS World first retrieves the Entity Number stored as the value of the Entity Label. The Entity Number is a strictly positive integer, that is used by the Simulation Object to find or create any GPSS entity.

Except for Block Labels, you can assign your own value to an Entity Label. Generally, you should do that only BEFORE you use the name to define any entity. Why? Because the Entity Number is set at the time of creation of the entity. If you then change the value of the label to something else, you will not be able to address that entity. In other words, if you want to use an EQU Command to assign your own entity number, put it early in the model, before the entity definition statement.

### User Variables

A User Variable is a Named Value not used as an Entity Label. You can give it a numeric or string value, and you can integrate it as a continuous variable.

User Variables can be global, usable throughout the model, or local, usable only within a single PLUS Procedure. The latter are declared in a Temporary Declaration in the PLUS Procedure in which it is defined. All other User Variables are global.

You can assign a value to a User Variable through EQU Commands, through PLUS Assignment Statements, or through numerical integration, setup by an INTEGRATE Command.

User Variables must be initialized before they can be used. You must assign values to them before you can use them in Expressions or integrations.

You can observe the values of User Variables in online Plot Windows and Expressions Windows, which have been opened onto your simulation.

### PLUS Statement Labels

You can use names to identify specific statements in a PLUS Procedure. To do so, simply begin the statement with a name followed by a colon. Such statements can then be targets of GOTO Statements. Both Labeled Statements and GOTO Statements are discussed below, in the section on PLUS Statements.

Statement labels have local scope. They do not clash with names defined outside the Procedure definition. This means that they neither refer to global objects, nor can they be referenced from outside the Procedure.

## Procedure Names

You must name a PLUS Procedure or Experiment when you define it. Thereafter, you can invoke the same Procedure by a Procedure Call using the same name.

Procedure names are global in scope. Any Procedure can be invoked from any statement in the model. If you define a Procedure with a given name, any existing user defined PLUS Procedure with that name is redefined.

Experiments are special kinds of Procedures. They are identical to Procedures in syntax except that the keyword EXPERIMENT replaces PROCEDURE. Experiments can only be invoked by a CONDUCT Command.

## Matrix Labels

Similar to User Variables, Matrix Entities can have local or global scope. A Matrix defined in a Temporary Matrix PLUS Declaration is local, known only with the Procedure in which it is declared. Such a Matrix is created when a Procedure invocation begins, and destroyed when it ends.

Global Matrix Entities are declared in a GPSS MATRIX Command. The Label you use becomes the Matrix Entity Label. Global Matrix Entities are permanent and may be referenced anywhere in the model.

All Matrices are created with uninitialized elements. You must assign values to them before you can use them in Expressions.

You can use a Matrix Window to view the dynamics of any cross section of a Matrix. In addition, you can observe the values of Matrix elements in online Plot Windows and Expressions Windows, which have been opened onto your simulation.

### 8.2.3. Expressions

A PLUS Expression is a combination of one or more elements, called factors. Expressions are built by using operators and Procedure calls to combine factors. The rules for building Expressions are described in the next few sections.

Expressions can be used in PLUS Procedures and in the operands of GPSS Statements. Usually, when an Expression is used in a Block operand, it must be enclosed in parentheses. The list of acceptable syntactic variables will then include *ParenthesizedExpression* as one or the operand's alternative forms. For compatibility some Commands do not need to use the outer parentheses, but if you always parenthesize Expressions used in GPSS Statements, you will be safe.

## Data Types

Any User Variable, Matrix element, Savevalue, or Transaction Parameter can have a value on any of several forms called Data Types. Further, each has an uninitialized form which prevents its use before it has been assigned its first value.

The three major Data Types are Integer, Real, and String. The first two may be referred to as numeric Data Types.

Integers are 32 bit two's complement numbers. If, during arithmetic operations, an integer overflows, it is converted to a real number.

Real data items are double precision floating point numbers. They have a precision of 15 decimal digits and a range of exponents of -306 to 306.

Strings are sequences of ASCII characters. They can be any size, up to the maximum memory request declared in the "Simulate" page of the Settings Notebook. A whole class of String Procedures resides in the Procedure Library for creating and manipulating strings.

Data types are converted from one to the other explicitly by Procedure calls, or implicitly during the evaluation of Expressions. This is discussed further below.

## Factors

Factors are the basic building blocks of Expressions. You combine them in Expressions which, in turn, can be used in GPSS Statement operands and PLUS Procedures. The detailed grammar is in the Appendix, but the following definitions should be suggestive.

The GPSS World Expression factors are:

1. String Constants, such as "**A stitch in time ...**".
2. Real constants, such as **201.6**.
3. Integer Constants, such as **17**.
4. Names, such as **Water\_Level**.
5. PLUS Matrix elements, such as **Array1[ P\$Part, X\$Order\_Index+20 ]**. Matrix elements must be *Name [ ExpressionList ]*. The expression List can contain from 1 to 6 expressions, corresponding to the dimensions of the matrix. Each can vary in complexity from a simple integer to a highly complex PLUS expression.
6. Procedure Calls, such as **Word(X\$Quote,2)** or **Myproc(X\$Arg1,X\$Arg2)**. Procedure calls must be *Name(ExpressionList )*.
7. System Numeric Attributes, such as **AC1**, **F\$My\_Facility**, **MX\$Mat1(2,1)**, and **SR\*My\_Parm**.

You use operators and Procedure calls to combine factors into Expressions, using the rules associated with each operator or Procedure. These are discussed below.

Factors are defined formally in the Appendix.

## Operators

The arithmetic operators of GPSS World Expressions are listed here in decreasing order of precedence. Note that the multiplication operator is normally **#**, not **\*** (which is the indirect addressing operator within System Numeric Attributes). You can switch the roles of **#** and **\*** so that **\*** denotes multiplication and **#** denotes GPSS indirect addressing. This is done by clicking a checkbox labeled "**Switch \* and #**" in the first page of the Object's Settings Notebook (Edit / Settings).

All arithmetic operators coerce string operands to numeric values.

Operator	Action	Result	Arity	Assoc
-	Negation	The Additive Inverse	Unary	Right
^	Exponentiation	The Arithmetic Power	Binary	Right
'NOT'~	Invert	1 (TRUE) or 0 (FALSE)	Unary	Right
'AND'	Logical AND	1 (TRUE) or 0 (FALSE)	Binary	Left
'OR'	Logical OR	1 (TRUE) or 0 (FALSE)	Binary	Left
'G' >	Greater Than	1 (TRUE) or 0 (FALSE)	Binary	Left
'L' <	Less Than	1 (TRUE) or 0 (FALSE)	Binary	Left
'E'	= Equal To	1 (TRUE) or 0 (FALSE)	Binary	Left
'NE' /=	Not Equal	1 (TRUE) or 0 (FALSE)	Binary	Left
'LE' <=	Less Than or Equal	1 (TRUE) or 0 (FALSE)	Binary	Left
'GE' >=	Greater Than or Equal	1 (TRUE) or 0 (FALSE)	Binary	Left
# (or *)	Multiplication	The Arithmetic Product	Binary	Left
/	Division	The Arithmetic Quotient	Binary	Left
\	Integer Division	The Integer Quotient	Binary	Left
@	Modulo Division	The Integer Remainder	Binary	Left
+	Addition	The Arithmetic Sum	Binary	Left
-	Subtraction	The Arithmetic Difference	Binary	Left

The arity of an operator indicates the number of operands it requires. Unary operators always appear to the left of the operand.

Operators are either left or right associative. This is given in column 5, above. An operand surrounded by left associative operators of equal precedence is taken by the operator to its left.

Expressions are combinations of one or more factors, connected by operators, and evaluated according to a well-defined set of rules.

## Evaluation

When an Expression is evaluated, values are determined and combined in order to calculate a final result. The following actions occur:

1. String and numeric constants evaluate to a copy of themselves.
2. Named Values evaluate to the associated value.
3. SNAs evaluate to a simulation or entity state value.
4. Procedure calls evaluate each argument, then return the result of the Procedure invocation. Built-in Procedures coerce the arguments.
5. Operators evaluate one or two operands, coerce the intermediate results to numeric values, and calculate the resulting value of the operation.

The Operators in GPSS World have a precedence order that affects how the Expressions you create are evaluated. For example, the exponentiation operator **^** has higher priority than the multiplication operator **#**. This means that the Expression

**4 # 3 ^ 2**

is evaluated to a result of 36, not 144, because the first intermediate result is formed by evaluating the operator with the higher precedence. If you want a different order of evaluation than that implied by the operator precedence hierarchy, you should use parentheses to control the evaluation. Continuing the last example,

**(4 # 3) ^ 2**

evaluates to 144.

GPSS World operators are "overloaded" in the sense that their operands are coerced to the proper data type just before the operation is applied. Therefore, you do not need to worry about data types when creating PLUS Expressions.

Truncations are not done automatically during Expression evaluation. To do so, you must either invoke Procedures that truncate, or you must run in GPSS/PC Compatibility Mode. In Compatibility Mode, the old GPSS rules are used to truncate SNAs and the intermediate results of VARIABLE and BVARIABLE evaluations. Compatibility Mode is discussed in Chapter 1.

## Evaluation of Expressions

Expressions are combinations of mathematical operators, library functions, SNAs, and constants that obey the rules of elementary algebra. An Expression is evaluated according to the hierarchy of operators listed above, and otherwise left to right. The order of evaluation may be controlled by inserting paired parentheses.

Expressions may be *evaluated*, *evaluated numerically*, or *evaluated as a string*. The last two modes reflect an additional step applied after the normal evaluation.

When an Expression is *evaluated numerically*, a string result is converted to its numerical equivalent based on the numeric characters that begin the string. A string beginning with non-numeric characters is converted to numeric zero.

Similarly, when an Expression is *evaluated as a string*, any numeric result is converted to a string equivalent. In reports and Data Streams, the representation of large numbers can be controlled by suppressing scientific notation, as is done in the Model Settings Notebook: This is set in the "Reports" page of the Model Settings Notebook.

### CHOOSE **Edit / Settings**

then select the **Report** page. Then set the checkbox marked "Scientific Notation" as desired. Scientific notation uses a base value followed by a power of ten. For example, eleven hundred would be represented as 1.1e3 in scientific notation.

Special rules apply when a simulation is run in GPSS/PC Compatibility Mode. This are discussed more fully in Chapter 1. In Compatibility Mode, SNAs are truncated, and when Bvariable Entities and Variable Entities are evaluated, intermediate results are truncated.

## Data Conversion

In general, you do not need to worry about data types and conversions. All the work is done behind the scenes.

Data types are converted, as needed, when operated upon, or when processed as arguments to Procedures. For example, strings combined by an arithmetic + operator are converted to numeric values and the results are added together yielding a numeric result. Similarly, String Procedures that take a string as an argument, will convert a numeric value to its string equivalent, if necessary.

Data items are not changed by evaluation unless operated upon, or used as an argument to a Procedure. Arithmetic operators change arguments to numeric values before performing the operation, and all Procedures coerce each argument to the suitable form. This is done automatically.

Coercion only occurs to operand or an operator, or to arguments of a built-in Procedure during evaluation. The arguments of user defined PLUS Procedures are not coerced prior to invocation of the Procedure. However, the coercion of operand and arguments may occur within the body of the User Procedure, if required by the other PLUS Statements.

When an operand or argument is coerced to a string, any numeric value must be converted to its ASCII equivalent. Normally, extremely large or small numbers are presented in scientific notation in mantissa-exponent format. Scientific notation can be suppressed by a setting in the "Report" page of the Model Settings Notebook.

When a string is coerced to a numeric value, only the first characters that can be interpreted as numeric are used. If there are none, the string is converted to numeric 0. When a real value is coerced to integer, it is truncated toward 0. For example, 2.1 truncates to 2, and -2.1 truncates to -2.

### 8.2.4. PLUS Statements

Although GPSS Statements are restricted to a single line of text of up to 250 characters, PLUS Statements can span many text lines. PLUS PROCEDURE Statements may appear anywhere in the Model except in another PROCEDURE Statement. The other PLUS Statements can only appear inside a PLUS Procedure.

Only a small number of statement types are needed by PLUS to provide a powerful programming language. They are:

- ◆ **PROCEDURE** - Define a PLUS Procedure.
- ◆ **EXPERIMENT** - Define a PLUS Experiment.
- ◆ **TEMPORARY** - Define and restrict the scope of a local User Variable or local Matrix.
- ◆ **BEGIN / END** - Compound Statement. Create a block of PLUS Statements.
- ◆ **Assignment** - Set the value of a Named Value or Matrix element.
- ◆ **Procedure Call** - Invoke a Library Procedure.
- ◆ **Labeled Statement** - The superclass of Statements that begin with a Label.
- ◆ **IF / THEN** - Test an Expression and act on a "TRUE" result.
- ◆ **IF / THEN / ELSE** - Test an Expression and act on the result.
- ◆ **WHILE / DO** - Perform action repetitively.
- ◆ **GOTO** - Jump to a new location within the Procedure.
- ◆ **RETURN** - Finish the processing and, optionally, give a result to the caller.

For full examples of GPSS World Models using PLUS Routines, see the PLUS PRIMER in Chapter 17 of the GPSS World Tutorial Manual. Here we present the PLUS Statement Types in alphabetic order.

## 1. Assignment Statement

### Syntax

**Value = Expression;**

*LValue* - A Named Value or Matrix element.

*Expression* - A well-formed PLUS Expression, defined above.

### Intended Use

Assignment Statements are used to set the value of variables. You can assign any data type to a Named Value or a Matrix element. These may be restricted in scope by a TEMPORARY or TEMPORARY MATRIX Statement in the same PLUS Procedure.

### Example

**Alert = "TRUE";**

In this example, the Named Value, Alert, is given the value of the string constant "TRUE".

### Action

An Assignment Statement evaluates the Expression on the right side and sets the value of the Named Value or Matrix element equal to a copy of the result.

If the *Name* used for the *LValue* is declared in a Temporary Declaration in the current Procedure, the assignment goes to a temporary User Variable or Matrix with local scope. Otherwise, the *Name* is assumed to be a global reference to a permanent Matrix or User Variable known throughout the simulation.

## 2. Compound Statement

### Syntax

**BEGIN StatementList END;**

*StatementList* - A list of well-formed PLUS Statements. Instantiated by a sequence, possibly null, of PLUS Statements. Each PLUS Statement contains an internal semicolon terminator. Must be *StatementSequence*.

### Intended Use

Compound Statements are used primarily in Procedure definitions, and to control the flow of action within a Procedure.

In a Procedure definition, a PLUS Procedure Statement requires a single statement in order to define a Procedure. A Compound Statement is generally used here, so that an arbitrarily complex PLUS Procedure may be defined.

Compound Statements are also used to group several PLUS Statements into a single block of statements. This is useful in IF Statements and WHILE Statements so that only a specific group of statements gets performed when a certain decision is made.

### Example

```
PROCEDURE Decision(Indicator) BEGIN
    TEMPORARY Return_Value;

    Return_Value = Old_Indicator;
    IF (Indicator > 0) then BEGIN
        ALERT = "TRUE";
        Old_Indicator=Indicator;
    END;
    ELSE Alert = "FALSE";

    RETURN Return_Value;

END;
```

In this example, Compound Statements are used in both ways. The PROCEDURE Statement includes a Compound Statement for holding the body of the PLUS Procedure. Internally, the "True" branch of the IF Statement uses a Compound Statement to cause more than one statement to be performed when the Expression in the IF Statement tests TRUE.

### Action

A Compound Statement groups other PLUS Statements so they can be treated as a unit. A Compound Statement can be used anywhere a PLUS Statement is required.

### 3. EXPERIMENT Statement

#### Syntax

**EXPERIMENT *Name* ( *ArgumentList* ) *Statement***

*Name* - A user-defined Procedure Name. Must be *Name*.

*ArgumentList* - A sequence of user-defined names, separated by commas, used as formal argument list. Instantiated by a list of *Name* items, possibly null, separated by commas.

*Statement* - Procedure body. A PLUS statement. Must be *Statement*.

#### Intended Use

An Experiment is a special kind of PLUS Procedure. EXPERIMENT Statements are used to define special User Procedures used to control multiple runs of a Simulation. Normally, the statement used as the Experiment body is a Compound Statement.

Experiments are generally used in connection with the DoCommand() library procedure, to control the Simulations, and the ANOVA library procedure to automatically analyze results. Normally, the Experiment fills a Global Matrix Entity with results and passes it to the ANOVA procedure. You can create screening or optimizing experiments automatically using GPSS World's Automatic Experiment Generators, by opening the appropriate dialog in the Edit Menu of the Main Window.

An Experiment can only be invoked by a CONDUCT Command. The Experiment Generators will load a Function Key with the appropriate CONDUCT Command, if you like.

Experiments, and any simple Procedure called directly or indirectly by an Experiment, may use the DoCommand library procedure to execute GPSS Statements including Commands and Block Statements.

#### Example

An example of an Experiment is discussed in detail in Lesson 19 of the *GPSS World Tutorial Manual*.

#### Action

An EXPERIMENT Statement creates a User Defined Procedure. When it is Translated, the Procedure is added to the simulation's Procedure Library, and is available for invocation by a CONDUCT Command.

The optional Formal Argument list passed to the Experiment by the CONDUCT Command is used to create temporary User Variables, addressed by the given names. Each receives a copy of the value resulting from evaluating the actual argument asserted in the Experiment invocation. Later references to the formal argument name refer to the variable created in this manner. The arguments are often used to specify which part of a response surface is to be explored by this invocation of the Experiment. With proper planning, Experiments can be HALTed, Saved, and restarted later.

While running, an Experiment takes control of the Simulation Object. Having begun an Experiment with a CONDUCT Command, your ability to interact with the simulation is limited. You can always display the running Simulation System Clock ( View / Clock ), but generally you will have to abort the Experiment in order to modify the Simulation. Then, unless you have planned ahead, you may have to restart the Experiment from the beginning.

### 4. GOTO Statement

#### Syntax

**GOTO *Label* ;**

*Label* - A PLUS Statement Label. A unique name appearing on a Labeled Statement, defined below.

## Intended Use

GOTO Statements are used to cause the processing sequence of a Procedure to jump abruptly to a specific statement, tagged with a PLUS Statement Label.

## Example

**GOTO Sanctuary;**

In this example, when the PLUS Procedure comes upon the GOTO Statement, it immediately finds the statement with the label, SANCTUARY, and begins there.

## 5. IF Statement

### Syntax

**IF ( *Expression* ) THEN *Statement***

*Expression* - A well-formed PLUS Expression, defined above. Must be *Expression*.

*Statement* - Conditional statement. A PLUS Statement.

## Intended Use

IF Statements are used primarily to conditionally perform a block of statements

## Example

```
IF (Indicator > 0) THEN BEGIN  
    Alert = "TRUE";  
    Old_Indicator=Indicator;  
END;
```

In this example, the Expression is evaluated numerically. True logical relations yield an integer 1 result. If the result of the evaluation is nonzero, the block of Assignment Statements is performed.

## Action

An IF Statement evaluates an Expression numerically, and determines if the result is nonzero. If so, the Conditional Statement, which may be compound, is performed.

## 6. IF-ELSE Statement

### Syntax

**IF ( *Expression* ) THEN *Statement1* ELSE *Statement2***

*Expression* - Test Expression. A well-formed PLUS Expression, defined above. Must be *Expression*.

*Statement1* - True branch statement. A PLUS Statement. Must be *Statement*.

*Statement2* - False branch statement. A PLUS Statement. Must be *Statement*.

### Intended Use

IF-ELSE Statements are used primarily to choose between two statement blocks, only one of which is to be performed.

### Example

```
IF (Indicator > 0) THEN BEGIN
    Alert = "TRUE";
    Old_Indicator=Indicator;
END;
ELSE Alert = "FALSE";
```

In this example, the Expression is evaluated numerically. True logical relations yield an integer 1 result. If the result of the evaluation is nonzero, the true branch statement is performed. In this case, the block of Assignment Statements is performed. If the result of numerically evaluating the test Expression is zero, the very last Assignment Statement is performed, instead.

### Action

An IF-ELSE Statement evaluates an Expression numerically, and determines if the result is nonzero. If so, the true branch statement, which may be compound, is performed. Otherwise, the false branch statement, which may also be compound, is performed.

ELSE clauses pair with the most recent IF clause. If several IF Statements appear in succession just before an ELSE clause, the latest unpaired IF clause is paired with the ELSE clause to form an IF-ELSE Statement. You can avoid ambiguity by using Compound Statements within IF and IF-ELSE Statements.

## 7. Labeled Statement

### Syntax

***Label : Statement***

*Label* - A PLUS Statement Label. A unique user-defined name. Must be *Name*.

*Statement* - A PLUS Statement. Must be *Statement*.

### Intended Use

Labeled Statements are used as the targets of GOTO Statements.

### Example

```
GOTO Sanctuary;
...
Sanctuary: RETURN 100;
```

In this example, when the PLUS Procedure comes upon the GOTO Statement, it immediately finds the statement with the label, SANCTUARY, and begins there. In this case, the Procedure terminates returning a numerical value of 100.

## 8. PROCEDURE Statement

## Syntax

### **PROCEDURE Name ( ArgumentList ) Statement**

*Name* - A user-defined Procedure Name. Must be *Name*.

*ArgumentList* - A sequence of user-defined names, separated by commas, used as formal argument list. Instantiated by a list of *Name* items, possibly null, separated by commas.

*Statement* - Procedure body. A PLUS statement. Must be *Statement*.

## Intended Use

PROCEDURE Statements are used to define User Procedures. Normally, the statement used as the Procedure body is a Compound Statement.

## Example

```
PROCEDURE Decision(Indicator) BEGIN
    TEMPORARY Return_Value;
    Return_Value = Old_Indicator;
    IF (Indicator > 0) THEN BEGIN
        Alert = "TRUE";
        Old_Indicator=Indicator;
    END;
    ELSE Alert = "FALSE";
    RETURN Return_Value;
END;
```

In this example, a PROCEDURE Statement is used to define a User Procedure. When Translated, the Procedure is integrated into the Procedure Library of the simulation. Thereafter, it can be invoked by an appropriate Procedure Call, such as:

**Result = Decision(200+My\_Constant);**

## Action

A PROCEDURE Statement creates a User Defined Procedure. When it is Translated, the Procedure is added to the simulation's Procedure Library, and is available for invocation.

The Formal Argument list is used to create temporary User Variables, addressed by the given names. Each receives a copy of the value resulting from evaluating the actual argument asserted in the Procedure invocation. Later references to the formal argument name refer to the variable created in this manner.

Procedures have global scope. They may be invoked from anywhere within the simulation.

## 9. Procedure Call Statement

## Syntax

### **ProcedureName ( ExpressionList ) ;**

*ProcedureName* - A name used to define a Procedure in the Procedure Library. May be a Math Procedure, a String Procedure, a Probability Distribution, or a User Defined PLUS Procedure. Must be *Name*.

*ExpressionList* - The argument list of the Procedure One or more well-formed PLUS Expressions, separated by commas. Must be *ExpressionList*.

## Intended Use

Procedure Call Statements are used to invoke Procedures in the Library.

### Example

**Real\_Value=Beta(1,100,200,2,2);**

In this example, a sample is drawn from the Beta probability distribution. Pre defined Procedures are available for math and string functions, and probability distributions. They are discussed later in this Chapter.

### Action

A Procedure Call Statement invokes a Procedure in the Library. Pre-defined and User Defined Procedures may be called.

Expressions may be used as arguments to Procedures. They are evaluated at the very beginning of the invocation. The arguments of built-in Procedures are coerced to a specific data type. This is discussed in the documentation of the specific Procedure. The arguments to User Defined Procedures are evaluated normally.

## 10. RETURN Statement

### Syntax

**RETURN Expression ;**

*Expression* - Optional. A well-formed PLUS Expression. Must be *Expression* or *Null*.

### Intended Use

RETURN Statements are used to terminate the processing of a PLUS Procedure, and to establish the result value to be used as the result of the Procedure invocation.

### Example

**RETURN "Success";**

In this example, a RETURN Statement is used to complete the Procedure and to establish a string constant as the result of the Procedure invocation.

### Action

When a RETURN Statement is performed, the Expression in the RETURN Statement, if any, is evaluated normally. The Procedure is terminated, and all temporary Named Variables and temporary Matrices are deleted. If an Expression was asserted in the RETURN Statement, its result is used as the result of the Procedure. If there is no Expression, integer 0 is used.

## 11. TEMPORARY Statement

### Syntax for TEMPORARY declaration

**TEMPORARY NameList ;**

*NameList* - Name List. A list of user defined names to become local User Variables. *Namelist* is instantiated by a list of *Name* items separated by commas.

### Syntax for TEMPORARY MATRIX declaration

## **TEMPORARY MATRIX *Name*[ *IntegerList* ] ;**

*Name* - A user defined local Matrix name. Must be *Name*.

*IntegerList* - A sequence of 1 to 6 strictly positive integers, separated by commas. The number of elements in each dimension of the Matrix. *IntegerList* is instantiated by a list of up to 6 *PosInteger* items, separated by commas.

### **Intended Use**

TEMPORARY Statements are used to create Named Values and Matrices that exist only during the invocation of a PLUS Procedure.

### **Example**

```
TEMPORARY Return_Value,Accumulator;  
TEMPORARY MATRIX DataArray[2,3,4];
```

In this example, two TEMPORARY Statements are used in a Procedure. The first creates two Named Values for use within a single Procedure invocation. The second creates a 3 dimensional local Matrix named DataArray.

Temporary data are not automatically initialized. You must assign data to temporary Named Values and Matrix Elements before you can refer to them in Expressions.

### **Action**

A TEMPORARY declaration creates one or more uninitialized Named Values for used during a single Procedure invocation.

A TEMPORARY MATRIX declaration creates a single uninitialized Matrix, of up to 6 dimensions, for use during a single Procedure invocation.

TEMPORARY Named Values and TEMPORARY MATRICIES have local scope. They can be accessed only within their containing Procedure, and not by Procedures invoked from the containing Procedure. Named Variables and Matrices not declared in temporary declarations have global scope, are known throughout the Model, and exist for the life of the simulation. A global matrix must be defined in a GPSS MATRIX Statement.

When a Procedure terminates, all of its temporary Named Values and temporary Matrices are freed.

## **12. WHILE Statement**

### **Syntax**

#### **WHILE ( *Expression* ) DO *Statement***

*Expression* - Test Expression. Required. A well-formed PLUS Expression, defined above. Must be *Expression*.

*Statement* - Target statement. Required. A PLUS Statement.

### **Intended Use**

WHILE Statements are used primarily to perform repetitive actions.

### **Example**

```
Accumulator=1;  
Counter=1;  
WHILE (Counter<=X_Integer) DO BEGIN  
    Accumulator=Accumulator#Counter;  
    Counter=Counter+1;  
END;
```

In this example, a Compound Statement is performed repetitively in a while loop. If X\_Integer is a positive integer, the while loop will continue to accumulate the factorial of X\_Integer in the Named Value, Accumulator.

Each time the target statement is performed, the Named Value Counter is incremented. When Counter becomes larger than X\_Integer, the target statement is not performed, and processing continues with the statement after the WHILE Statement.

### Action

When a WHILE Statement is encountered, the test Expression is evaluated numerically. If the result is nonzero, the target statement is performed, and the "Test-Perform" process is repeated. If the test Expression is zero, the target statement is not performed, and instead, processing continues with the statement following the While Statement.

To avoid a nonterminating loop, you must ensure that the evaluation of the test Expression becomes zero, at some time. Normally, this is done somewhere in the target statement by an Assignment Statement or Procedure call.

## 8.3. The Procedure Library

A Procedure must be in a Procedure Library for you to invoke it during a simulation. There are two kinds of libraries. The User Library and the GPSS World Library. The User Library is the collection of PLUS Procedures you have included in the Model. The GPSS World Library contains a set of ready made mathematical and string Procedures that you can invoke in any PLUS Expression.

The Procedure library is a set of PLUS Procedures that you can call in Expressions. Some Procedures are supplied for you, but you can define and add your own Procedures as well.

The built-in part of the Procedure Library includes Utility Procedures, File Procedures, Dynamic Call Procedures, Math Procedures, Probability Distributions, String Procedures, and Query Procedures

### 8.3.1. Utility Procedures

The GPSS World Procedure Library includes important Utility Procedures needed for the control of simulation runs and analysis of Experiments.

#### 1. DoCommand - Translate and Execute a GPSS Statement

##### Syntax

**DoCommand( CommandString )**

##### Arguments

*CommandString* - A string representation of the GPSS Statement to be executed. Required. Coerced to string. The argument must be *Expression*.

##### Return Value

None.

##### Example

`DoCommand("SHOW ""This is my string"" ");`

##### Action

DoCommand translates its argument string in global scope and then sends the result to the simulation for execution. Only Experiments or Procedures invoked during an Experiment can invoke

the DoCommand library procedure. It is this powerful procedure which allows the Experiment to control the simulation environment.

Since the string is Translated in global scope, Temporary Variables and Arguments are not accessible and should not be represented in the command string.

In its invocations, a string containing a Command or Statement is Translated in the global context and then executed by the Simulation Object.

Here are some tips for using the DoCommand library procedure.

1. The PolyCatenate String Procedure can combine any number of strings.
2. Use 4 double quotes around strings within strings. Each inner string should be sandwiched by pairs of double quotes, which get reduced to single quotes when translated.
3. Do not pass a string to DoCommand that contains the name of a TEMPORARY variable or argument because these variables are not accessible in global context. You can always use them in a string created before the call to DoCommand.

Normally, GPSS World simulations enqueue all commands (except HALT and SHOW) on a low priority Command Queue and work on them one at a time until the queue is empty. However, DoCommand behaves a little differently. It does not return to the calling procedure until the low priority Command Queue is empty. This means that after a START Command issued through the DoCommand library procedure returns to the calling Procedure, the simulation has completed and is ready for the results to be extracted.

## 2. ANOVA - Perform Multiway Analysis of Variance.

### Syntax

```
StandardError = ANOVA( ResultMatrixName,  
                           ReplicateDimension, InteractionLimit )
```

### Arguments

*ResultMatrixName* - The GPSS Matrix containing the results to be analyzed. Required. May be a GPSS Matrix of up to 6 dimensions of any shape. The argument must be *Name*.

*ReplicateDimension* - The dimension of the Result Matrix used for replicates. Use 0 if there are no replicates. Required. Coerced to integer. The argument must be *Expression*.

*InteractionLimit* - The Limit of Factor Interactions to be analyzed. This value can be 1, 2, or 3. It is normally used to preserve Degrees of Freedom for the estimate of the Standard Error instead of using them on an interaction presumed to be unimportant. Required. Coerced to integer. The argument must be *Expression*.

### Return Value

*StandardError* - The Standard Error of the Analysis of Variance is returned. If an error occurs, the value 0 is returned. Real.

### Example

**SHOW ANOVA(ABC,1,1)**

**3.8756**

This example uses a SHOW Command to demonstrate how the ANOVA Procedure can be invoked, returning the Standard Error of the Analysis of Variance. The ANOVA Procedure also produces an ANOVA Table and a table of descriptive statistics and writes them to the Journal Window.

### Action

The ANOVA Procedure analyzes the data in the Result Matrix, excluding any element with the UNSPECIFIED Data Type. If a Replicate Dimension is specified in the ANOVA invocation, that dimension of the matrix is presumed to contain the results of replicate runs and is used exclusively to estimate the Standard Error. Two and Three level interactions between factors are normally included in the analysis, unless they are restricted by the third argument. If the third argument is 2, only 2-way interactions are estimated, if 1, no interactions are estimated. Restricting interactions presumed to be 0 has the benefit of providing a better estimate of the Standard Error.

If possible, the ANOVA Procedure writes an ANOVA Table and a table of descriptive statistics to the Journal Window. The resulting Standard Error is returned by the procedure.

If a GPSS Table Entity exists with the name *ResultMatrixName*\_Residuals, where *ResultMatrixName* is the name used for the first argument of the ANOVA invocation, a table of residuals will be created in it based on the Analysis of Variance.

A full discussion of the ANOVA Procedure and its output is contained in Chapter 12. Also, Lesson 19 of *The GPSS World Tutorial Manual* treats the use of the ANOVA Procedure with User Experiments.

### 3. Exit - Conclude the GPSS World Session.

#### Syntax

**Exit( *ExitCode* )**

#### Arguments

*ExitCode* - -1, 0, 1 to specify "Save None", "Query Modified", or "Save All" open Model and Simulation Objects.

#### Return Value

*None*

#### \Example

**Exit( 0 )**

This example uses the Exit library procedure to conclude the GPSS World Session. If any Model or Simulation Objects have been modified, a dialog box will appear for each asking if the object should be saved to file or not.

#### Action

The Exit procedure HALTs all simulations, then closes all objects one at a time.

The Exit Code can be used to control the writing of Model Objects and Simulation Objects to files. If Operand a is 0, all modified files bring up a message box inquiring as to whether or not each object should be saved. If Operand A is 1, all Objects are saved. If Operand A is -1, no Objects are saved.

The Exit() Procedure is useful when sessions are run in Batch Mode which is discussed in Section 2.3.2.

### 8.3.2 Math Procedures

The World Procedure Library includes several Mathematical Procedures. In all cases, the argument is coerced to a numeric value before the Procedure performs its operation. Angular data are in radians. Numeric values are returned as real numbers.

The Math Procedures are

**ABS( *Expression* )** - Absolute value.

**ATN( *Expression* )** - Arctangent in radians.

**COS( *Expression* )** - Cosine. *Expression* must be in radians.

**EXP( *Expression* )** - e raised to the power given by *Expression*.

**INT( *Expression* )** - Truncation toward zero.

**LOG( *Expression* )** - Natural logarithm.

**SIN( *Expression* )** - Sine. *Expression* must be in radians.

**SQR( *Expression* )** - Square Root.

**TAN( *Expression* )** - Tangent. *Expression* must be in radians.

### 8.3.3 Query Procedures

Transaction Query Procedures are available which return information based on any Transaction in the simulation. Except for QueryXNExist(), if you attempt to query a nonexistent Transaction, an Error Stop will occur. If there is any question, you should test for existence first.

All arguments are coerced to integers in order to look up either a Transaction or one of its parameters. The Transaction state query Procedures now include:

**QueryXNExist( *TransactionNumber* )** - Return integer 1 if the Transaction exists in the simulation, integer 0 , if not.

**QueryXNParameter( *TransactionNumber*, *Parameter* )** - Return the value of a Transaction Parameter. Error Stop occurs if the Parameter does not exist.

**QueryXNAsemblySet( *TransactionNumber* )** - Return the integer Assembly Set of a Transaction.

**QueryXNPriority( *TransactionNumber* )** - Return the integer Transaction Priority of a Transaction.

**QueryXNM1( *TransactionNumber* )** - Return the numeric Mark Time of a Transaction.

### 8.3.4 String Procedures

The GPSS World Procedure Library contains a number of String Procedures that make it easy to manipulate string data types. As with all built-in Procedures, the String Procedures coerce arguments into proper form before processing them.

Strings are sequences of ASCII characters. The individual characters can be addressed by an index called an "Offset". String Procedures that must find a specific character use the Offset as a 1-relative index into an array of characters. The first character in the string is associated with an index of 1, the second with 2, and so on.

A null string is a string of length 0. It contains no characters, and is denoted " ". Null strings are still considered to be valid strings.

A word is a consecutive sequence of printable characters, not including blanks or tabs. Words may be separated by one or more blank and/or tab characters.

The remainder of this section describes the String Procedure Library. For each Procedure, the syntax of Procedure invocation is given in a syntax line. Any Procedure may be invoked in an Expression, without the assignment of the result, as well as in a PLUS Assignment Statement.

**Align( *InsertString*, *SourceString*, *Offset* )** - Return a copy of one string placed in another, right justified.

**Catenate( *String1*, *String2* )** - Return a copy of two strings combined into one.

**Copies( *SourceString*, *Count* )** - Create a string from many copies of a string.

**Datatype( *Datum* )** - Return a string denoting the data type of the argument.

**Find( *TestString*, *SourceString* )** - Return the Offset of one string in another.

**Left( *SourceString*, *MaxCount* )** - Return a copy of a substring starting on the left.

**Length( *SourceString* )** - Return the count of characters in a string.

**Lowercase( *SourceString* )** - Return the lowercase representation of a string.

**Place( *InsertString*, *SourceString*, *Offset* )** - Place one string in another. Left justify.

**PolyCatenate( *String1*, *String2*, ... )** - Return a copy of two or more strings combined into one.

**Right( *SourceString*, *MaxCount* )** - Return a copy of a substring starting on the right.

**String( *Datum* )** - Convert a data item to its string equivalent.

**StringCompare( *String1*, *String2* )** - Return an integer result if string comparison.

**Substring( *SourceString*, *Offset*, *MaxCount* )** - Return a copy of a substring of the string argument.

**Trim( *SourceString* )** - Remove leading and trailing white space.

**Uppercase( *SourceString* )** - Return the uppercase equivalent of a string.

**Value( *Datum* )** - Return the numeric equivalent of a string.

**Word( *SourceString*, *WordNumber* )** - Return a copy of one of the words in a string.

## 1. Align - Right justify a string.

### Syntax

*ReturnString*=Align(*InsertString*,*SourceString*,*Offset* )

### Arguments

*InsertString* - The string to be copied. Required. Coerced to string. The argument must be Expression.

*SourceString* - The string to be overwritten. Required. Coerced to string. The argument must be Expression.

*Offset* - The 1-relative offset in *SourceString* to receive the rightmost character of *InsertString*. Required. Coerced to integer. The argument must be Expression.

### Return Value

*ReturnString* - The string created as a result of inserting *InsertString* into *SourceString* at offset *Offset*.

### Example

SHOW Align("ABC","123456789",6)

"123ABC789"

This example uses a SHOW Command to demonstrate how the align() string Procedure right justifies the insert string at position 6 of the target string, and displays the result.

### Action

Align( ) coerces the first and second arguments to strings, and the third to an integer. It then creates a string of blanks large enough to contain the result, and copies the target string into the result string. Finally, it copies the insert string, or the part of it that fits, into the result string.

If the Offset is larger than the target string, the resulting string will be larger, as well. Any characters not specified by the argument strings will be blanks.

If the Offset is less than the length of the insert string, only the characters that fit will be inserted, starting from the rightmost character of the insert string.

If the Offset is not strictly positive, no characters will be inserted.

## 2. Catenate - Combine two strings.

### Syntax

***ReturnString = Catenate( String1, String2 )***

### Arguments

*String1* - The string to be first in the concatenated result string. Required. Coerced to string. The argument must be *Expression*.

*String2* - The string to be second in the concatenated result string. Required. Coerced to string. The argument must be *Expression*.

### Return Value

*ReturnString* - The string created as a result of placing *String2* immediately after *String1*.

### Example

**SHOW Catenate("ABC","123")**

**"ABC123"**

This example uses a SHOW Command to demonstrate how the Catenate() string Procedure combines two strings.

### Action

Catenate( ) first coerces both arguments to strings. It then creates a string large enough to contain the result, and copies the first string and the second string into the result string.

## 3. Copies - Build a string from multiple copies.

### Syntax

***ReturnString = Copies( SourceString, Count )***

### Arguments

*SourceString* - The string to be duplicated. Required. Coerced to string.  
The argument must be *Expression*.

*Count* - The number of copies to make. Required. Coerced to integer.  
The argument must be *Expression*.

## Return Value

*ReturnString* - The string created as a result of concatenating *Count* copies of *SourceString*.

## Example

SHOW Copies("ABC ",3)

"ABC ABC ABC "

This example uses a SHOW Command to demonstrate how the copies( ) string Procedure creates a string out of 3 copies of an existing string.

## Action

Copies( ) coerces the first argument to a string and the second to an integer. It then creates a string large enough to contain the result, and copies the source string to the result string once for each count in the second argument.

If the count is not strictly positive, a null string results.

## 4. DataType - Determine the type of a data item.

### Syntax

***ReturnString = DataType( Datum )***

### Arguments

*Datum* - The data item to be examined. Required. The argument must be *Expression*.

## Return Value

*ReturnString* - The string identifying the data type. One of:

"INTEGER" - *Datum* is a 32 bit integer,

"REAL" - *Datum* is a double precision floating point number,

"STRING" - *Datum* is a character string, or

"UNSPECIFIED" - *Datum* has not been given a value.

## Example

SHOW DataType("ABC")

"STRING"

This example uses a SHOW Command to demonstrate how the DataType() string Procedure identifies the data type of a string constant.

## Action

DataType( ) does not coerce its argument. It determines the data type of the argument and returns the string constant associated with that type.

## 5. Find - Find one string in another.

### Syntax

***ReturnInteger = Find( TestString, SourceString )***

### Arguments

*TestString* - The string to be found. Required. Coerced to string. The argument must be *Expression*.

*SourceString* - The string to be tested for occurrence of *TestString*. The argument must be *Expression*.

### Return Value

*ReturnInteger* - The 1-relative offset of the first occurrence of *TestString* in *SourceString*. A real value. Zero if not found.

### Example

**SHOW Find("ABC","123ABC789")**

**4**

This example uses a SHOW Command to demonstrate how the find() string Procedure finds the occurrence of "ABC" starting at position 4 of string "123ABC789".

### Action

Find( ) coerces the first and second arguments to strings, and tests the second argument for the existing of a substring equal to the first argument. If it cannot be found, find( ) returns 0. Otherwise, find( ) returns the 1-relative offset in the second string of the first occurrence of the first string.

## 6. Left - Return an initial substring.

### Syntax

***ReturnString = Left( SourceString, MaxCount )***

### Arguments

*SourceString* - The source string to be used to create the substring. Required. Coerced to string. The argument must be *Expression*.

*MaxCount* - The maximum number of characters to be used in the substring. Required. Coerced to integer. The argument must be *Expression*.

### Return Value

*ReturnString* - The string created as the left *MaxCount* characters of *SourceString*.

### Example

**SHOW Left("123456789",6)**

**"123456"**

This example uses a SHOW Command to demonstrate how the left( ) string Procedure extracts the first 6 characters of the string "123456789".

### Action

`Left( )` coerces the first argument to a string, and the second to an integer. If `MaxCount` is less than 0, it is made equal to 0. `Left( )` then creates a string of length equal to the smaller of `MaxCount` and the length of `SourceString`. The result string is then filled with characters from `SourceString`, starting with the first.

## 7. Length - Count string characters.

### Syntax

***ReturnInteger = Length( SourceString )***

### Arguments

`SourceString` - The string to be examined. Required. Coerced to string. The argument must be `Expression`.

### Return Value

`ReturnInteger` - The number of characters in the string. A Real value.

### Example

**SHOW Length("ABC")**

**3**

This example uses a SHOW Command to demonstrate how the `length()` string Procedure returns the character count of the string "ABC".

### Action

`Length( )` coerces the argument to a string, counts its characters, and returns the count as an integer. Null strings have a count of 0.

## 8. Lowercase - Convert string to lower case.

### Syntax

***ReturnString = LowerCase( SourceString )***

### Arguments

`SourceString` - The string to be converted. Required. Coerced to string. The argument must be `Expression`.

### Return Value

`ReturnString` - The string created as a result of converting `SourceString` to lower case.

### Example

**SHOW Lowercase("123 AbC")**

**"123 abc"**

This example uses a SHOW Command to demonstrate how the `lowercase( )` string Procedure copies a string and converts the upper case letters to lower case.

### Action

`Lowercase( )` coerces the argument to a string, and creates a copy of the string. It then converts the upper case letters in the copy to lower case and returns it as the result string.

## 9. Place - Left justify a string.

### Syntax

***ReturnString =Place(InsertString,SourceString,Offset )***

### Arguments

*InsertString* - The string to be inserted. Required. Coerced to string. The argument must be *Expression*.

*SourceString* - The string to receive the insertion. Required. Coerced to string. The argument must be *Expression*.

*Offset* - The 1-relative offset in *SourceString* to receive the leftmost character of *InsertString*. Required. Coerced to integer. The argument must be *Expression*.

### Return Value

*ReturnString* - The string created as a result of inserting *InsertString* into *SourceString* at offset *Offset*.

### Example

**SHOW Place("ABC","/123456789",3)**

**"123ABC789"**

This example uses a SHOW Command to demonstrate how the place() string Procedure left justifies the insert string at position 3 of the target string, and displays the result.

### Action

`Place( )` coerces the first and second arguments to strings, and the third to an integer. It then creates a string of blanks large enough to contain the result, and copies the target string into the result string. Finally, it copies the insert string into the result string.

If the end of the inserted string extends past the end of the original target string, the resulting string will be larger, as well. Any characters not specified by the argument strings will be blanks.

If the offset is less than 1, the leftmost characters of the insert string are truncated, and the remaining characters, if any, are inserted at position 1 of the target string.

## 10. PolyCatenate - Combine two or more strings.

### Syntax

***ReturnString = PolyCatenate(String1, ... )***

### Arguments

*String1, ...* - The strings to be combined in order in the concatenated result string. Required. Coerced to string. The arguments must be *Expression*.

### Return Value

*ReturnString* - The string created as a result of placing copies of the strings one after the other.

## Examples

**SHOW PolyCatenate("The ","time ","is ",AC1)**

**"The time is 0"**

This example uses a SHOW Command to demonstrate how the PolyCatenate( ) string Procedure combines four strings. The SNA AC1 is automatically coerced to a string.

**SHOW PolyCatenate("Yours","","","truly,")**

**"Yours truly,"**

This example uses a SHOW Command to demonstrate how the PolyCatenate() string Procedure combines three string constants.

## Action

PolyCatenate( ) first coerces all arguments to strings. Any number of arguments may be used. It then creates a string large enough to contain the result, and copies the argument strings into the result string one after the other. Any of the arguments can be a null string.

## 11. Right - Return a terminal substring.

### Syntax

***ReturnString = Right(SourceString, MaxCount)***

### Arguments

*SourceString* - The source string to be used to create the substring. Required. Coerced to string. The argument must be *Expression*.

*MaxCount* - The maximum number of characters to be used in the substring. Required. Coerced to integer. The argument must be *Expression*.

### Return Value

*ReturnString* - The string created as the right *MaxCount* characters of *SourceString*.

### Example

**SHOW Right("123456789",6)**

**"456789"**

This example uses a SHOW Command to demonstrate how the right() string Procedure extracts the last 6 characters of the string "123456789".

## Action

Right( ) coerces the first argument to a string, and the second to an integer. If *MaxCount* is less than 0, it is made equal to 0. Right( ) then creates a string of length equal to the smaller of *MaxCount* and the length of *SourceString*. The result string is then filled with the last *MaxCount* characters from *SourceString*.

## 12. String - Create string equivalent.

### Syntax

## ***ReturnString = String( Datum )***

### **Arguments**

*Datum* - The data item to be converted. Required. Coerced to string. The argument must be *Expression*.

### **Return Value**

*ReturnString* - The string created as a result of converting *Datum* to its string equivalent.

### **Example**

**SHOW String(12345)**

**"12345"**

This example uses a SHOW Command to demonstrate how the string() string Procedure converts the integer constant 12345 to the string constant "12345".

### **Action**

String( ) coerces the arguments a string, and returns a copy as the result. If the argument is already a string, it is not modified.

## **13. StringCompare - Compare two strings.**

### **Syntax**

## ***ReturnInteger = StringCompare( String1, String2 )***

### **Arguments**

*String1* - The first of two strings to be compared. Required. Coerced to string. The argument must be *Expression*.

*String2* - The second of two strings to be compared. Required. Coerced to string. The argument must be *Expression*.

### **Return Value**

*ReturnInteger* - Integer -1, 0, or 1 as *String1* precedes, equals, or succeeds *String2*.

### **Example**

**SHOW StringCompare("ABC","abc")**

**-1.0000000**

This example uses a SHOW Command to demonstrate how the StringCompare( ) string Procedure compares the two string constants and returns the result of the comparison.

### **Action**

StringCompare( ) coerces the first and second arguments to strings. It then compares them lexicographically.

If the two strings are identical, an integer 0 is returned. Otherwise the strings are compared character by character until a difference is detected.

If the first difference occurs because one string is shorter than the other, the shorter string is said to precede the longer one. If the shorter string is *String1*, -1 is returned. If the shorter string is *String2*, 1

is returned.

If the first difference occurs because the character from *String1* precedes that from *String2* in the ASCII collating sequence, *String1* is said to precede *String2*, and -1 is returned. Otherwise, *String2* is said to precede *String1*, and 1 is returned.

## 14. Substring- Return part of a string.

### Syntax

***ReturnString = SubString( SourceString, Offset, MaxCount )***

### Arguments

*SourceString* - The string to be used as the source of the substring. Required. Coerced to string. The argument must be *Expression*.

*Offset* - The 1-relative offset in *SourceString* of the first character of the substring. Required. Coerced to integer. The argument must be *Expression*.

*MaxCount* - The maximum number of characters from *SourceString* to use in the substring. Required. Coerced to integer. The argument must be *Expression*.

### Return Value

*ReturnString* - The string created as a result of creating a copy of a substring derived from *SourceString*.

### Example

**SHOW Substring("123456789",3,4)**

**"3456"**

This example uses a SHOW Command to demonstrate how the substring() string Procedure copies 4 characters from the string constant "123456789" to form a new string.

### Action

Substring( ) coerces the first argument to a string, and the second and third to integers. If the *Offset* argument is less than 1, it is set to 1. Substring ( ) then creates a string large enough to contain the lesser of the *MaxCount* argument or the number of characters remaining after the offset in the source string. Substring( ) then copies the characters from the substring, if any, into the result string.

## 15. Trim - Remove leading and trailing white space.

### Syntax

***ReturnString = Trim( SourceString )***

### Arguments

*SourceString* - The string to be used as the source of characters. Required. Coerced to string. The argument must be *Expression*.

### Return Value

*ReturnString* - The string created as a result of trimming blanks and tabs from *SourceString*.

### Example

**SHOW Trim(" A B C ")**

**"A B C"**

This example uses a SHOW Command to demonstrate how the Trim( ) string Procedure creates a new string by removing leading and trailing blanks from a string continuing them.

### Action

Trim( ) coerces the arguments to a string. It then creates a string large enough to contain the result, and copies the source string into the result string, omitting leading and trailing blank or tab characters.

## 16. Uppercase - Convert string to upper case.

### Syntax

***ReturnString* = Uppercase( *SourceString* )**

### Arguments

*SourceString* - The string to be converted. Required. Coerced to string. The argument must be *Expression*.

### Return Value

*ReturnString* - The string created as a result of converting *SourceString* to upper case.

### Example

**SHOW Uppercase("123 aBc")**

**"123 ABC"**

This example uses a SHOW Command to demonstrate how the uppercase( ) string Procedure copies a string and converts the lower case letters to upper case.

### Action

Uppercase( ) coerces the argument to a string, and creates a copy of the string. It then converts the lower case letters in the copy to upper case and returns it as the result string.

## 17. Value - Convert to numeric equivalent.

### Syntax

***ReturnReal* = Value( *Datum* )**

### Arguments

*Datum* - The data item to be converted. Required. The argument must be *Expression*.

### Return Value

*ReturnReal* - The double precision real numeric value equivalent to *Datum*. A real value.

## Example

**SHOW Value("123.4")**

**123.4000000**

This example uses a SHOW Command to demonstrate how the value( ) string Procedure converts the string "123.4" to its numeric value.

## Action

Value( ) determines the data type of the argument. Integers or strings are converted to their real numeric equivalent, and returned. Real arguments are returned without modification.

## 18. Word - Extract a word from a string.

### Syntax

**ReturnString = Word( SourceString, WordNumber )**

### Arguments

*SourceString* - The string to be examined. Required. Coerced to string. The argument must be *Expression*.

*WordNumber* - The 1-relative cardinal number of the word in *SourceString* to be returned. The argument must be *Expression*.

### Return Value

*ReturnString* - The string containing the word extracted from *SourceString*.

## Example

**SHOW Word("My country ♦tis of thee.",2)**

**"country"**

This example uses a SHOW Command to demonstrate how the word( ) string Procedure extracts the second word of a source string.

## Action

Word( ) coerces the first argument to a string, and the second to an integer. It then finds the word corresponding to the second argument, and creates a string large enough to contain it. Finally, it copies the characters of the word, if any, into the result string.

Words are consecutive printable characters other than tabs or blanks. If the corresponding word does not exist in the source string, a null string is returned.

## 8.3.5 Data Stream Procedures

The built-in GPSS World Library contains a number of Procedures for controlling Data Streams from within your own PLUS Procedures. Data Streams are discussed in Section 4.16. These can be used to read from or write to files or to store and access a large amount of data in memory.

You can invoke these procedures in place of using the corresponding GPSS Block. The library procedures allow you to do everything that the corresponding Blocks do with one exception: the

PLUS library procedures do not support Replace Mode.

**Open( *DataStream*, *FileNameString* )** - initialize a Data Stream.

**Close( *DataStream* )** - terminate a Data Stream and retrieve its error code.

**Read( *DataStream* )** - retrieve a text line from a Data Stream.

**Write( *DataStream*, *String* )** - pass a text line to a Data Stream.

**Seek( *DataStream*, *NewLinePosition* )** - set the Current Line Position of a Data Stream and retrieve the previous Line Position.

## 1. Open - Initialize a Data Stream

You can use a Data Stream to read and write to a file, or to maintain a set of directly accessible data in the memory of your computer. open, close, read, write and seek operations exist both as GPSS Blocks and as PLUS library procedures. You can perform complex file input/output operations within your own PLUS Procedures or by using higher level GPSS Block Entities, and you may mix the two modes. The Blocks are discussed in Chapter 7.

### Syntax

**ReturnCode=Open( *DataStream*, *FileNameString* )**

### Arguments

*DataStream* - The number of the new Data Stream to be opened. Required. Coerced to integer. The argument must be *Expression*.

*FileNameString* - The name of the file to be written to. Required. Coerced to string. The argument must be *Expression*.

### Return Value

*ReturnCode* - A code indicating the success of the operation. Success is 0, otherwise an error occurred which is described by the value of the return code, as listed below.

### Example

**Result = Open(2,"MYFILE.TXT");**

In this example, the Open() Procedure creates the type of Data Stream known as an I/O Stream, and gives it number 2 for identification. If an error occurs during the Open, an error code is returned.

### Action

The Open library procedure Block causes a Data Stream to be created, and sets the Current Line Position to 1.

The first argument must identify the Data Stream to be used. A Data Stream is a sequence of text lines used by a GPSS World simulation. Each Data Stream is identified by a unique number. There are 2 types of Data Stream:

1. Input/Output (I/O) Streams, and
2. In-Memory Streams.

The second argument is evaluated as a string. If it is a null string, an In-Memory Stream is created. Otherwise an I/O Stream is created, and the argument is presumed to be a file specification. If a path is not included in the file specification, it is assumed that the simulation directory is to be used.

A Data Stream is a sequence of text lines used by a GPSS World simulation. Each Data Stream is identified by a unique number, so that many can be processed at the same time within in a single simulation. Data Stream numbers are arbitrary positive integers, assigned by you.

If, when you Open() a file type Data Stream, you use a file name without a path, the directory of the Simulation Object is assumed to be the location of the file. When an existing file is found, it is completely loaded into virtual memory during the processing of the OPEN Block. If no file is found, it is assumed that you are creating one, and processing continues.

If the Data Stream has already been opened, the open operation concludes without raising an error condition.

If an error is detected, the error code is stored internally as well as being returned by the Procedure. A CLOSE Block can be used to retrieve a stored Error Code.

Data Streams are buffered in virtual memory. When an existing file is found by an open operation, it is completely loaded into virtual memory. If no file is found, it is assumed that you are creating one, and processing continues. After the open operation completes, all data is kept as part of the Simulation Object until the Data Stream is closed. Any changes to the data are returned to the file system only when the Data Stream is terminated by a CLOSE Block or a Close() library procedure.

Chapter 4 (4.16) contains a full discussion of Data Streams. The open operation can also be performed by an OPEN Block, described in Chapter 7, as well as the Open() PLUS library procedure, described here.

## Return Codes

- ◆ 0 - No Errors.
- ◆ 10 - Error. Filename too long. Data Stream not created.
- ◆ 11 - Error. Error while reading an external file. Data Stream not created.
- ◆ 12 - Error. Memory request was denied while trying to read an existing file. Data Stream not created.

## 2. Close - terminate a Data Stream and retrieves its error code.

### Syntax

**ReturnValue=Close( *DataStream* )**

### Arguments

*DataStream* - The number of the new Data Stream to be opened. Required. Coerced to integer. The argument must be *Expression*.

### Return Value

*ReturnValue* - A code indicating the success of the operation. Success is 0, otherwise an error occurred which is described by the value of the return code, as listed below. If an error occurred during a previous Open(), Read(), Write(), or Seek() operation, the original nonzero error code is returned, as described below.

### Example

**Result = Close(2);**

In this example, the Close() Procedure terminates the operation of Data Stream 2 and frees all the resources associated with it. If any error occurred in a previous operation involving Data Stream number 2, the original error code is returned. Otherwise a 0 value is returned.

### Action

The Close library procedure shuts down a Data Stream and retrieves its error code.

The argument is evaluated numerically, truncated, and used as the entity number of the Data Stream. The result must be a positive integer.

Chapter 4 (4.16) contains a full discussion of Data Streams. The close operation can be performed by a CLOSE Block, described in Chapter 7, as well as the Close library procedure, described here.

## Return Codes

- ◆ 0 - No Errors.
- ◆ 10 - Open Error. Filename too long. Data Stream not created.
- ◆ 11 - Open Error. Error while reading an external file. Data Stream not created.
- ◆ 12 - Open Error. Memory request was denied while trying to read an existing file. Data Stream not created.
- ◆ 21 - Read Error. A memory request was denied while trying to perform a read operation.
- ◆ 22 - Read Error. Data Stream has not been successfully opened.
- ◆ 31 - Write Error. A memory request was denied while trying to perform a Write operation.
- ◆ 32 - Write Error. Data Stream has not been successfully opened.
- ◆ 41 - Close Error. An I/O Error prevented the file from being written to disk.
- ◆ 43 - Close Error. Data Stream has not been successfully opened.
- ◆ 51 - Seek Error. Data Stream has not been successfully opened.

### 3. Read - retrieve a text line from a Data Stream.

#### Syntax

***ReturnString=Read( DataStream )***

#### Arguments

*DataStream* - The number of the new Data Stream to be opened. Required. Coerced to integer. The argument must be *Expression*.

#### Return Value

*ReturnString* - The string read from the Current Line of the Data Stream.

#### Example

**NextLine = Read(1);**

In this example, the Read() Procedure retrieves a text line from Data Stream number 1 and places a copy in the User Variable named NextLine. If an error occurs, a null string is assigned to NextLine and an Error Code indicating the specific reason is stored internally, and can be retrieved by a Close operation. These values are listed under the Close Procedure, described above.

#### Action

The argument is evaluated numerically, truncated, and used as the Data Stream Entity number. This must be a positive integer.

If an error is detected, the error code is stored internally. A CLOSE Block or a Close() Procedure invocation can be used to retrieve the Error. Chapter 4 (4.16) contains a full discussion of Data Streams.

The text line is determined by the Current Line Position, a 1-relative line index associated with the Data Stream. During a Read operation the line indicated by the Current Line Position, even if the line is a null string, is returned. Then, the Current Line Position is incremented, i.e. move to the next line number. If there is no line to be read, a null string is returned without any error code being stored. A seek operation can be used to change the Current Line Position.

## 4. Write - pass a text line to a Data Stream.

### Syntax

***ReturnCode=Write( DataStream,SourceString )***

### Arguments

*DataStream* - The number of the new Data Stream to be opened. Required. Coerced to integer. The argument must be *Expression*.

*SourceString* - The sting to become the next line in the Data Stream. Required. Coerced to integer. The argument must be *Expression*.

### Return Value

*ReturnCode* - A code indicating the success of the operation. Success is 0, otherwise an error occurred which is described by the value of the return code, as listed below.

### Example

**Result= Write(1,"New Line 20");**

In this example, the Write() Procedure sends a text line to Data Stream number 1. If an error occurs, a nonzero code is returned and is also stored internally. The error codes are described below.

### Action

The first argument is evaluated numerically, truncated, and used as the Data Stream Entity number. This must be a positive integer.

The second argument is evaluated as a string. The Write Procedure then:

1. Moves all text lines at, or after, the Current Line Position down one position.
2. If the Current Line Position is after the last text line, sets it to just after the last text line in the Data Stream.
3. Places a copy of the new text line at the Current Line Position.
4. Increments the Current Line Position.

If an error is detected, the error code is stored internally as well as being returned by the Procedure. A Close() Procedure invocation can be used to retrieve a stored Error Code.

Chapter 4 (4.16) contains a full discussion of Data Streams. WRITE Blocks support an additional mode of operation called "Replace Mode". It is not supported by the Write() library procedure.

## Error Codes

◆ 0 - No Errors.

◆ 31 - Error, A memory request was denied while trying to perform a write operation.

◆ 32 - Error, Data Stream has not been successfully opened.

## 5. Seek - set the Current Line Position of a Data Stream.

### Syntax

***PreviousLinePosition=Seek( DataStream,NewLinePosition )***

### Arguments

*DataStream* - The number of the new Data Stream to be opened. Required. Coerced to integer. The argument must be *Expression*.

*NewLinePosition* - The new 1-relative value of the Current Line Position of the Data Stream.

### Return Value

*PreviousLinePosition* - The value of the Current Line Position of the Data Stream before the Seek operation is executed.

### Example

**LastPosition = Seek(1,22);**

In this example, the Current Line Pointer of the Data Stream is changed to 22.

### Action

Each Data Stream has a *Current Line Position*. This is a 1-relative index to the next line position to be read or written. The first argument is evaluated numerically, truncated, and used as the Data Stream Entity number. The second argument is evaluated numerically, truncated, and used as the new Current Line Position of the Data Stream. Both must be positive integers.

If an error is detected, the error code is stored internally. A Close() Procedure invocation can be used later to retrieve the Error Code. These values are listed under the Close Procedure, described above.

## 8.3.6 Dynamic Call Procedures

The built-in GPSS World Library contains a number of Procedures for invoking functions that exist in external executable files including Dynamic Link Libraries, known as DLLs. You can use the appropriate Procedure, as discussed below, to invoke third party functions in separate executable files that observe the standard CDECL programming protocol. They may optionally return an integer return code, and they may optionally take an integer, char \* (string), or double (real) argument.

All PLUS Dynamic Call Procedures require you to name both the executable file (EXE or DLL) and the function that is to be invoked. Further, you must match the PLUS Procedure to the requirements of the target function. If the function requires an integer argument you must use the Call\_Integer() Procedure. Similarly, use the Call\_String() or the Call\_Real() to invoke a DLL function requiring a string (i.e. char \*) or a real (i.e. double) argument. Use the Call() Procedure to invoke a function with no (i.e. void) argument. The choice of PLUS Procedure allows the argument to be coerced to the appropriate data type required by the target function.

If the external function returns an integer return code, the PLUS Call Procedure will provide it as the result of the PLUS Procedure call. Otherwise, the resulting value will be meaningless. Clearly, you must know the detailed characteristics of the function you are calling.

The Dynamic Call Procedures are:

***ReturnCode = Call( ExecutableFileName, FunctionName )*** - Invoke an external CDECL function with no ( i.e. void) arguments.

**ReturnCode = Call\_Integer( ExecutableFileName, FunctionName, Argument)** - Invoke an external CDECL function with a single integer argument.

**ReturnCode = Call\_String( ExecutableFileName, FunctionName, Argument )** -  
Invoke an external CDECL function with a single string (i.e. char \*) argument.

**ReturnCode = Call\_Real( ExecutableFileName, FunctionName, Argument )** -  
Invoke an external CDECL function with a single real (i.e. double) argument.

## The Search for your DLL or EXE File

The first argument in all PLUS Dynamic Calls is a string naming the executable module (either a .DLL or .EXE file) containing the target function. If the string specifies a path, only that path is searched. When specifying a path, be sure to use backslashes (\), not forward slashes (/). If a path is not specified and the filename extension is omitted, the default library extension .DLL is appended. However, the filename string can include a trailing point character (.) to indicate that the module name has no extension. When no path is specified, the search for the module proceeds in the following sequence:

1. The directory from which the application loaded. i.e. the GPSS World Module Directory.
2. The current directory.
3. The Windows system directory.
4. The Windows directory.
5. The directories that are listed in the PATH environment variable.

The second argument in the Call invocation is used to identify the target function. You must specify the same name exported in the named DLL or EXE file, including appended underscore, if any. If you do not include a required underscore in the function name, the Call Procedure will fail.

### 8.3.7. Probability Distributions

This chapter contains the information you need to use the theoretical probability distributions in the Procedure Library. Empirical distributions, on the other hand, are normally created by using the GPSS FUNCTION Command, using D or C type random Functions. This is discussed in Chapter 6. As other alternatives, it is easy to create your own random variate generators by defining your own PLUS Procedures, or to read empirical data from a file.

Over 20 built in probability distributions are included in the Procedure Library. These distributions are applicable to a wide range of practical situations. [For an excellent detailed discussion, see Law, A.M. and W.D. Kelton: *Simulation Modeling and Analysis*, 2nd Ed., McGraw-Hill, New York (1991)].

Each Procedure call to a probability distribution requires that you specify a *stream* argument. In this position, you are to assert an Expression that evaluates to a Random Number Generator Entity number. Random Number Generator Entities are created, as needed, so you need not predefine them. Several Random Number Generators are used by GPSS World for GENERATE, ADVANCE, and TRANSFER blocks. These are specified in the "Random" page of the Model Settings notebook. This is discussed in Chapter 2.

Most of the probability distributions are specified by parameters that select from a family of functions. Procedure arguments denoted *locate*, *scale*, and *shape* are often used for this purpose. The *locate* argument acts like a post-hoc adder, that is applied after the draw from the distribution. It allows you to move the distribution horizontally, to any position on the x axis. The *scale* and *shape* arguments, if used, select a member from the family of distributions.

The built-in Procedure Library contains the following probability distributions:

◆ Beta

◆ LogLaplace

◆ Binomial

◆ LogLogistic

◆ Discrete Uniform	◆ LogNormal
◆ Exponential	◆ Negative Binomial
◆ Extreme Value A	◆ Normal
◆ Extreme Value B	◆ Pareto
◆ Gamma	◆ Pearson Type V
◆ Geometric	◆ Pearson Type VI
◆ Inverse Gaussian	◆ Poisson
◆ Inverse Weibull	◆ Triangular
◆ Laplace	◆ Uniform
◆ Logistic	◆ Weibull

## 1. Beta

### Syntax

**Real = BETA( Stream, Min, Max, Shape1, Shape2 )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Min**- The smallest sample to be generated. Required. Coerced to real. Must be less than *max*. The argument must be *Expression*.

**Max**- The largest sample to be generated. Required. Coerced to real. Must be greater than *min*. The argument must be *Expression*.

**Shape1** - The first selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

**Shape2** - The second selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

### Return Value

**Real** - The real value generated as a single instance of the probability distribution.

### Probability Density Function

$\lambda_1 = \text{Min}$

$\lambda_2 = \text{Max}$

$\alpha_1 = \text{Shape1}$

$\alpha_2 = \text{Shape2}$

$$f(x) = \frac{(x - \lambda_1)^{\alpha_1 - 1} (1 - (x - \lambda_1))^{\alpha_2 - 1}}{(\lambda_2 - \lambda_1)^{\alpha_1 + \alpha_2 - 1} B(\alpha_1, \alpha_2)}, \quad x \in (\lambda_1, \lambda_2)$$

= 0, *otherwise,*

where  $B(\alpha_1, \alpha_2)$  is the beta function, defined by

$$B(\alpha_1, \alpha_2) = \int_0^1 t^{\alpha_1 - 1} (1 - t)^{\alpha_2 - 1} dt.$$

$$\text{Mean: } \left( \frac{\alpha_1(\lambda_2 - \lambda_1)}{\alpha_1 + \alpha_2} + \lambda_1 \right)$$

$$\text{Variance: } \frac{(\lambda_2 - \lambda_1)^2 (\alpha_1 \alpha_2)}{(\alpha_1 + \alpha_2)^2 (\alpha_1 + \alpha_2 + 1)}$$

Figure 8♦1. The Beta Distribution

## Notes

The Beta Distribution degenerates to the Uniform when the *Shape1* and *Shape2* arguments are 1. That is, Beta( *Stream*, *Min*, *Max*, 1, 1 ) is distributed as Uniform( *Stream*, *Min*, *Max* ).

Beta( *Stream*, *Min*, *Max*, 1, 2 ) is a left triangle; Beta( *Stream*, *Min*, *Max*, 2, 1 ) is a right triangle.

## 2. Binomial

### Syntax

**Integer = BINOMIAL( *Stream*, *TrialCount*, *Probability* )**

### Arguments

***Stream*** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

***TrialCount*** - The number of Bernoulli trials in a sample. Required. Coerced to integer. Must be strictly positive. The argument must be *Expression*.

**Probability** - The success probability of a Bernoulli trial. Must be between 0 and 1. Required. Coerced to real. The argument must be *Expression*.

## Return Value

**Integer** - The integer value generated as a single instance of the probability distribution.

## Probability Mass Function

***t = Trial Count***

***p = Probability***

$$f(x) = \frac{t!}{x!(t-x)!} p^x (1-p)^{t-x}, \quad x \in \{0, 1, \dots, t\}$$
$$= 0, \quad \text{otherwise.}$$

***Mean : t p***

***Variance : t p (1 - p)***

Figure 8♦2. The Binomial Distribution

## 3. Discrete Uniform

### Syntax

***Integer = DUNIFORM( Stream, Min, Max )***

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Min** - The smallest sample to be generated. Required. Coerced to integer. Must be less than or equal to *max*. The argument must be *Expression*.

**Max**- The largest sample to be generated. Required. Coerced to integer. Must be greater than or equal to *min*. The argument must be *Expression*.

## Return Value

**Integer** - The integer value generated as a single instance of the probability distribution.

## Probability Mass Function

$$i = \text{Min}$$

$$j = \text{Max}$$

$$f(x) = \frac{1}{j-i+1}, \quad x \in (i, i+1, \dots, j)$$
$$= 0, \quad \text{otherwise.}$$

$$\text{Mean} : \frac{i+j}{2}$$

$$\text{Variance} : \frac{(j-i+1)^2}{12}$$

Figure 8♦3. The Discrete Uniform Distribution

## 4. Exponential

### Syntax

**Real = EXPONENTIAL( Stream, Locate, Scale )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

## Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$$\beta = \text{Scale}$$

$$\lambda = \text{Locate}$$

$$f(x) = \frac{1}{\beta} e^{-\frac{(x-\lambda)}{\beta}}, \quad \beta \geq 0$$

$$= 0, \quad \text{otherwise.}$$

$$\text{Mean: } \beta + \lambda$$

$$\text{Variance: } \beta^2$$

Figure 8♦4. The Exponential Distribution

## Notes

The Weibull Distribution degenerates to the Exponential when the *shape* argument is 1. That is, Weibull( *Stream*, *Locate*, *Scale*, 1 ) is distributed as Exponential( *Stream*, *Locate*, *Scale* ).

The Gamma Distribution degenerates to the Exponential when the *shape* argument is 1. That is, Gamma( *Stream*, *Locate*, *Scale*, 1 ) is distributed as Exponential( *Stream*, *Locate*, *Scale* ).

## 5. Extreme Value A

### Syntax

**Real = EXTVALA( Stream, Locate, Scale )**

## Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

## Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$$\beta = \text{Scale}$$

$$\lambda = \text{Locate}$$

$$f(x) = \frac{1}{\beta} e^{-\frac{|x-\lambda|}{\beta}} e^{-\frac{(x-\lambda)}{\beta}}, \quad \beta \geq 0$$
$$= 0, \quad \text{otherwise.}$$

Figure 8-5. The Extreme Value A Distribution

## 6. Extreme Value B

### Syntax

**Real = EXTVALB( Stream, Locate, Scale )**

## Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

## Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$$\beta = \text{Scale}$$

$$\lambda = \text{Locate}$$

$$f(x) = \frac{1}{\beta} e^{-\frac{\lambda-x}{\beta}} - e^{-\frac{\lambda-x}{\beta}}, \quad \beta \geq 0$$

$$= 0, \quad \text{otherwise.}$$

Figure 8♦6. The Extreme Value B Distribution

## 7. Gamma

### Syntax

**Real = GAMMA( Stream, Locate, Scale, Shape )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

**Shape** - The selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

## Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$\alpha = \text{Shape}$

$\beta = \text{Scale}$

$\lambda = \text{Locate}$

$$f(x) = \frac{\beta^{-\alpha} (x - \lambda)^{\alpha-1} e^{-\frac{(x-\lambda)}{\beta}}}{\Gamma(\alpha)}, \quad x > \lambda$$
$$= 0, \quad \text{otherwise.}$$

where  $\Gamma(\alpha)$  is the gamma function, defined by

$$\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t} dt.$$

**Mean:**  $\alpha \beta + \lambda$

**Variance:**  $\alpha \beta^2$

Figure 8-7. The Gamma Distribution

## Notes

The Gamma Distribution degenerates to the Exponential when the *shape* argument is 1. That is,  $\text{Gamma}(\text{Stream}, \text{Locate}, \text{Scale}, 1)$  is distributed as  $\text{Exponential}(\text{Stream}, \text{Locate}, \text{Shape})$ .

For positive integer  $m$ ,  $\text{Gamma}(\text{Stream}, 0, \text{Scale}, m)$  is distributed as the  $m$ -Erlang(*Scale*) distribution.

## 8. Geometric

### Syntax

**Integer = GEOMETRIC(Stream, Probability)**

## Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Probability** - The probability of success in each Bernoulli trial. Must be between 0 and 1. Required. Coerced to real. The argument must be *Expression*.

## Return Value

**Integer** - The integer value generated as a single instance of the probability distribution.

## Probability Mass Function

$$p = \text{probability}$$

$$f(x) = p (1 - p)^x, \quad x \in \{0, 1, \dots\}$$
$$= 0, \quad \text{otherwise.}$$

$$\text{Mean: } \frac{1 - p}{p}$$

$$\text{Variance: } \frac{1 - p}{p^2}$$

Figure 8♦8. The Geometric Distribution

## 9. Inverse Gaussian

### Syntax

**Real = INVGAUSS(Stream,Locate,Scale,Shape )**

## Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

**Shape** - The selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

## Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$$\alpha = \text{Shape}$$

$$\beta = \text{Scale}$$

$$\lambda = \text{Locate}$$

$$f(x) = \begin{cases} \sqrt{\frac{\alpha}{2\pi(x-\lambda)^3}} e^{-\frac{\alpha(x-\lambda-\beta)^2}{2\beta^2(x-\lambda)}}, & x > \lambda \\ 0, & \text{otherwise.} \end{cases}$$

Figure 8♦9. The Inverse Gaussian Distribution

## 10. Inverse Weibull

### Syntax

**Real = INVWEIBULL( Stream, Locate, Scale, Shape )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

**Shape** - The selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

## Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$\alpha = \text{Shape}$

$\beta = \text{Scale}$

$\lambda = \text{Locate}$

$$f(x) = \beta \alpha^{-\beta} (x - \lambda)^{-\beta-1} e^{(-\alpha^{-\beta}(x-\lambda)^{-\beta})}, \quad x \geq \lambda, \quad \beta > 0$$
$$= 0, \quad \text{otherwise.}$$

Figure 8♦10. The Inverse Weibull Distribution

## 11. Laplace

### Syntax

**Real = LAPLACE( Stream, Locate, Scale )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

### Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$\alpha = \text{Shape}$

$\beta = \text{Scale}$

$\lambda = \text{Locate}$

$$f(x) = \frac{e^{-\frac{|x-\lambda|}{\beta}}}{2\beta}$$

**Mean** :  $\lambda$

**Variance** :  $2\beta^2$

Figure 8♦11. The Laplace Distribution

## 12. Logistic

### Syntax

**Real** = LOGISTIC( **Stream**, **Locate**, **Scale** )

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

### Return Value

**Real** - The real value generated as a single instance of the probability distribution.

### Probability Density Function

$\beta = \text{Scale}$

$\lambda = \text{Locate}$

$$f(x) = \frac{e^{\frac{(x-\lambda)}{\beta}}}{\beta \left(1 + e^{\frac{(x-\lambda)}{\beta}}\right)^2}$$

**Mean :  $\lambda$**

**Variance :  $\frac{(\pi \beta)^2}{3}$**

Figure 8♦12. The Logistic Distribution

## 13. LogLaplace

### Syntax

**Real = LOGLAPPLACE( Stream, Locate, Scale, Shape )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

**Shape** - The selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

### Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## 14. LogLogistic

### Syntax

**Real = LOGLOGIS( Stream, Locate, Scale, Shape )**

## Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

**Shape** - The selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

## Return Value

**Real** - The real value generated as a single instance of the probability distribution.

# 15. LogNormal

## Syntax

**Real = LOGNORMAL( Stream, Locate, Scale, Shape )**

## Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Required. Coerced to real. The argument must be *Expression*.

**Shape** - The selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

## Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$\sigma$  = Shape, from the Normal Distribution

$\mu$  = Scale, from the Normal Distribution

$\lambda$  = Locate

$$f(x) = \frac{e^{-\frac{(Log[x-\lambda]-\mu)^2}{2\sigma^2}}}{(x-\lambda)\sqrt{2\pi\sigma^2}}, \quad x > \lambda$$
$$= 0, \quad \text{otherwise.}$$

Mean:  $e^{\mu + \frac{\sigma^2}{2}} + \lambda$

Variance:  $e^{2\mu + \sigma^2} (e^{\sigma^2} - 1)$

Figure 8♦15. The Lognormal Distribution

## 16. Negative Binomial

### Syntax

**Integer** = NEGBINOM( *Stream*, *SuccessCount*, *Probability*)

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**SuccessCount** - The number of successful Bernoulli trials required before returning the failure count. Required. Coerced to integer. Must be strictly positive. The argument must be *Expression*.

**Probability** - The success probability of a Bernoulli trial. Must be between 0 and 1. Required. Coerced to real. The argument must be *Expression*.

### Return Value

**Integer** - The integer value generated as a single instance of the probability distribution.

## Probability Mass Function

*s = Success Count*

*p = Probability*

$$f(x) = \frac{(s+x-1)!}{x!(s-1)!} p^s (1-p)^x, \quad x \in (0, 1, \dots)$$
$$= 0, \quad \text{otherwise.}$$

$$\text{Mean : } \frac{s(1-p)}{p}$$

$$\text{Variance : } \frac{s(1-p)}{p^2}$$

Figure 8♦16. The Negative Binomial Distribution

## Notes

The Negative Binomial Distribution degenerates to the Geometric when the *SuccessCount* argument is 1. That is, NegBinom( *Stream*, 1, *Probability* ) is distributed as Geometric( *Stream*, *Probability* ).

## 17. Normal

### Syntax

*Real* = NORMAL( *Stream*, *Mean*, *StdDev* )

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Mean** - The mean value of the distribution. Required. Coerced to real. The argument must be *Expression*.

**StdDev** - The standard deviation of the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

### Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$$\mu = \text{Mean}$$

$$\sigma = \text{Standard Deviation}$$

$$f(x) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}},$$

$$\text{Mean} : \mu$$

$$\text{Variance} : \sigma^2$$

Figure 8♦17. The Normal Distribution

## 18. Pareto

### Syntax

**Real = PARETO( Stream, Locate, Scale )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

### Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$\beta = \text{Scale}$

$\lambda = \text{Locate}$

$$f(x) = \begin{cases} \frac{\beta \lambda^\beta}{x^{\beta+1}}, & x > \lambda \\ 0, & \text{otherwise.} \end{cases}$$

$$\text{Mean: } \frac{\beta \lambda}{\beta - 1}, \quad \beta > 1$$

$$\text{Variance: } \frac{\beta \lambda^2}{(\beta - 2)(\beta - 1)^2}, \quad \beta > 2$$

Figure 8♦18. The Pareto Distribution

## 19. Pearson Type V

### Syntax

**Real = PEARSON5( Stream, Locate, Scale, Shape )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

**Shape** - The selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

### Return Value

**Real** - The real value generated as a single instance of the probability distribution.

### Probability Density Function

$\alpha = \text{Shape}$

$\beta = \text{Scale}$

$\lambda = \text{Locate}$

$$f(x) = \frac{(x - \lambda)^{-(\alpha+1)} e^{-(\frac{\beta}{x-\lambda})}}{\beta^{-\alpha} \Gamma(\alpha)}, \quad x > \lambda$$

$$= 0, \quad \text{otherwise},$$

where  $\Gamma(\alpha)$  is the gamma function, defined by

$$\Gamma(\alpha) = \int_0^{\infty} t^{\alpha-1} e^{-t} dt.$$

$$\text{Mean: } \frac{\beta}{\alpha - 1} + \lambda$$

$$\text{Variance: } \frac{\beta^2}{(\alpha - 1)^2 (\alpha - 2)}$$

Figure 8◆19. The Pearson Type V Distribution

## 20. Pearson Type VI

### Syntax

**Real = PEARSON6( Stream, Locate, Scale, Shape1, Shape2 )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

**Shape1** - The first selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

**Shape2** - The second selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

### Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$\alpha_1 = Shape1$

$\alpha_2 = Shape2$

$\beta = Scale$

$\lambda = Locate$

$$f(x) = \frac{\left(\frac{x-\lambda}{\beta}\right)^{\alpha_1-1}}{\beta \cdot B(\alpha_1, \alpha_2) \left(1 + \frac{x-\lambda}{\beta}\right)^{\alpha_1+\alpha_2}}, x > \lambda$$
$$= 0, \quad \text{otherwise},$$

where  $B(\alpha_1, \alpha_2)$  is the beta function, defined by

$$B(\alpha_1, \alpha_2) = \int_0^1 t^{\alpha_1-1} (1-t)^{\alpha_2-1} dt.$$

Figure 8♦20. The Pearson Type VI Distribution

## 21. Poisson

### Syntax

**Integer = POISSON( Stream, Mean )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Mean** - The mean number of events to occur. Required. Must be strictly positive. Coerced to real. The argument must be *Expression*.

### Return Value

**Integer** - The integer value generated as a single instance of the probability distribution.

### Probability Mass Function

$\lambda = \text{Mean}$

$$f(x) = \frac{\lambda^{-\lambda} \lambda^x}{x!}, \quad x \in \{0, 1, \dots\}$$
$$= 0, \quad \text{otherwise,}$$

$\text{Mean} : \lambda$

$\text{Variance} : \lambda$

Figure 8♦21. The Poisson Distribution

## 22. Triangular

### Syntax

**Real** = TRIANGULAR( *Stream*, *Min*, *Max*, *Mode* )

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Min** - The smallest value to be drawn from the distribution. Must be less than *mode*. Required. Coerced to real. The argument must be *Expression*.

**Max** - The largest value to be drawn from the distribution. Must be greater than *mode*. Required. Coerced to real. The argument must be *Expression*.

**Mode** - The most frequent value of the distribution. Must be greater than *min* and less than *max*. Required. Coerced to real. The argument must be *Expression*.

### Return Value

**Real** - The real value generated as a single instance of the probability distribution.

### Probability Density Function

**a = Min**

**b = Max**

**c = Mode**

$$f(x) = \frac{2(x-a)}{(b-a)(c-a)}, \quad a \leq x \leq c$$

$$= \frac{2(b-x)}{(b-a)(b-c)}, \quad c < x \leq b$$

$$= 0, \quad \text{otherwise.}$$

**Mean :**  $\frac{a+b+c}{3}$

**Variance :**  $\frac{a^2 + b^2 + c^2 - ab - ac - bc}{18}$

Figure 8♦22. The Triangular Distribution

## Notes

"Right" triangular distributions can be generated as Beta distributions.

## 23. Uniform

### Syntax

**Real = UNIFORM( Stream, Min, Max )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Min** - The smallest value to be drawn from the distribution. Must be less than *max*. Required. Coerced to real. The argument must be *Expression*.

**Max** - The largest value to be drawn from the distribution. Must be greater than *min*. Required. Coerced to real. The argument must be *Expression*.

### Return Value

**Real** - The real value generated as a single instance of the probability distribution.

### Probability Density Function

$$a = \text{Min}$$

$$b = \text{Max}$$

$$f(x) = \frac{1}{(b-a)}, \quad a \leq x \leq b$$

$$= 0, \quad \text{otherwise.}$$

$$\text{Mean : } \frac{a+b}{2}$$

$$\text{Variance : } \frac{(b-a)^2}{12}$$

Figure 8♦23. The Uniform Distribution

## Notes

The Beta Distribution degenerates to the Uniform when the *shape* arguments are 1. That is, Beta(*Stream, Min, Max, 1, 1*) is distributed as Uniform(*Stream, Min, Max*).

## 24. Weibull

### Syntax

**Real = WEIBULL( Stream, Locate, Scale, Shape )**

### Arguments

**Stream** - The random number generator entity number. Required. Coerced to integer. Must be greater than or equal to 1. The argument must be *Expression*.

**Locate** - The shift value used to position the distribution. Required. Coerced to real. The argument must be *Expression*.

**Scale** - The compression value used to expand or contract the distribution. Must be strictly positive. Required. Coerced to real. The argument must be *Expression*.

**Shape** - The selection value used to choose from a family of shapes. Required. Coerced to real. Must be strictly positive. The argument must be *Expression*.

### Return Value

**Real** - The real value generated as a single instance of the probability distribution.

## Probability Density Function

$\alpha = \text{Shape}$

$\beta = \text{Scale}$

$\lambda = \text{Locate}$

$$f(x) = \frac{\alpha}{\beta^\alpha} (x - \lambda)^{(\alpha-1)} e^{-(\frac{x-\lambda}{\beta})^\alpha}, \quad x > \lambda$$
$$= 0, \quad \text{otherwise.}$$

$$\text{Mean: } \frac{\beta}{\alpha} \Gamma\left(\frac{1}{\alpha}\right) + \lambda$$

$$\text{Variance: } \frac{\beta^2}{\alpha} \left( 2 \Gamma\left(\frac{2}{\alpha}\right) - \frac{1}{\alpha} \left( \Gamma\left(\frac{1}{\alpha}\right) \right)^2 \right)$$

where  $\Gamma(\alpha)$  is the gamma function, defined by

$$\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t} dt.$$

Figure 8♦24. The Weibull Distribution

## Notes

The Weibull Distribution degenerates to the Exponential when the *shape* argument is 1. That is, `Weibull( stream, Locate, Scale, 1 )` is distributed as `Exponential( Stream, Locate, Scale )`.

`Weibull( Stream, Locate, Scale, 2 )` is known as the Rayleigh distribution.

[\[Table of Contents\]](#)

# Chapter 9 - Advanced Topics

## 9.1. Transaction Chains

Transactions are temporarily bound to other entities by occupying linked lists called chains. Some entities, such as Facilities, have several chains. Other entities have just a single Retry Chain. Each Transaction may be on any number of chains. However, occupying one kind of chain sometimes precludes occupancy by the same Transaction on another. For example, a Transaction on one or more Interrupt Chains cannot be on the Future Events Chain.

A Transaction can be on no more than one of the following chains:

- ◆ Future Events Chain
- ◆ Current Events Chain
- ◆ Facility or Storage Delay Chain
- ◆ Facility Pending Chain
- ◆ User Chain

A Transaction may be waiting for any number of conditions to occur, can be in any number of Transaction Groups, and may be preempted from any number of Facilities at any one time. This means that any single Transaction can be on any number of Interrupt Chains and any number of Group Chains and any number of Retry Chains at the same time.

### Current Events Chain

The Current Events Chain (CEC) is a linked list of ready Transactions which have Blocks yet to be entered before simulated time advances. Although the CEC is kept in priority order, the Active Transaction is usually returned to the CEC ahead of its peers. For this reason, once a Transaction starts to move in the simulation, it tends to keep moving, unless a higher priority Transaction is reactivated.

When the Active Transaction comes to rest on some Transaction Chain, the highest priority Transaction remaining on the CEC becomes the Active Transaction. If the CEC is empty, the most imminent Transaction on the Future Events Chain is moved to the CEC.

### Future Events Chain

The Future Events Chain (FEC) is a time-ordered chain which holds Transactions which must wait for a later simulated time. When all simulation activity for the current clock time is complete, the next Transaction is taken from the FEC. It is this action which causes the system clock to be updated. The value of the system clock is always equal to the scheduled time of the last Transaction to be taken from the FEC.

ADVANCE Blocks and GENERATE Blocks are the only way to place a Transaction on the FEC. These blocks take a time increment as an operand and calculate the absolute time before placing the Transaction on the FEC. When the system clock reaches this absolute time, the Transaction is moved to the CEC so that it may resume its movement in the simulation. In this manner, a duration or inter arrival time can be simulated.

PREEMPT Blocks, and the new DISPLACE Block, can be used to remove Transactions from the FEC. Such Transactions can be rescheduled by entering another ADVANCE Block.

When the scheduler must select a new Active Transaction, if it cannot find one on the CEC, it must take a Transaction from the FEC. This removal of one or more Transactions always causes the system clock to advance. When more than one Transaction resides on the FEC at the next imminent event time, a Time Tie is said to exist. If the Model Settings so dictate, when Time Tied Transactions are moved from the FEC to the CEC, their order is randomized within priorities. This is done to prevent unintentional processing cycles from developing.

A time tie is the occurrence of more than one Transaction with the same time value at the front of the FEC. When a time tie is detected, time tied Transactions are removed from the FEC in random order. For this purpose GPSS World draws pseudo random numbers from the random number generator specified in the Simulate page of the Model Settings Notebook. The removed Transactions are all placed in priority order on the Current Events Chain (CEC). The highest priority Transaction on the CEC then becomes the active Transaction. You can suppress the randomization of time ties by specifying 0 as the associated random number generator.

## Retry Chains

Transactions which fail all tests required for Block entry are placed on a Retry Chain. These tests occur when a Transaction attempts to enter a GATE, TEST, TRANSFER ALL, or TRANSFER BOTH Block. Each entity has a chain of blocked Transactions, called a Retry Chain. Any Transaction on a Retry Chain is waiting for the value of an SNA to change. When the value of the SNA changes, any Transaction on the Retry Chain of the entity is reactivated. This results in replacement on the CEC. When the Transaction becomes the Active Transaction, the specific condition test is repeated. Since this process often uses computer time without advancing Transactions in the model, an injudicious choice of conditions can lead to an inefficient simulation. The power of GATE and TEST blocks must be exercised with caution.

Transactions on Retry Chains are replaced on the CEC by the process of reactivation. This is discussed below. If, on retry, any test is successful, the Transaction enters its next Block. When a Transaction enters a Block it is removed from all Retry Chains automatically.

## Facility Chains

Each Facility Entity has four Transaction chains. They are:

- ◆ PENDING CHAIN - A list of Transactions waiting to PREEMPT the Facility in "Interrupt Mode".
- ◆ INTERRUPT CHAIN - A list of Transactions which have been preempted from ownership of this Facility.
- ◆ DELAY CHAIN - A priority chain of Transactions waiting for ownership of the Facility.
- ◆ RETRY CHAIN - A list of Transactions which are waiting for the status of the Facility to change.

The Pending Chain holds Transactions waiting to enter an Interrupt Mode PREEMPT Block. A Transaction which attempts to enter an Interrupt Mode PREEMPT Block on behalf of a Facility is refused entry to the PREEMPT Block if any preempted Transactions are on the Interrupt Chain of the Facility. A Transaction which is refused entry is placed on the Pending Chain of the Facility. This causes the Transaction to come to rest in the simulation. When a Transaction gives up ownership of the Facility, the first Transaction on the Pending Chain is given ownership and allowed to enter the PREEMPT Block.

The Interrupt Chain is a list of preempted Transactions. When a Transaction enters a PREEMPT Block, and the Facility is currently owned by another Transaction, ownership is given to the new Transaction. The old Transaction is placed on the Interrupt Chain so that its ownership may be restored later. Transactions on one or more Interrupt Chains can still move in the simulation, however, their movement is restricted. Such a Transaction cannot exist on the FEC and cannot leave an ASSEMBLE, GATHER, or MATCH Block where it has been put in a Match Condition. When a Transaction gives up ownership of the Facility, if the Pending Chain is empty, the first Transaction on the Interrupt Chain is given ownership of the Facility.

The Delay Chain holds Transactions waiting for ownership. A Transaction which attempts to enter a SEIZE Block on behalf of a Facility in use is refused entry to the SEIZE Block and is placed on the Delay Chain of the Facility in priority order. Similarly, a Transaction which attempts to enter a Priority Mode PREEMPT Block on behalf of a Facility in use (by a Transaction of equal or higher priority) is refused entry to the PREEMPT Block and is placed on the Delay Chain of the Facility in priority order. This causes the Transaction to come to rest in the active model and a new Active Transaction to be chosen. Then, when a Transaction gives up ownership of the Facility, if the Pending Chain and the Interrupt Chain are empty, the highest priority Transaction on the Delay Chain is given ownership of the Facility.

The Retry Chain is a list of Transactions waiting for a Facility state change. These Transactions are reactivated when the Facility changes from one state to another.

Transactions waiting on a Delay Chain, a Pending Chain, or an Interrupt Chain, or owning a Facility are said to be "in contention" for the Facility. Since a contending Transaction will eventually become the owner of the Facility, contention for a Facility carries the obligation of releasing the Facility. If a Transaction which owns a Facility attempts to leave the simulation by entering a TERMINATE Block or an ASSEMBLE Block, an Error Stop occurs. However, a preempted Transaction is permitted to leave the simulation. Normally, each Transaction remains in contention until it voluntarily enters a RELEASE or RETURN Block on behalf of that Facility. However, PREEMPT and FUNAVAIL blocks have options which can remove other Transactions from contention for a Facility. This removes the

obligation to return ownership as well. In fact, a non-contending Transaction which attempts to enter a RETURN or RELEASE Block will cause an Error Stop.

To summarize, when a Facility is freed by an owning Transaction, pending Interrupt Mode PREEMPTS are first to be given Facility ownership, followed by previously preempted Transactions on the Interrupt Chain, followed by Transactions waiting normally in priority order on the Delay Chain. When a new owner is chosen from the Delay Chain or the Pending Chain, it enters the SEIZE or PREEMPT Block immediately, and then is scheduled by being placed on the CEC behind its priority peers.

## Storage Entity Chains

Each Storage Entity has two Transaction chains. These chains are linked lists of Transactions:

- ◆ DELAY CHAIN - A priority chain of Transactions waiting for storage units.
- ◆ RETRY CHAIN - A list of Transactions which are waiting for the status of the Storage Entity to change.

The Delay Chain holds Transactions waiting for storage units. When a Transaction attempts to enter an ENTER Block on behalf of a Storage Entity, its storage demand is compared to the number of storage units available. The maximum available is defined in a STORAGE Command. If the storage demand cannot be satisfied, the Transaction is refused entry to the ENTER Block and is placed on the Delay Chain of the Storage in priority order. This causes the Transaction to come to rest in the simulation. A new Active Transaction is chosen. Then, when a Transaction gives up storage units, the Delay Chain is scanned in priority order, reactivating Transactions whose storage demands can be satisfied. A "first fit with skip" discipline is used. Each Transaction, in turn, is tested. If its demand can be satisfied it is allowed to enter the ENTER Block and is placed on the CEC behind its priority peers. If its demand cannot be satisfied, it remains on the Storage Entity's Delay Chain.

The Retry Chain is a list of Transactions waiting for a Storage Entity state change. These Transactions are reactivated when the Storage Entity changes from one state to another.

## User Chains

Each Userchain Entity contains a Transaction chain called a User Chain. For a more detailed explanation of the entity type, Userchain, please refer to Chapter 4. Here we discuss the Transaction chain, called the User Chain, which is contained in each Userchain entity.

User chains are linked lists of Transactions which have been removed from the Current Events Chain by a LINK Block. Traditionally, there have been two uses for User Chains.

First, it is possible to implement extremely complex Transaction scheduling disciplines with User Chains. This can be done by assigning a numerical order value to a Transaction Parameter before LINKing the Transaction on a User Chain.

Second, older implementations of GPSS suggest that User Chains be used to avoid scheduling inefficiencies in the GPSS processor. This is less true in GPSS World because blocked Transactions do not remain on the CEC in GPSS World. However, it is still more efficient to avoid testing conditions which cannot possibly result in a successful test. In this case, you can place the blocked Transaction(s) on a User Chain until there is a possibility of success.

## 9.2. The Transaction Scheduler

It is convenient to think of a GPSS simulation as a set of Transactions which occupy Blocks in a Block Diagram. Both the Block Input Window and the Blocks Window are essentially Block Diagrams. At any one time, every Transaction is in exactly one Block, but most Blocks may contain any number of Transactions. Each Transaction, in turn, gets an opportunity to move according to a prescribed path through the Block Diagram. When a Transaction is refused entry to a Block, it must wait in its current Block until conditions become favorable for its movement. The part of GPSS World that is responsible for this movement is called the Transaction Scheduler. Each Block type has its own routine which is executed when a Transaction attempts to enter that Block type. It is the job of the Transaction Scheduler to call the appropriate routine.

The first thing the Transaction Scheduler does is to identify the "Active Transaction". If the CEC is not empty, the highest priority, head-of-line, Transaction on the CEC becomes the Active Transaction. If

the CEC is empty, the Transaction Scheduler replenishes the CEC with the Transaction(s) from the FEC with the lowest time value. This action also updates the system clock.

The Transaction Scheduler then tries to move the Active Transaction as far as it can in the simulation. In effect, the Transaction Scheduler removes the Active Transaction from the CEC, calls the routine for the next sequential Block (NSB), and unless something extraordinary occurs, replaces the Transaction in front of its peers (i.e. same priority) on the CEC. This gives higher priority Transactions a chance to move in the simulation. The CEC replacement can be modified by PRIORITY and BUFFER blocks. After a Transaction enters a BUFFER Block, it is replaced behind its peers on the CEC. BUFFER blocks are useful if a reactivated Transaction must get ahead of the Transaction which reactivated it. Other blocks can interfere with the replacement of a Transaction on the CEC. For example, ADVANCE(+) (i.e. positive time increment) calculates a scheduled time and places the Transaction on the FEC. Other blocks such as LINK, ENTER, SEIZE, and PREEMPT can cause the Active Transaction to come to rest on a Transaction Chain.

Removal from or replacement to the CEC has no effect on the system clock. The simulated time remains the same until there are no Transactions left on the CEC. Continual replacement of the Active Transaction on the CEC gives newly reactivated higher priority Transactions on the CEC a chance to become the Active Transaction. When the Active Transaction comes to rest on a Delay Chain or cannot move because of some other condition, the Transaction Scheduler chooses another Active Transaction and attempts to move it in the simulation.

## The Movement of Transactions

Transactions must be on the CEC in order to move. Even PREEMPTed or DISPLACEd Transaction must become the Active Transaction before they can attempt entry into their new destination Block.

Since a Transaction may be refused entry into a Block, a Transaction scheduling may not lead to a Block entry. For this reason, most simulations have fewer Block entries than Transaction scheduling. On the other hand, EXECUTE blocks can cause additional Block entries.

When the Active Transaction attempts to enter a Block, the Transaction Scheduler calls the Block routine associated with the next Block type. It is the Block routine which decides whether or not the Transaction can enter the Block. Several Block types can refuse to allow the Transaction to enter. These are: ENTER, SEIZE, PREEMPT, GATE, TEST. In addition, if the Transaction has not cleared all its preemptions, it will be refused by ADVANCE(+) Blocks and will not be allowed to leave ASSEMBLE, GATHER, or MATCH Blocks.

When the Active Transaction cannot enter any Block it is said to "come to rest" within the simulation. It is then removed from the CEC and placed on one of the Transaction chains discussed above. Then, a different Transaction is chosen to be the Active Transaction.

## Blocking and Reactivation

The Active Transaction is "blocked" when it must wait for one or more entities to change state. GATE, TEST, TRANSFER BOTH, and TRANSFER ALL Blocks can require that specific conditions be met at one or more entities before the Active Transaction is allowed to proceed in the model. Each entity has a Retry Chain for Transactions which were blocked while trying to enter one of the above GPSS Blocks. When the state of the entity is changed by some other Transaction, all Transactions on the associated Retry Chain are replaced on the Current Events Chain behind their priority peers.

Reactivation is the movement of blocked Transactions to the CEC. If the Active Transaction changes the state of an entity, it is possible that one or more Transactions will be reactivated before the Active Transaction attempts to enter its next Block. If a higher priority Transaction is reactivated, it will become the Active Transaction. If you wish a newly reactivated Transaction to progress immediately, you must either place the active Transaction on the CEC behind its priority peers (BUFFER or PRIORITY Block, BU option), or you must cause the reactivated Transaction to have a higher priority than the old active Transaction. When a reactivated Transaction becomes the Active Transaction, the original blocking test is retried.

A Transaction is not permitted to be blocked on a test which will never be retried. This will lead to an Error Stop.

If an entity state changes more than once before the system clock is updated, some states may not be detected. This can happen if the entity state is changed twice before the suspended blocked Transaction tests the condition. Usually, this possibility can be excluded by careful use of the BUFFER Block.

Do not use TEST or GATE Blocks in Refuse Mode, or TRANSFER (BOTH or ALL) to Block on User Variables. Transactions cannot be Blocked on Named Values because the latter do not have a Retry Chain. If you need to react to values achieved by an integrated variable, you should associate one or two Transaction generation thresholds with the User Variable. You can do this in the INTEGRATE Command. Otherwise, use a Savevalue Entity instead of a User Variable.

## 9.3. Synchronization

### Assembly Sets

An Assembly Set is a collection of Transactions. Transactions in the same Assembly Set are said to be related. When each Transaction is created, it is given an integer denoting its Assembly Set. Transactions created by GENERATE Blocks are given distinct integers starting with 1. Transactions created by SPLIT Blocks are given the Assembly Set of their parent.

A Transaction can change its Assembly Set by entering an ADOPT Block.

Assembly Sets are useful for causing synchronization among Transactions. It is easy to create, wait for, and destroy related Transactions in a simulation. This makes it easy to represent processes which at some point must wait for certain events to occur. The following GPSS blocks are used for that purpose:

- ◆ ADOPT - Set Transaction◆'s Assembly Set.
- ◆ ASSEMBLE - Wait for and destroy related Transactions.
- ◆ GATHER - Wait for related Transactions.
- ◆ MATCH - Wait for related Transaction to reach conjugate MATCH Block.
- ◆ SPLIT - Create related Transactions.

## 9.4. Preemption and Displacement

Preemption is the replacement of one Transaction which owns a Facility by another Transaction. The old Transaction is removed from ownership of the Facility and is placed on the Interrupt Chain of the Facility. The new Transaction becomes the owner of the Facility. Preemption occurs when a Transaction enters a PREEMPT Block, and differs depending on the mode of the PREEMPT Block.

PREEMPT Blocks operate in either "Priority Mode" or "Interrupt Mode". In either case, if a Transaction is preempted, it is placed on the Interrupt Chain of the Facility and ownership is given to the Active Transaction. However, the behavior of the two modes differs when the Active Transaction cannot gain ownership of the Facility. If the Active Transaction attempts to enter a Priority Mode PREEMPT Block, and the Facility is owned by another Transaction of equal or higher priority, the Active Transaction comes to rest on the Facility◆'s Delay Chain FIFO (first in, first out) within priority. If the Active Transaction attempts to enter an Interrupt Mode PREEMPT Block, and there already is a Transaction preempted at the Facility, the Active Transaction comes to rest on the Pending Chain of the Facility.

PREEMPT and DISPLACE Blocks are provided to disrupt service periods. It is common to simulate a service time by having a Transaction SEIZE a Facility and then enter an ADVANCE Block with a positive time argument. When a Transaction is PREEMPTed at any Facility, or if it is DISPLACEd, if it is on the FEC it must be removed. This is done without changing the system clock. Since you may choose to continue a service period where you left off, a residual time is saved when a Transaction is removed from the FEC due to a DISPLACE, PREEMPT or FUNAVAIL Block entry. The residual time is the scheduled Transaction time (BDT) minus the current system clock time. This time is saved automatically and, in addition, it may be stored in a Transaction parameter. When the Transaction regains ownership of all Facilities for which it contends, it may be automatically rescheduled on the FEC using the residual time. If you choose, you may control this process explicitly by the options available in PREEMPT and FUNAVAIL statements.

A preempted Transaction cannot exist on the FEC. If a Transaction on the FEC is preempted, it is removed from the FEC and placed on a Facility◆'s Interrupt Chain. If a preempted Transaction attempts to enter an ADVANCE Block with a positive time increment, it is refused entry. When all preemptions have been cleared for a Transaction, it may then enter an ADVANCE(+) Block. If the PREEMPTed Transaction has been removed from contention for the Facility with the RE option of the PREEMPT Block, the preempted Transaction is not restricted from the FEC.

Preempted Transactions can move through the simulation and can be preempted from any number of Facilities. A Transaction is represented on the Interrupt Chain of each Facility where it has been PREEMPTed. Since a PREEMPTed Transaction may still move through the simulation, a Transaction may be PREEMPTed from any number of Facilities at any one time. However, a Transaction cannot SEIZE or PREEMPT a Facility at which it is currently PREEMPTed.

A Transaction may be displaced from one Block and moved to another. If a Transaction is on the FEC, CEC, a Delay Chain, a Pending Chain, or a User Chain and is PREEMPTed by a PREEMPT Block with a C operand, or is displaced by a DISPLACE Block, it is removed from the original chain, scheduled for a new Block, and placed on the CEC.

A preempted Transaction, which is still in contention for a Facility, cannot enter a TERMINATE Block. Such Transactions must enter a RELEASE or RETURN Block before they are permitted to TERMINATE. Alternately, if you intend to TERMINATE a preempted Transaction, you could remove the Transaction from contention for the Facility using the RE option in the PREEMPT Block.

[\[Table of Contents\]](#)

# Chapter 10 - Performance Tips

This chapter contains a few tips that may enhance the performance of your simulations. Debugging and troubleshooting are treated in Chapter 13.

Simulations always proceed slowly when one or more online windows are open upon them. If speed is important, keep the dynamic windows closed as much as possible.

GPSS World is designed for multitasking operation. However, the message related task switching can sometimes cause window update delays. You can alleviate this by opening fewer online windows, by Halting the simulation before opening multiple views, and by running one simulation at a time.

## 10.1. Memory Allocations

No entity allocations are required by GPSS World. All are automatic. For this reason there is no REALLOCATE statement.

Since GPSS World utilizes virtual memory, you can use more memory for your simulations than exists in your computer. This generally causes extra disk accesses to occur. If you find that your simulations are taking longer to run because of this, you can remedy the situation by adding more physical memory to your computer.

For safety sake there is a limit to the size of virtual memory requests which occur during a simulation. This prevents the inadvertent access to a large amount of virtual memory causing performance degradation. If you need to change this limit, you can do so in the Simulate Page of the Model Settings Notebook.

## 10.2. Identifying Congestion Points

A common cause of performance problems is an unlimited creation of Transactions inside your simulation.

It may be useful to view the simulation through each of the graphics windows. This will often show you the reason that your simulation is growing. Look in the Blocks window for congestion points where Transactions are accumulating. Look at the details view of each of the other entity windows and check the size of the Retry Chain. This indicate the number of Transactions blocked on a state change of each entity.

You may want to produce a Standard Report to study the state of the simulation. To force GPSS World to produce a report, you can simply type:

### REPORT

A buildup of Transactions may be an indication of what would really happen under the conditions you have simulated. For example, if your arrival rates exceed the capacity of the simulated resources, queues will build up indefinitely. In this case, you would expect to run out of memory eventually. Your simulation is telling you that a system so designed would be inadequate to handle the simulated load

## 10.3. Operating Tips.

1. Simulations run MUCH slower when one or more dynamic windows are open. A large number of messages must be sent continually to update the windows. You can speed performance immensely by closing all online windows.
2. Generally, simulations without Data Stream I/O run faster when only one simulation is run at a time. This avoids additional task switching overhead.
3. If performance is important, be sure that you have a hardware math coprocessor capability. You may be able to find a coprocessor which runs on a faster clock than exists on the motherboard of your personal computer.

4. If all these methods fail to improve performance satisfactorily, you should consider a CPU upgrade. Be sure to get hardware math coprocessor capabilities. All GPSS World products can benefit from symmetric multiprocessing. The Intel variants are optimized for Pentium.

## 10.4. Modeling Tips

The following changes to your models may speed up the running of your simulations.

1. If your simulation is behaving in an unexpected manner, you should HALT the simulation and determine if Transactions are building up anywhere. Start with the Blocks Window and look for red Blocks which indicate a blocking condition. More information is available in the Detailed view. Begin STEPPing through the simulation so you can see the dynamics of the Transaction flow. Open the Detailed View of each Window type and look for large Retry Chains. That would indicate an unsatisfied blocking condition.
2. A performance problem can result from retesting blocked Transactions. Several of the most powerful GPSS blocks carry with them the danger that much computer time will be wasted on unsuccessful testing. When a Transaction is blocked, the Simulation Object places it on one or more Retry Chains so that a retest may be scheduled when conditions change. If you use GATE, TEST, TRANSFER BOTH, or TRANSFER ALL blocks and there are large Retry Chains, chances are that a lot of computer time is spent in unsuccessful tests. You should arrange your model to minimize unsuccessful testing in these Blocks.
3. If you have Transactions waiting for a specific condition and there is no chance of a successful test, it is more efficient to place blocked Transactions on a User Chain until there is a possibility of success.
4. GPSS is a powerful simulation language unlike FORTRAN and other programming languages. Transactions do not have to be GENERATEd on every clock tick. GPSS schedules future events and unblocks Transactions automatically, you do not have to keep testing.
5. Replace Refuse Mode GATE and TEST Blocks with alternate constructs. If this is not possible, test conditions which change infrequently.
6. Use a GPSS FUNCTION statement to define probability distributions instead of a complex expression using library functions.
7. COUNT and SELECT blocks can generate a lot of testing. Try to replace these with less powerful Blocks closely matched to your model.
8. You can force the Simulation Object to use a one-day calendar for implementation of the Future Events Chain. This is a performance tuning option that may improved the performance of simulation with a small FEC.
9. You can set aside a block of contiguous Transaction Parameter numbers and cause Simulation Object to allocate them all at once and to access them by direct addressing. This eliminates a serial search for Transaction Parameters. Such allocations are called Parameter Blocks, and are specified in the Simulate Page of the Model Settings Notebook, and are discussed in Chapter 2. You can use the EQU Command to cause named Parameters to reside in a Parameter Block.
10. Data Stream Blocks can be computationally expensive. You can speed processing by using a large amount of data on a small number of text lines. For example, use text lines that contain many data items. You can parse these structures using the string procedures in the Procedure Library.
11. You should keep Transaction Priorities contiguous. Use 1, 2, 3, not 12789, -30977, 22. This speeds up the overhead related to priority queues. Zero is the default.
12. At the expense of some interactivity, the simulation Poll count can be increased to reduce the polling overhead in the Simulation Object. This causes message polling to be less frequent. To do so enter a higher number in the Simulate Page of the Model Settings Notebook.
13. Integrations are generally slow. If you know the analytic solution to the ordinary differential equation, using it in an FVARIABLE, or PLUS Procedure, is much faster than actually playing out the integration numerically using one or more INTEGRATE Statements.
14. A larger Integration Error Tolerance will make integrations run faster, at the expense of some accuracy. The Integration Tolerance is set in the Simulate Page of the Model Settings Notebook. The default is 10-6.
15. Using parts-per-thousand in Fractional Mode TRANSFER blocks instead of a decimal fraction is a little faster.

16. MIN and MAX operators in GPSS Blocks are time consuming. It may be faster to create a PLUS Procedure to select entities.
17. Turn off sections of the Standard Report that you do not need. You can do this in the Report page of the Model Settings Notebook.
18. Real numbers in Numeric Groups are searched much more slowly than integers. If real numbers must be used, consider using a coding algorithm so that only integers are actually used as Group members.

[\[Table of Contents\]](#)

# Chapter 11 - Standard Reports

GPSS World provides for extensive statistics reporting that often gets you the results you need with no additional effort. The final states of all traditional GPSS Entities are reported in a Standard Report that is created automatically when each simulation terminates.

If you have additional reporting needs, you should consult the discussion of Data Streams in Chapter 4.

## 11.1. Report Management

Normally, GPSS World creates a detailed Standard Report of each simulation. Each Report Object is automatically created unless you specify NP as operand B of the START Command. You can also suppress the creation of Standard Reports using Settings. To do so, be sure **Create Standard Reports** is not checked in the Report page of the Model Settings Notebook.

The REPORT and START Commands are discussed in Chapter 6.

### 11.1.1. Report Windows

\You can cause a Report Object to be opened for each newly created Standard Report. To do so, check **Create Standard Reports** and **In Windows** in the Report page of the Model Settings Notebook.

### 11.1.2. Controlling Report Contents

You can select the contents of a model's Standard Reports using a set of checkboxes in the Report page of the Model Settings Notebook. This is opened when you select **Edit / Settings** from the Main Menu. This is discussed in Chapter 2.

Report Windows have full text editing, font, color, and printing capabilities. This is discussed in Chapter 2, under the section entitled, [Text Windows](#).

## 11.2. Sample Report

### 11.2.1. Model

The Model File that follows was used to generate the sample report output in the next section. When this Model File, named SAMPLE9.GPS, is Translated and run, it produces the automatic report named SAMPLE9.001. The model's only purpose is to display all the various graphics and text windows in the Tutorial Manual in Chapter 1, Lesson 11 and to demonstrate the layout of the report and meanings of the various parts of the report in this chapter.

```
; GPSS World Sample File - SAMPLE9.GPS Model to demo graphics windows
Pool STORAGE 400 ;Define Storage
Matrix1 MATRIX ,5,5 ;Define Matrix
Transit TABLE M1,200,200,20 ;Transit time in wait ; and process

GENERATE (Exponential(1,0,100))
JOIN Maingrp ;Xact joins grp called ; Maingrp
JOIN Numgrp,9999 ;Add 9999 to Numeric
SAVEVALUE Addup+,1 ;Total of Transactions
ASSIGN Param_1,232 ;Assign Xact parameter
JOIN Numgrp,P$Param_1 ;Put value in Param1
LOGIC S Switch_1 ;Turn on a logic switch
MSAVEVALUE Matrix1,2,2,QA$Tot_Process
QUEUE Tot_Process ;Queue for process time
SEIZE Facility1 ;Own first Facility
LINK Chain1,FIFO,Nxtblk ;Put on Userchain if busy
Nxtblk SEIZE Facility2 ;Own a second Facility
SEIZE Facility3 ;Own a third Facility
QUEUE Process_Time ;Keep track of process
ADVANCE 100,(Exponential(1,0,100))
```

```

DEPART Process_Time ;Record length
TABULATE Transit ;Add wait + process
RELEASE Facility1 ;Give up 1st Facility
ADVANCE 20 ;Delay time for Fac 2&3
RELEASE Facility2 ;Give up 2nd Facility
ADVANCE 10 ;Extra delay time-Fac 3
RELEASE Facility3 ;Give up 3rd Facility
DEPART Tot_Process ;Leave Queue
UNLINK Chain1,Nxtblk ;Take off waiting Xacts
ENTER Pool,100 ;Place 100 units in the Storage
LOGIC R Switch_1 ;Turn off logic switch
LEAVE Pool,50 ;Take 50 units from Storage
SAVEVALUE Collect-,1 ;Show negative Savevalue
REMOVE Maingrp ;Remove Xact from group
ADVANCE 50,1 ;Wait 50 (+/-1) time units
LEAVE Pool,50 ;Take 50 units from Storage
Finis TERMINATE 1 ;Destroy Xact

```

### 11.2.2. Report

This section contains a standard report generated by the program listed in Section 11.2.1. Each item is explained individually in Section 11.3.

Let's now examine the items included in a standard GPSS World Report. This section includes the complete report resulting from the simulation in Section 11.2.1. The individual items are considered separately in Section 11.3.

GPSS World Simulation Report - SAMPLE9.1.1  
 Tuesday June 6, 2000 14:00:59

START TIME	END TIME	BLOCKS	FACILITIES	STORAGES
0.000	5187.692	32	3	1

NAME	VALUE
ADDUP	10007.000
CHAIN1	10012.000
COLLECT	10017.000
FACILITY1	10011.000
FACILITY2	10014.000
FACILITY3	10015.000
FINIS	32.000
MAINGRP	10005.000
MATRIX1	10003.000
NUMGRP	10006.000
NXTBLK	12.000
PARAM_1	10008.000
POOL	10002.000
PROCESS_TIME	10016.000
SWITCH_1	10009.000
TOT_PROCESS	10010.000
TRANSIT	10004.000

LABEL	LOC	BLOCK	TYPE	ENTRY COUNT	CURRENT COUNT	RETRY
	1	GENERATE		61	1	0
	2	JOIN		60	0	0
	3	JOIN		60	0	0
	4	SAVEVALUE		60	0	0
	5	ASSIGN		60	0	0
	6	JOIN		60	0	0
	7	LOGIC		60	0	0
	8	MSAVEVALUE		60	0	0
	9	QUEUE		60	9	0
	10	SEIZE		51	0	0
	11	LINK		51	0	0
NXTBLK	12	SEIZE		51	0	0
	13	SEIZE		51	0	0
	14	QUEUE		51	0	0

15	ADVANCE	51	1	0
16	DEPART	50	0	0
17	TABULATE	50	0	0
18	RELEASE	50	0	0
19	ADVANCE	50	0	0
20	RELEASE	50	0	0
21	ADVANCE	50	0	0
22	RELEASE	50	0	0
23	DEPART	50	0	0
24	UNLINK	50	0	0
25	ENTER	50	0	0
26	LOGIC	50	0	0
27	LEAVE	50	0	0
28	SAVEVALUE	50	0	0
29	REMOVE	50	0	0
30	ADVANCE	50	0	0
31	LEAVE	50	0	0
FINIS	32	TERMINATE	50	0

FACILITY	ENTRIES	UTIL.	AVE.	TIME	AVAIL.	OWNER	PEND	INTER	RETRY	DELAY
FACILITY1	51	0.937	95.278	1	51	0	0	0	9	
FACILITY2	51	0.853	86.719	1	51	0	0	0	0	
FACILITY3	51	0.949	96.523	1	51	0	0	0	0	

QUEUE	MAX	CONT.	ENTRY	ENTRY(0)	AVE.CONT.	AVE.TIME	AVE.(-0)	RETRY
TOT_PROCESS	11	10	60	0	6.439	556.700	556.700	0
PROCESS_TIME	1	1	51	0	0.660	67.112	67.112	0

STORAGE	CAP.	REM.	MIN.	MAX.	ENTRIES	AVL.	AVE.C.	UTIL.	RETRY	DELAY
POOL	400	400	0	150	5000	1	23.628	0.059	0	0

TABLE	MEAN	STD.DEV.	RANGE	RETRY	FREQUENCY	CUM.%
TRANSIT	553.184	307.992				
			0 - 200.000	8	16.00	
			200.000 - 400.000	10	36.00	
			400.000 - 600.000	7	50.00	
			600.000 - 800.000	12	74.00	
			800.000 - 1000.000	11	96.00	
			1000.000 - 1200.000	2	100.00	

USER	CHAIN	SIZE	RETRY	AVE.CONT	ENTRIES	MAX	AVE.TIME
CHAIN1		0	0	0.277	48	1	29.927

XACT	GROUP	GROUP	SIZE	RETRY
	MAINGRP		10	0

NUMERIC	GROUP	GROUP	SIZE	RETRY
	NUMGRP		2	0

LOGICSWITCH	VALUE	RETRY
SWITCH_1	1	0

SAVEVALUE	RETRY	VALUE
ADDUP	0	60.000
COLLECT	0	-50.000

MATRIX	RETRY	INDICES	NUMERIC	VALUE
MATRIX1	0			

Intermediate Values are Zero.  
 (2,2) 6.413  
 Intermediate Values are Zero.

```
CEC XN PRI      M1      ASSEM CURRENT NEXT PARAMETER VALUE
    61   0 5187.692    61      1      2
```

```
FEC XN PRI BDT      ASSEM CURRENT NEXT PARAMETER VALUE
    51   0 5389.554    51      15     16
                                PARAM_1 232.000
    62   0 5523.253    62      0      1
```

This section explains the items in the standard GPSS World report.

## Title

GPSS World Simulation Report - SAMPLE9.1.1  
 Tuesday June 6, 2000 14:00:59

The title line of the standard report is taken from the name of the Model File that produced the report.  
 The Date and Time of the running of the model is also included.

## General Information

START TIME	END TIME	BLOCKS	FACILITIES	STORAGES
0.000	5187.692	32	3	1

◆ START TIME. The absolute system clock at the beginning of the measurement period. Utilizations and space-time products are based on the START TIME. The START TIME is set equal to the absolute system clock by a RESET or CLEAR statement.

◆ END TIME. The absolute clock time that the termination count became 0.

◆ BLOCKS. The number of Block entities in the simulation at the end of the simulation.

◆ FACILITIES. The number of Facility entities in the simulation at the end of the simulation.

◆ STORAGES. The number of Storage entities in the simulation at the end of the simulation.

## Names

NAME	VALUE
ADDUP	10007.000
CHAIN1	10012.000
COLLECT	10017.000
FACILITY1	10011.000

◆ NAME. User assigned names used in your GPSS World model since the last Translation.

◆ VALUE. The numeric value assigned to the name. System assigned numbers start at 10000.

## Blocks

LABEL	LOC	BLOCK	TYPE	ENTRY COUNT	CURRENT COUNT	RETRY
	1	GENERATE		61	1	0
	2	JOIN		60	0	0
	3	JOIN		60	0	0
	4	SAVEVALUE		60	0	0
	5	ASSIGN		60	0	0

◆ LABEL. Alphanumeric name of this Block if given one.

◆ LOC. Numerical position of this Block in the model. "Location".

- ◆ BLOCK TYPE. The GPSS Block name.
- ◆ ENTRY COUNT. The number of Transactions to enter this Block since the last RESET or CLEAR statement or since the last Translation.
- ◆ CURRENT COUNT. The number of Transactions in this Block at the end of the simulation.
- ◆ RETRY. The number of Transactions waiting for a specific condition depending on the state of this Block entity.

## Facilities

	FACILITY	ENTRIES	UTIL.	AVE.	TIME	AVAIL.	OWNER	PEND	INTER	RETRY
DELAY										
FACILITY1	9	51	0.937	95.278	1	51	0	0	0	0
FACILITY2	0	51	0.853	86.719	1	51	0	0	0	0
FACILITY3	0	51	0.949	96.523	1	51	0	0	0	0

- ◆ FACILITY. Name or number of the Facility entity.
- ◆ ENTRIES. The number of times the Facility was seized or preempted since last RESET or CLEAR command or since the last Translation of the model.
- ◆ UTIL. The fraction of simulated time in the last measurement period that the Facility was owned. A measurement period begins with the Translation of a model or the issuing of a RESET or CLEAR command.
- ◆ AVE. TIME. The average time of ownership by individual Transactions during the measurement period. A measurement period begins with a Translation of the model, or when a RESET or CLEAR command is issued.
- ◆ AVAIL. The availability state of the Facility entity at the end of the simulation. 1 means available, 0 means unavailable.
- ◆ OWNER. The number of the Transaction which owns the Facility entity. 0 means the Facility is not owned.
- ◆ PEND. The number of Transactions waiting to preempt this Facility by entering "Interrupt Mode" PREEMPT blocks.
- ◆ INTER. The number of Transactions currently preempted at this Facility. The count of Transactions on the interrupt chain.
- ◆ RETRY. The number of Transactions waiting for a specific condition depending on the state of this Facility entity.
- ◆ DELAY. The number of Transactions waiting to SEIZE the Facility. This chain also contains Transactions waiting to preempt the Facility in "Priority Mode" PREEMPT blocks.

## Queues

	QUEUE	MAX	CONT.	ENTRY	ENTRY(0)	AVE.CONT.	AVE.TIME	AVE.(-0)
RETRY								
TOT_PROCESS	11	10	60	0	6.439	556.700		
556.700		0						
PROCESS_TIME	1	1	51	0	0.660	67.112		
67.112		0						

- ◆ QUEUE. Name or number of the Queue entity.

- ◆ MAX. The maximum content of the Queue entity during the measurement period. A measurement period begins with the Translation of a model or the issuing of a RESET or CLEAR command.
- ◆ CONT. The current content of the Queue entity at the end of the simulation period.
- ◆ ENTRY. Entry count. The total count of Queue entries during the measurement period.
- ◆ ENTRY(0). "Zero entry" count. The total count of Queue entries with a 0 residence time.
- ◆ AVE.CONT. The time weighted average of the Queue entity content during the measurement period. The space-time product divided by the time duration of the measurement period.
- ◆ AVE.TIME. The average time per unit of Queue content utilized during the measurement period. The space-time product divided by the total entry count.
- ◆ AVE.(-0). The average time per unit of Queue content utilized during the measurement period, adjusted for "zero entries". The space-time product divided by (the total entry count less the zero entry count).
- ◆ RETRY. The number of Transactions waiting for a specific condition depending on the state of this Queue entity.

## Storages

	STORAGE	CAP.	REM.	MIN.	MAX.	ENTRIES	AVL.	AVE.C.	UTIL.	RETRY
DELAY										
POOL	400	400	0	150	5000	1	23.628	0.059	0	0

- ◆ STORAGE. Name or number of the Storage entity.
- ◆ CAP. The Storage capacity of the Storage entity defined in the STORAGE statement.
- ◆ REM. The number of unused Storage units at the end of the simulation.
- ◆ MIN. The minimum number of Storage units in use during the measurement period. A measurement period begins with the Translation of a model or the issuing of a RESET or CLEAR command.
- ◆ MAX. The maximum number of Storage units in use during the measurement period.
- ◆ ENTRIES. The number of "entries" into the Storage entity during the measurement period. The total accumulation of operand B of ENTER statements.
- ◆ AVL. The availability state of the Storage entity at the end of the simulation. 1 means available, 0 means unavailable.
- ◆ AVE.C. The time weighted average of the Storage content during the measurement period. The space-time product divided by the time duration of the measurement period.
- ◆ UTIL. The fraction of the total space-time product of the Storage entity utilized during the measurement period.
- ◆ RETRY. The number of Transactions waiting for a specific condition depending on the state of this Storage entity.
- ◆ DELAY. The number of Transactions waiting to enter ENTER blocks on behalf of this Storage entity.

## Tables and Qtables

TABLE	MEAN	STD. DEV.	RANGE	RETRY	FREQUENCY
CUM.%					
TRANSIT	553.184	307.992			

0 - 200.000	8	16.00
200.000 - 400.000	10	36.00
400.000 - 600.000	7	50.00
600.000 - 800.000	12	74.00
800.000 - 1000.000	11	96.00
1000.000 - 1200.000	2	100.00

◆ TABLE. Name or number of the Table or Qtable entity.

◆ MEAN. The weighted arithmetic average of tabulated values.

◆ STD.DEV. The weighted sample standard deviation of tabulated values.

S.D.= **SQR( (SOS/(COUNT-1)) - (SUM<sub>2</sub>/(COUNT)(COUNT-1)) )**

Where SOS is the accumulated sum-of-squares.

◆ RANGE. The lower and upper limits of the frequency class being reported. Values of the Table argument which are greater than the lower limit and less than or equal to the upper limit cause this frequency class to be updated. The B operand of the TABULATE statement can be used to provide a weighting factor. Frequency classes with accumulations of 0 are not reported.

◆ RETRY. The number of Transactions waiting for a specific condition depending on the state of this Table entity.

◆ FREQUENCY. The total weighted count of tabulated items to fall within this range. The total of all TABULATE B operands.

◆ CUM.% The cumulative frequency count expressed as percent of total count.

## Userchains

USER CHAIN	SIZE	RETRY	AVE.CONT	ENTRIES	MAX	AVE.TIME
CHAIN1	0	0	0.277	48	1	29.927

◆ USER CHAIN. Name or number of the Userchain entity.

◆ SIZE. The number of Transactions on the User Chain at the end of the measurement period.

◆ RETRY. The number of Transactions waiting for a specific condition depending on the state of this Userchain entity.

◆ AVE.CONT. The time weighted average of the User Chain occupancy count during the measurement period. The space-time product divided by the time duration of the measurement period.

◆ ENTRIES. The total number of Transactions placed on the User Chain during the measurement period.

◆ MAX. The maximum number of Transactions on the User Chain during the measurement period.

◆ AVE.TIME. The average time per Transaction on the User Chain during the measurement period. The space-time product divided by the total entry count.

## Transaction Groups

XACT GROUP	GROUP	SIZE	RETRY
	MAINGRP	10	0

◆ XACT GROUP. Name or number of the Transaction Group entity.

◆ GROUP SIZE. The number of Transactions which are members of the group at the end of the simulation.

◆ RETRY. The number of Transactions waiting for a specific condition depending on the state of this Transaction Group entity.

## Numeric Groups

NUMERIC GROUP	GROUP	SIZE	RETRY
	NUMGRP	2	0

- ◆ NUMERIC GROUP. Name or number of the Numeric Group entity.
- ◆ GROUP SIZE. The number of numeric values which are members of the group at the end of the simulation.
- ◆ RETRY. The number of Transactions waiting for a specific condition depending on the state of this Numeric Group entity.

## Logicswitches

LOGICSWITCH	VALUE	RETRY
SWITCH_1	1	0

- ◆ LOGICSWITCH. Name or number of the Logicswitch entity.
- ◆ VALUE. The value of the Logicswitch entity at the end of the simulation. 1 denotes "set" or "true", 0 denotes "reset" or "false".
- ◆ RETRY. The number of Transactions waiting for a specific condition depending on the state of this Logicswitch entity.

## Savevalues

SAVEVALUE	RETRY	VALUE
ADDUP	0	60.000
COLLECT	0	-50.000

- ◆ SAVEVALUE. Name or number of the Savevalue entity.
- ◆ VALUE. The value of the Savevalue entity at the end of the simulation.
- ◆ RETRY. The number of Transactions waiting for a specific condition depending on the state of this Savevalue entity.

## Matrix Entities

MATRIX	RETRY	INDICES	NUMERIC	VALUE
MATRIX1	0			Intermediate Values are Zero.
		(2,2)		6.413
				Intermediate Values are Zero.

- ◆ MATRIX. Name or number of the Matrix entity.
- ◆ RETRY. The number of Transactions waiting for a specific condition depending on the state of this Matrix entity.
- ◆ INDICES. Up to 6 integers that specify the element of the Matrix.
- ◆ NUMERIC VALUE. The value of this Matrix entity element at the end of the simulation. Elements with 0 values are reported in groups.

## The Current Events Chain

CEC	XN	PRI	M1	ASSEM	CURRENT	NEXT	PARAMETER	VALUE
61	0	5187.692	61		1	2		

- ◆ XN. Transaction number of each Transaction on the Current Events Chain

- ◆ PRI. Scheduling priority of the Transaction.
- ◆ M1. Mark time. The time the Transaction, or its earliest ancestor, was GENERATED, or the time the Transaction entered a MARK Block with no operands.
- ◆ ASSEM. The Assembly Set number of the Transaction.
- ◆ CURRENT. The number of the Block where the Transaction existed at the end of the simulation or at the time of this report.
- ◆ NEXT. The number of the next Block scheduled to be entered by the Transaction.
- ◆ PARAMETER. The names or numbers of parameters of the Transaction.
- ◆ VALUE. The value of the parameter.

## The Future Events Chain

FEC	XN	PRI	BDT	ASSEM	CURRENT	NEXT	PARAMETER	VALUE
51	0	5389.554		51	15	16		
							PARAM_1	232.000
62	0	5523.253		62	0	1		

- ◆ XN. Transaction number of each Transaction on the Future Events Chain.
- ◆ PRI. Scheduling priority of the Transaction.
- ◆ BDT. Block departure time. The time of the absolute system clock when the Transaction is scheduled to leave the Future Events Chain.
- ◆ ASSEM. The Assembly Set number of the Transaction.
- ◆ CURRENT. The number of the Block where the Transaction existed at the end of the simulation or at the time of this report.
- ◆ NEXT. The number of the next Block scheduled to be entered by the Transaction.
- ◆ PARAMETER. The names or numbers of parameters of the Transaction.
- ◆ VALUE. The value of the parameter.

[\[Table of Contents\]](#)

# Chapter 12 - GPSS World Statistics

## 12.1. Introduction

GPSS World has a versatile statistics collection capability. For uncomplicated simulations, the statistics in the Standard Report and that reported by the ANOVA library procedure will meet the needs of most users. In these cases, GPSS World will collect and report the simulation statistics automatically. This is discussed in Chapters 11 and 8.

If you need more detailed statistics, the use of variable entities and PLUS Procedures provides excellent control. In these cases, the power of the mathematical library functions is at your disposal. You then define statistics tables in one or more TABLE or QTABLE Statements. The actual recording of data is triggered by one or more TABULATE Blocks in the simulation. Each time a TABULATE is entered, a datum is registered in the table. This can be viewed dynamically in one or more Tables Windows. Alternately, you could record values in Savevalue or Matrix Entities just prior to the completion of the simulation. Tables, Qtables, Savevalues, and Matrices are all reported automatically in the Standard Report, if you have checked them in the Report Page of the Model Settings Notebook.

QTABLE Commands provide an easier way to collect statistics based on Queue Entities. In this case, the tabulations are triggered by entry into DEPART Blocks instead of TABULATE Blocks.

Normally, you should arrange your simulation so that control values are taken from User Variables and the results are in User Variables when the simulation terminates. The reason for this is that this is how the PLUS Experiments created by the Automatic Experiment Generators work. If there is any chance you will want to use them, you will need to make these accommodations.

For more in-depth analysis, you can accumulate simulation results in a Result File by using Open, Close, Read, Write, Seek in the form of GPSS Blocks or procedure calls. This can accumulate simulation results in an historical database that can be used with advanced third party statistical analysis software. For analysis of data in a Result Matrix, the built in Multiway ANOVA Procedure is available.

## 12.2 ANOVA

If you are new to simulation, you will notice that when you repeat a simulation in the same session you may get different results due to random effects. Such effects must be carefully distinguished from the real effects caused by new designs in your simulated systems. The ANOVA library procedure provides you with a simple and yet powerful technique for establishing that your results do or do not emerge above the noise level caused by randomness.

In order to measure the amount of random noise, you must repeat your simulations changing nothing except the random number seeds. These repeat runs are called replications and are important for measuring the statistical noise, i.e. the Standard Error, in your experiment. In the simplest scheme, each different design you choose to simulate would be replicated several times. Then the ANOVA library procedure will use the data to detect if the effects of changing the factors of your experiment were significantly greater than the statistical noise level. The results of this analysis will appear automatically in the Simulation Object's Journal Window.

The ANOVA Procedure provides for push button data analysis. It calculates Confidence Intervals and an Analysis of Variance of simulation results. Normally, during an experiment, you fill a special global Matrix Entity called a Result Matrix with the results of the simulations and then pass it as an argument to the ANOVA Procedure. The details of the use of GPSS World's Multiway ANOVA library procedure are discussed in detail in the next chapter in Section 13.5.3.

## 12.3. RESET

If you wish to exclude the effects of starting conditions from your final simulation, you should use the RESET Command to begin the measurement period after transient effects have disappeared. The RESET Command is discussed in Chapter 6. Plot Windows are useful for observing the convergence of a simulation to steady state conditions.

## 12.4. Space-Time Products

A space-time product is a time duration multiplied by a count. Space-time products are used in the calculation of several kinds of useful statistics. For example, to calculate the wages due to a labor force you could use the number of people who worked each given number of hours. To calculate the total hours worked, you could multiply the hours times the number of people to form a space-time product for each number of hours worked. You could then add up all the space time products in order to calculate the total person-hours. Similarly, many statistics reported in the Standard Reports are accumulations of space-time products.

Facility, Queue, Storage, and Userchain entities have SNAs which require space-time products in their evaluation. For example, the average storage in use is a time weighted average which gives more weight to those numbers which have the longest durations. The accumulated space-time product is divided by the simulated time duration in order to calculate the average storage in use.

Facility, Queue and Userchain entities also have space-time products. They are used to calculate average content via a corresponding SNA. The Facility space-time product is the accumulation of total busy time. The other space-time products are equal to the area under the graph of Queue content or Userchain content.

The RESET command may be used to begin a new measurement period. A RESET command sets the space-time product to zero and the total count equal to the current count. This allows new space-time products to be accumulated. In the SNAs based on space-time product calculations, this method of initialization may introduce a small bias on the low side.

The following table shows the SNAs calculated from space-time products.

**Table 12◆5. Space-Time System Numeric Attributes**

SNA ENTITY	CALCULATION
FR UTILIZATION (ppt)	Facility 1000 times (Space-time product) divided by (Total Simulated Time)
FT AVERAGE HOLDING TIME PER CAPTURE	Facility (Space-time-product) divided by (Total capture count)
QA AVERAGE CONTENT	Queue (Space-time-product) divided by (Elapsed time)
QT AVERAGE RESIDENCE TIME	Queue (Space-time-product) divided by (Total count)
QX AVERAGE RESIDENCE TIME EXCLUDING ZEROS	Queue (Space-time product) divided by (Total Count) - (Count finding zero)
SA AVERAGE CONTENT	Storage (Space-time-product) divided by (Elapsed time)
SR FRACTIONAL UTILIZATION	Storage 1000 times (Space-time product) divided by (Capacity) times (Elapsed time)
ST AVERAGE HOLDING TIME	Storage (Space-time-product) divided by (Sum of storage requirements)
CA AVERAGE CONTENT	Userchain (Space-time-product) divided by (Elapsed time)

CT AVERAGE RESIDENCE  
TIME

Userchain (Space-time-product) divided by  
(Total count)

[\[Table of Contents\]](#)

# Chapter 13 - Experimentation

This chapter tells you all about the experimentation features in GPSS World, and how to use them. A general overview is presented in the chapter introduction, followed by background material, then a detailed presentation of all the features relating to experimentation. We conclude the chapter with a list of practical tips that address problems you may encounter.

You should be able to get started after reading the introduction if you are willing to use the Help facilities, the Tutorial Manual, and refer to details in this manual. If you are unfamiliar with some of the nomenclature you may want to read section 13.2.2 first.

However, to get to most out of what GPSS World offers you need to become familiar with what lies in the rest of this chapter. Also helpful are several new sample models including OneWay.gps, Multiway.gps, ExperEther.gps, LatinSquare.gps, GraecoLatin.gps and Lessons 19 and 20 in the Tutorial Manual.

## 13.1 Introduction

The experimental phase of a simulation project assumes the existence of a fully developed, well-tested, GPSS World simulation that embodies the influences of each of the factors whose effects are to be measured. Only then, should you proceed with the experimental phase of your project. Conceptually, GPSS World supports three different approaches to experimentation: Screening Experiments, User Experiments, and Optimizing Experiments. There are features that support each of these.

The Experiments generated by GPSS World use partial factorial experiments with a fixed number of treatment levels for each factor. When you create your own experiments, you do not have this restriction. You are limited to 6 factors, each of which may have any number of treatment levels. Your job then is to define a one-way or an orthogonal experiment, fill a Result Matrix with the yield of each run, and pass it to the ANOVA library procedure.

### 13.1.1 Screening Experiments

A Screening Experiment is usually used to identify the most important factors affecting the simulated system. This information is crucial for directing the rest of the investigation in the most efficient way. The results of a screening experiment show which factors are not effective and should receive low priority with respect to further study. Also, the sensitivity of the yields of an experiment to one or more of the screened factors raises a flag that assumptions made in this part of the simulation should be verified carefully. GPSS World has an automatic experiment generator that can create Screening Experiments for you. To use it, you fill in a dialog that is accessed from the Edit Menu of the Main Window. This results in PLUS code being inserted into your Model Object. Optionally, this process will also load a Function Key with the appropriate CONDUCT Command for the simulation. After that, your role is to create the Simulation Object (Ctrl+Alt+S), start the experiment (normally F11), and then analyze the results.

### 13.1.2 User Experiments

User Experiments are more flexible, but you must create and run them yourself. Even so, GPSS World provides a lot of support along the way. The ultimate goal is for you to provide the data needed by the GPSS World ANOVA library procedure, which will analyze your experiments with up to 6 factors, including 2 and 3-way interactions between factors. The main requirement of GPSS World ANOVA is that you must pass it the name of the GPSS Matrix where the results of your experiment have been saved.

There are two important things to know about Result Matrices. First, before you begin your experiment you should initialize your Matrix's elements to the UNSPECIFIED state. Here's a typical Statement to do this:

```
INITIAL MyResultMatrix,UNSPECIFIED
```

This makes it clear to the ANOVA routine when a run of the experiment has not been completed.

The second thing is that if your experiment has more than one factor, you need to make it symmetrical so that the GPSS World ANOVA routine can cleanly analyze the variance. The technical term for this special symmetry is "orthogonality". All you have to do to achieve orthogonality is to look at all pairs of factors in your experiment. If, within each pair, each Treatment Level of the first Factor appears the same number of times within each Treatment Level of the second Factor, the experiment will be orthogonal. If you make everything symmetrical, it will be orthogonal. Don't worry, GPSS World will tell you when it's not.

GPSS World provides the PLUS Language that you can use to write programmable experiments. A good way to get started is to look at chapters 19 and 20 in the Tutorial Manual, or to use the automatic experiment generators in the Edit Menu to create samples for you to follow. Actually, you don't really need to use PLUS. You can set up your own Command lists in Include-files or enter them manually. All you need is some way to get the yield(s) of each run in your experiment into one or more Result Matrices that can be passed to the ANOVA library procedure.

If you do decide to write your experiment in PLUS, you should consider using the same programming style as the automatic experiment generators, which is discussed later in this chapter. The use of a Run Procedure is a case in point. A Run Procedure is a simple PLUS Procedure that is called every time a run in the experiment is to be performed. Since the Run Procedure is called by a PLUS Experiment, it is permitted to invoke the DoCommand library procedure, giving it the power to issue RMULT, START, RESET, and other Commands. It is then convenient to place all the Commands needed to set up each run into the Run Procedure. To reset variables in the Run Procedure, you should use the CLEAR ,OFF form of the CLEAR Command so that your Result Matrix is preserved. To vary the Random Number seeds, generated experiments pass a unique run number as an argument to the Run Procedure. However, you may prefer to do this in some other way. One last hint: since the command string passed to the DoCommand library procedure is eventually Translated in global scope, you must avoid using local argument and temporary variable names in it. If you need to use values from arguments or temporary variables, just build the command string dynamically before invoking DoCommand, as is done in the generated experiments. This is discussed further, below, in Section 13.3.2.

### 13.1.3 Optimizing Experiments

Optimization and the quantitative prediction of the behavior of a system are often the primary objectives of a simulation project. Both of these are supported directly by GPSS World. A Response Surface is an equation that predicts the results of a simulation. It is often desirable to establish a Response Surface for abbreviating results, providing a predictive methodology, quantifying the sensitivity of results to numerical inputs and other assumptions, and determining optimal treatment levels. GPSS World's programmable Experiment feature provides the basis for a variety of Response Surface methodologies. A PLUS Experiment that calculates a Response Surface can be generated automatically, based on the input from a User Dialog. The resulting Experiment attempts to use the Method of Steepest Ascent and a Method of Local Exploration to find an optimum value. If successful, it reports the mathematical description of the best fit Response Surface and the predicted optimum conditions in the Journal Window of the Simulation Object.

As with Screening Experiments, the automatic generation of an Optimizing Experiment begins in the Edit Menu of the Main Window of GPSS World. When a Model Object is selected, you can click on Edit / Insert Experiment / Optimizing ... in order to open the dialog. After you fill in the blanks, and edit the Run Procedure, GPSS World inserts the finished PLUS Experiment into your model and optionally loads the appropriate CONDUCT Command into the F12 Function Key. Then, all you must do to start the experiment is to Create the Simulation Object (Ctrl+Alt+S) and press F12.

## 13.2 Experimentation and The Analysis of Variance

Next we explore some of the technical considerations that are relevant to experimentation in GPSS World. This section presents a discussion of the main issues that need to be considered when you invoke the ANOVA Procedure or when you generate an Experiment.

### 13.2.1 Motivation

Simulations are not completely faithful to the real world system they are intended to represent. Instead, they are merely representations intended to capture the most important behavioral characteristics of the target system.

One major difference with the real world is that simulations are perfectly repeatable. That is, if we repeat a simulation many times, we get precisely the same results every time. Such a thing is extremely unlikely in the natural sciences. In fact, it is so unnatural that we intentionally introduce variability into our simulations in order to make them look more realistic. In practice, we often find simulations in which the random variation in modeled processes, artificially introduced by us, is essential for capturing the behavior of the target system.

Another source of variability that does not occur in simulations is that due to the measurement itself. Whereas in the real world some measurement tools are very noisy and troublesome, in the simulated environment we enjoy a god-like perspective where all things are ultimately knowable without disturbing the measured environment. The point is that the observations from simulations are unnaturally crisp, and may occasionally reveal effects that exist in the real world but are extremely difficult to observe there.

Unlike experiments conducted in the real world, in simulation we have much better control over the variability introduced to mimic the apparent randomness of repeated real world measurements. Generally, we introduce one or more streams of "pseudo" random numbers, which are able to pass certain statistical tests of randomness. We go even further when we select probability distributions thought to accurately reflect the target system's behavior. GPSS World provides over twenty of these, to be selected and used by the simulation analyst. In our quantitative analysis of simulation results, we will assume we have successfully modeled the random variance of measured values in the real world in this way.

The Analysis of Variance, or ANOVA, is a highly developed methodology for extracting information from the results of an experiment. In essence, it breaks up the variance of observations, and associates the pieces with the experimental factors and their interactions. One part, called the Error term, is associated with the unavoidable intrinsic randomness of the observations. In one sense, ANOVA attempts to account for all the variation of the observations from their average. That which is left over, the "Error", is a variation that has not been accounted for by association with the experimental conditions. It is an estimate of the intrinsic variability of observations called the Standard Error. Usually, only those effects that exceed the magnitude of the Standard Error are presumed to be real and not due to random fluctuations. A test using the "F Statistic" is normally used to ascertain this.

We must be careful to distinguish the variance introduced for realism from the variance encountered in our statistical models, the Standard Error. Generally, we do not want to distort the former because it will often cause our simulation to miss important real world behavior. On the other hand, the purpose of Design of Experiments is to reduce the unaccounted variance in our statistical analysis. This is quite a different matter. Variance Reduction Techniques, which distort variance introduced for realism, are to be avoided in the methodology we are presenting here. Our goal is to model real-world randomness accurately, not to reduce it.

We are, however, motivated to find ways to reduce the variability of the observations in the Analysis of Variance, as long as we do not distort the way we represent natural randomness. Reducing the estimate of the Standard Error in an Analysis of Variance allows significant effects to emerge above the statistical noise level. Two methods of doing so, which we will discuss below, improve the residual data used in the estimate of the Standard Error. The first by increasing the number of observations, the second by changing the underlying statistical model. These are discussed in Section 13.2.3.

### 13.2.2 Nomenclature

When we conduct an experiment we begin by selecting one or more metrics that quantify the state of the system or some other outcome of interest. Measurements of these quantities are called "observations" or "yields", and the set of all observations comprises the "results" of the experiment.

We will examine one or more forces called "factors" that are believed to influence the value of some of the observations. We will assign values called "treatment levels" to the factors when we specify the conditions for each execution of the simulation. When there are multiple factors, we say that the conditions are specified by "treatment combinations", since multiple treatment levels must be specified. If the influence of some factor differs when the treatment level of some other factor is varied, we say that there is an "interaction" between the two factors. Since we will be using an additive model, when the effects of two factors together is not the sum of their separate effects, we say that there is an interaction between them. There can be distinct interactions involving any number of factors.

We will simulate the target environment multiple times calling each instance a "run". The treatment combination specifies the conditions of the run, and one or more observations form the results of the run. When the conditions of a set of runs are the same except for randomization, we say that the runs in the set are "replicates", and form a "cell" of the experiment.

In the course of the Analysis of Variance, done for you by GPSS World, the observations are partitioned into components, called "effects" which are presumed to be due to the influence of the factors and their interactions. How this partition is done depends on an underlying additive model called the "statistical model". An intrinsic random deviation causes the observations to differ slightly from the sum of effects in the statistical model. The "error term" in an observation is derived by subtracting estimates of the effects corresponding to terms in the statistical model. Since the error is the quantity left over, it is often called the "residual".

For multiway experiments, that is those with more than one factor, the Analysis of Variance in GPSS World requires that the experiment be orthogonal in order to complete the analysis. This means that the estimators within the analysis must be uncorrelated. In practice, providing the same number of runs within each treatment combination of a balanced design guarantees orthogonality.

Effects, as denoted by letter groups, have important uses in the design of fractional factorial experiments, discussed below. To fractionate a factorial design, a small number of effects are chosen by the designer as "Generators" in order to specify the set of runs in the experiment. Unfortunately, this also causes some effects to become indistinguishable, or "aliased", with others. In GPSS World, the set of generators separated by equal (=) signs is called the "defining relation" of the experiment.

GPSS World supports experimentation through internal library procedures and through PLUS, the embedded Programming Language Under Simulation. PLUS is both a low-level procedural language accessible from within simulations, and a high-level control language that can direct the conditions and sequence of runs in an experiment.

### 13.2.3 ANOVA

The Analysis of Variance is a tool, pioneered by Sir Ronald Fisher, which is able to extract much of the information available in a set of measurements. In it, we quantify the variation of observations from the overall average, and then break it into pieces, each of which has a separate cause. If any experimental factor cannot be found to induce variability in a measurement, we say that it does not have a significant effect on it. On the other hand, if a factor does appear to induce variability, we compare the amount of it to an estimate of the intrinsic variability of the observation, the Standard Error. We do this to rule out apparent effects that are nothing more than random fluctuation. Our standard of comparison is that the variation from any source must be much larger than the Standard Error in order to be deemed a significant effect. The F test, named for Fisher, is used for this purpose. We use the F test as the criterion by which we declare the effects of experimental factors and their interactions to be statistically significant.

Implicit in the use of ANOVA is the existence of an additive mathematical model used to explain the components of variation in the observations. We will call this the "statistical model". The simplest statistical model is given in Figure 13-1.

$$y_i = m + e_i$$

*Figure 13-1. A Simple Statistical Model*

In this example, each observation is broken down into only two components, the grand mean of all observations,  $m$ , and the random component,  $e$ . Each observation has a starting point, the grand population mean, and then incurs a random deviation leading to its final value. The grand mean does not vary from observation to observation, whereas the random component does, and is subscripted accordingly. Although this model is suggestive, it allots all variation of the observations to random sources and none to factors, and so is not very useful in analyzing the results of an experiment.

Next, we turn to a statistical model used to analyze the data from an experiment with a single factor, namely, factor A.

$$y_{i,j} = m + a_i + e_{i,j}$$

*Figure 13-2. One Factor Statistical Model*

In Figure 13-2, notice the introduction of the subscripted alpha term, which denotes the effect of the  $i$ th treatment level of the one and only factor considered in the model. The experiment would include one or more runs at each of the treatment levels of factor A. All observations at a given treatment level are analyzed using the same value for  $a$ . Since there is only one factor in this experiment, the number of treatment combinations is just the number of treatment levels of that factor. An Analysis of Variance based on this statistical model will result in an ANOVA table that partitions the variation of the observations into that due to treatment A, and that due to the random variation.

$$y_{i,j,k} = m + a_i + b_j + i(ab)_{i,j} + e_{i,j,k}$$

*Figure 13-3. Two Factor Statistical Model*

In Figure 13-3, we move to a model where two factors are considered. Notice that we now include a term for the interaction between factor A and factor B, denoted  $i(ab)$ . This term may differ at each treatment combination, and therefore is subscripted twice since there are two factors. Recall from above that when the effects of two factors together is not the sum of their separate effects, there is an interaction between them. The strength of the interaction between factors A and B is denoted "AB" and would be reported in a line of the ANOVA table.

In a complete factorial experiment all interaction terms would be included. Just as the complete 2-factor model in Figure 13-3 has 5 (RHS) terms, a complete 3-factor model has 8 terms, and a complete 4-factor model has 16 terms. All this information is normally presented in the resulting ANOVA table. GPSS World can handle up to 6-factor models with up to and including 3-way interactions.

Figure 13-4, below, presents an ANOVA table reported by GPSS World. As explained above, the single capital letters denote factors and capital letter combinations denote interactions.

First, look at the bottom of the table. The Total Sum of Squares is to be partitioned, and the components are to be associated with the effects of factors and their interactions. Anything that is left over, that is, the residual sum of squares, is shown in the previous line, labeled "Error". The Mean Sum of Squares of the error term is used to estimate the Standard Error of the experiment.

Each Sum of Squares has a divisor associated with it called the Degrees of Freedom. From statistical considerations, the Degrees of Freedom is that divisor that must be used to create an unbiased estimator of the Standard Error, in the absence of other effects. For our purposes, it is sufficient to think of Degrees of Freedom as the proper divisor associated with a Sum of Squares in the ANOVA table. GPSS World will always calculate the Degrees of Freedom for you.

ANOVA						
Source of Variance	Sum of Squares	Degrees of Freedom	Mean Square	F	Critical Value of F (p=.05)	
A	28.000	2	14.000	5.600	3.89	
B	21.000	3	7.000	2.800	3.49	
AB	11.000	6	1.833	0.733	3.00	
Error	30.000	12	2.500			
Total	90.000	23				

Treatment Level A B	Count	Mean	Minimum	Maximum	95% C.I. (SE)
1 1	2	3.000	2.000	4.000	[ 0.564, 5.436 ]
1 2	2	1.000	1.000	1.000	[-1.436, 3.436 ]

01/27/01 13:26:44	1.5811388
-------------------	-----------

*Figure 13-4. An ANOVA Table*

Each factor and interaction in the statistical model is represented by a distinct line in the ANOVA table. In each line we have first the Sum of Squares and the Degrees of Freedom associated with that estimate. These are the basics from which the other numbers are derived. A simple division

results in the Mean Square, and by dividing that quotient by the Mean Square of the Error, from the bottom row of the table, we get the F statistic for that effect.

Now we are ready to draw some conclusions. We must decide if the F value is large enough to declare that the effect is significant. The threshold value we will use to make the comparison is called the "Critical Value of F" and is placed just to the right of our F statistic on the same line. If our F value exceeds the Critical Value, we conclude that we are dealing with a significant effect. If not, we conclude that the effect is not significant and we disregard any associated variation in observations as only due to random noise. The larger the F value the stronger the effect. The ANOVA table in Figure 13-4 shows the effect of factor A to be significant and the effects from factor B and the AB interaction to be not significant.

Sometimes an experiment will fail to detect an effect even though one actually exists. One of our goals is to make this as unlikely as possible. According to the ANOVA table, there are two ways to make an experiment better able to detect real effects. To get more positive results we would need either a larger F statistic or a smaller Critical Value of F.

It is desirable to remove any part of the Sum of Squares of the Error that is due to any important effect we have not included in the analysis. If we can do this, our F statistic will generally be larger. In experimentation in the natural sciences, this is done by making comparisons in as homogenous an environment as possible, a technique known as "blocking". However, in simulation studies this goal is perhaps best approached by identifying additional factors that must be included in the experiment.

Two other approaches are directed at increasing the Degrees of Freedom of the Error term. The first is simply to increase the number of replicates in the experiment. This is usually the most expensive approach but it can be quite effective. The second possibility relates to the design of the experiment and the statistical model behind the Analysis of Variance. The Mean Square of the Error is actually a residual term left after all other squares have been removed. If we can find an acceptable way to allow more of the data to remain after the effects are removed, we will have an estimate of the Standard Error that has more degrees of freedom. The resulting Critical Value of F will be smaller, thereby increasing the power of the analysis. This is what we are attempting to do when we choose to ignore some of the interactions.

None of the techniques we are considering will distort the randomness introduced to simulate real-world randomness. Variance Reduction Techniques that do should not be used here. Intentionally reducing the intrinsic variability of the observations would cause the F statistics to be overstated.

In many multifactor experiments we will chose to remove the highest order interactions in order to improve the information available on the main factors and the low order interactions. You can do this using the third argument of the ANOVA Procedure. If it is true that these interaction effects do not exist, this will allow GPSS World to use the extra degrees of freedom to achieve a better estimate of the F statistic. In addition, more Degrees of Freedom means a smaller critical value of F, as well. However, we must realize that removing terms from the statistical model assumes that there is no high-order effect. If there is a reasonable chance of one, we would be better off adding replicates to improve the statistics rather than limiting the analysis to low order interactions. Replicates are made known to the ANOVA Procedure in its second argument.

$$y_{i,j,k} = m + a_i + b_j + e_{i,j,k}$$

Figure 13-5. Two Factor Statistical Model without Interactions

Figure 13-5 shows a statistical model in which the 2-way interaction term was removed. If it is true that the interaction effect does not exist, we have improved the crispness of the Analysis of Variance by providing more Degrees of Freedom for the Mean Square of the Error. When you choose to apply this technique, you simply change the 3rd argument of the call to the GPSS World ANOVA library procedure to remove higher order interactions from the analysis. This built-in procedure can handle up to 3-way interactions, and allows you to give up 2-way or 3-way interactions, and above, to improve the significance testing of the other effects.

### 13.2.4 Selecting Factors

When you use the ANOVA library procedure you must define a GPSS Matrix Entity, called the Result Matrix, to store the individual results of each run. If you were analyzing more than one metric, you would have more than one Result Matrix. Since GPSS World Matrix Entities can have up to 6 dimensions, each of any size, the Result Matrix is also limited to 6 dimensions.

When you write your own GPSS World Experiment, the most important decision you will make is to select the factors of the experiment. These are the quantities that you can control in order to optimize your system. Each factor is to be represented by a User Variable that takes on treatment levels as

values. After you run each simulation in the experiment, you must place the resulting value into a Results Matrix so that the data can be analyzed by the ANOVA Procedure.

The position in the Result Matrix for each result is determined by the treatment combination. For example, if the experiment considers 4 machine types and two speeds of each, the result of simulating the third machine at high speed would go into the Result Matrix at position [3,2]. Actually, since replicates are usually tagged onto the end, the first run at this treatment combination would go into element [3,2,1] of the Result Matrix. Each dimension of the Result Matrix is analyzed as a factor by the ANOVA library procedure. The size of the dimension in the Result Matrix must be at least as large as the number the treatment levels of that factor, or of the maximum replicate count.. There is no arbitrary limit to the number of treatment levels within a factor, only that imposed by the virtual memory of your computer.

We can examine any effect by giving it its own dimension in the Result Matrix. If we so desire, we could even separate random number streams and treat them individually as distinct factors. Each random number seed would represent a distinct treatment level. This would provide information on the relative importance of each random input process. Usually, though, we use a single dimension for replicates, and vary all seeds from one replicate to the next within a single treatment combination. We then make known to the ANOVA Procedure which dimension is used for replicates. This is done in the second argument.

Not all experiments need a Replicate Dimension. The ANOVA is often able to extract an estimate of the Standard Error without one. The Latin Square sample model, "Latin Square.gps" is an example of this. However, depending on the design of the experiment, sometimes there is not enough residual data to estimate the Standard Error. Then the Analysis of Variance will not be able to calculate F statistics for the ANOVA Table. In such cases, you will have to either add replicate runs to the experiment or remove high level interactions from the statistical model in order to provide enough residual data for the estimate of the Standard Error. Both of these methods require a change in the call to the ANOVA library procedure.

### 13.2.5 The Random Number Streams

A guideline often used in the design of experiments is this: control all factors that you can; what you can't control, block; what you can't block, randomize. This applies to the use of the random streams in a discrete event simulation, as well.

To treat random number streams as factors is to allocate a dimension for each stream, and to use each distinct seed as a treatment level. Although it is possible to view the random number streams as factors in this way, and to examine their relative effects, it may be best done as a separate study. For the mainstream investigation of the main effects and interactions, we are more interested in using our random number streams to represent sources of natural and unavoidable variability, reserving the dimensions of the Result Matrix for the factors being studied.

The most common approach combines all random number streams into a single pseudo-factor, where each treatment level is a set of distinct seeds for all random number streams. A single dimension, the Replicate Dimension, is then allocated from the Result Matrix for this purpose and is made known to the ANOVA library procedure. This has some similarities with blocking, where all comparisons are performed within each homogeneous environment. The ANOVA library procedure can then treat this Replicate Dimension separately to better estimate the Standard Error of the experiment. The experiment then consists of an experimental "cell" for each treatment combination, with several replicate runs within each cell.

If these approaches are too expensive, or there isn't a dimension to spare in the Result Matrix, the runs should be "completely randomized". This means that randomization should not be restricted in any way, and that the random number seeds should not be repeated within a single random number stream. If this procedure fails to provide enough Degrees of Freedom for the analysis, you may need to limit the level of interactions to be included in the statistical model, as well.

## 13.3 GPSS World Features

We turn now to the specific features of GPSS World which can be used in any of the experimentation phases of a simulation project. First, we consider the features provided for the analysis of User Experiments. GPSS World provides for two kinds of automatically generated experiments that are designed for you, and User Experiments that are designed by you.

### 13.3.1 User Experiments

The immediate goal of a User Experiment is to provide the information needed by the built-in ANOVA library procedure. That means that you must fill a Result Matrix with all the results of the experiment. If you prefer, you can do this directly through the use of Command Lists and manual entries. However, the PLUS Language, which allows you to write programmable experiments can make your life a lot easier if your experiment is complex.

You should begin by selecting the factors whose effects you will be studying. For example, you may want to measure the effect of the number of phone lines on the average waiting time in a telephone exchange. Factor A of your experiment would be the number of phone lines, and you would assign it to the first dimension of your Result Matrix. Next we decide how many treatment levels (i.e. number of phone lines) will be studied. Let's say that we want to simulate exchanges with 4, 8, and 16 lines. We would then need 3 slots in the Result Matrix for these three conditions. Therefore the first dimension in the Result Matrix would have a size of 3.

We continue to add factors to the experiment in this way, using the number of treatment levels as the size of the corresponding dimension in the Result Matrix. Normally, we use the last dimension for replicates. The size of the Replicate Dimension must obviously be large enough to contain the results from any cell (i.e. treatment combination) in the experiment. The only restriction is that the total number of dimensions cannot exceed 6. When you analyze the data, you must explicitly tell the ANOVA routine which dimension is used for replicates.

Remember to initialize the Result Matrix to UNSPECIFIED, before you begin storing results into it. That way, it is clear to the ANOVA routine when a run has not been performed.

When you call the ANOVA routine with your results, you must also decide if you want to exclude 3-way, or even 2-way, interactions from the analysis. If you can justifiably do this, you will improve the statistics in the ANOVA table.

That is a rough outline of the creation of User Experiments. In the remainder of this section we will look at the steps in more detail. We will consider the embedded PLUS programming language, the requirements and restrictions of the Result Matrix, and the use of the ANOVA library procedure. Lesson 19 of the GPSS World Tutorial Manual guides you through an example.

Experiments created using the automatic experiment generators are not considered User Experiments, and are discussed later. However, you may find it useful to study the programming of a generated experiment before you attempt to do it yourself.

### 13.3.2 PLUS Experiments

PLUS is a simple but powerful programming language that is an important part of the GPSS World simulation environment. It began as a source of custom programmable subroutines to be accessed during GPSS simulations, and has progressed to become a control language that can direct the details of the runs in an experiment, and report on the results. The complete description of PLUS is in Chapter 8 of this manual. Lesson 17 of the GPSS World Tutorial Manual covers it, as well.

An Experiment is a PLUS Procedure where the keyword PROCEDURE is replaced by the keyword EXPERIMENT. Experiments can do everything that normal Procedures can, and a whole lot more. An Experiment is invoked by a CONDUCT Command which names the invoked Experiment and passes it any arguments it might require.

The power of a PLUS Experiment is based on its ability to invoke the "DoCommand" library procedure. Nearly all GPSS Commands can be executed by a DoCommand invocation with a Command String as an argument. Not only can an Experiment invoke DoCommand, but also so can any normal PLUS Procedure that has been called during the execution of an Experiment. This adds a lot of flexibility to the programming of complex experiments.

An example of a DoCommand invocation is:

```
DoCommand("START 1000");
```

The DoCommand Library Function is peculiar in that its argument string is compiled in global scope. This means that you cannot reference local variables in an argument of a DoCommand invocation. Instead, if you need to pass local values to DoCommand, you must use PLUS string commands to create an argument string including evaluated results. As an example of this, here's the "Run Execution Procedure" created by the Automatic Experiment Generators to log run information to the Journal:

```
*****
```

\* The Run Execution Procedure \*

```

*****
PROCEDURE SEM_GetResult() BEGIN
    /* Run Simulation and Log Results. */
    /* Treatments have already been set for this run. */
    TEMPORARY CurrentYield,ShowString,CommandString;

    /* Run Procedure Call */
    RunProc(SEM_NextRunNumber);

    /* Place Run Results in Journal. */
    ShowString = PolyCatenate("Run ",String(SEM_NextRunNumber),". ", " ");
    ShowString = PolyCatenate(ShowString," Yield=",String(CurrentYield),". ");
    ShowString = PolyCatenate(ShowString," Factor1=",String(Factor1), ";" );
    ShowString = PolyCatenate(ShowString," Factor2=",String(Factor2), ";" );
    ShowString = PolyCatenate(ShowString," Factor3=",String(Factor3), ";" );
    CommandString = PolyCatenate("SHOW """,ShowString,"""", " ");
    DoCommand(CommandString);

    SEM_NextRunNumber = SEM_NextRunNumber + 1;
    RETURN CurrentYield;

END;

```

*Figure 13-6. Using a DoCommand in a PLUS Procedure*

This procedure issues a SHOW Command in 3 steps. It records the result and conditions of the run in the Journal. First it creates a temporary string, ShowString, which contains variables evaluated locally. Second, it builds the overall string with the completed SHOW Command in the temporary variable CommandString. Finally, it invokes the DoCommand library procedure. Normal Procedures such as this can only invoke DoCommand when a CONDUCT Command is in effect. In other words, such Procedures must be called either directly, or indirectly, by a PLUS Experiment.

In experiments with many runs, it is convenient to use a PLUS Procedure to do the set up of each simulation run in the experiment. Such a procedure initializes the random number generators and issues commands that control the simulation. Then, you can set up your high level PLUS Experiment to call the Run Procedure each time a simulation is to be run.

An example of the default Run Procedure created by the Experiment Generators is in Section 13.4. 2. Even when GPSS World Creates the Experiment for you, it is up to you to modify the Run Procedure to fit the run conditions of your own simulation project.

### 13.3.3 The Result Matrix

The primary source of information from an experiment is kept in the Result Matrix. In order for the ANOVA library procedure to analyze the results of an Experiment, you must assure that the Result Matrix has been set up appropriately.

In GPSS World, a Matrix Entity can have up to 6 dimensions. There is no arbitrary limit on the size of each dimension other than that imposed by the virtual memory of the computer. However, one of the most important jobs in setting up a GPSS World Experiment is the selection of factors. Each factor to be included in the Experiment is assigned to one of the dimensions of the Result Matrix. In addition, depending on the design of the experiment, you will probably want to use one dimension for replicates, which must be large enough to handle the runs in the largest cell. For each dimension used for a factor, the size at least as large as the number the treatment levels of that factor. That is because there must be a place in the Result Matrix for each run. There can be any number of these treatments up to the limits imposed by the virtual memory of your computer.

If your experiment has more than one result metric, you need only define a separate Result Matrix for each one, and then utilize the ANOVA Procedure for each.

### Viewing a Result Matrix

GPSS World Matrices can be viewed in a Matrix Window or printed in a Standard Report. Since GPSS World is unable to view all 6 dimensions at one time, it presents only a two dimensional slice in a Matrix Window. A small Dialogue Box is used to select the slice of the Matrix that is to appear.

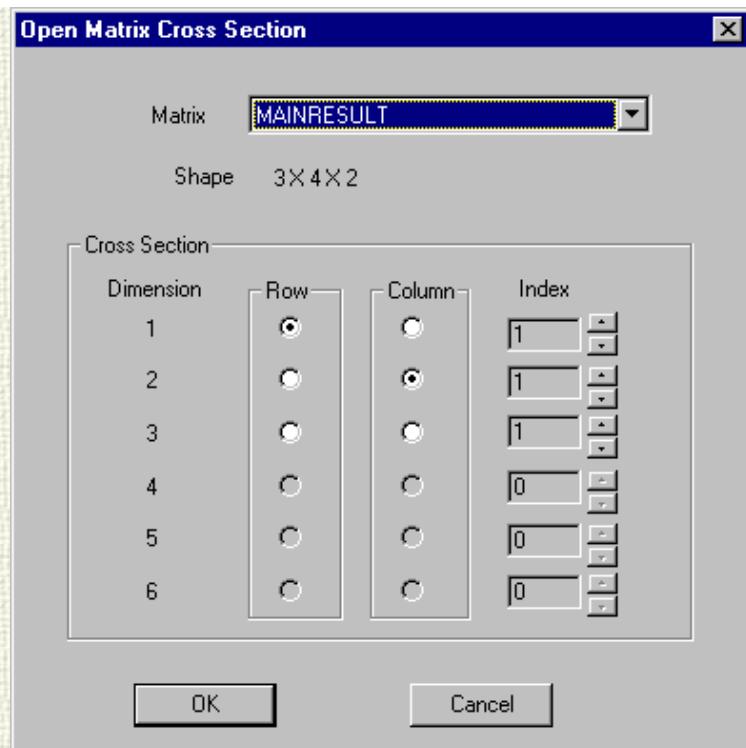


Figure 13-7. The Matrix Cross Section Selection Dialog

Figure 7 presents the dialog that allows the selection of a slice through a higher order Matrix Entity. In this case we have a 3 by 4 by 2 3-dimensional Result Matrix. To use it, first select the two dimensions to form the Rows and Columns of the Matrix Window, then use the index of the remaining dimensions to select deeper slices, other than the first. It may be helpful to think of a loaf of bread seen head-on. The row and column numbers indicate where on the slice we are, and the index of the third dimension chooses the slice. In Matrices with 4 or more dimensions, an index value would be needed for each dimension not participating as a row or column.

Once the Matrix Window, itself, is open you will have the chance to scroll to any location within that slice. Matrix Windows are updated dynamically, and any number of Matrix Windows can be open at the same time.

The other way to view the elements of a Matrix Entity is to include the results in the Standard Report. Page 2 of the Settings of a Model or Simulation Object (Edit / Settings) has a checkbox which when checked, puts a line for each element of the Matrix in subsequent Standard Reports.

## Initialization of a Result Matrix

Variables within GPSS World can take on a variety of data types without user intervention. Behind the scenes, values are converted from one form to another as needed. For example, a string is converted to a real number if the operator using that value requires a real argument.

Variables and Matrix elements can take on the value UNSPECIFIED to indicate that no value has yet been assigned to it. UNSPECIFIED is not a string and will cause an Error Stop if an operator encounters it while evaluating an expression.

For compatibility reasons, when GPSS Matrices are created or CLEARed, they are given the value of 0. However, a Result Matrix should have all elements initialized to the UNSPECIFIED state before the Experiment is executed. In this way, the ANOVA library procedure can detect when runs are missing. Otherwise, a 0 value is taken to be the result of a run of the experiment. The INITIAL Command now supports the initialization of a Matrix to UNSPECIFIED.

### **INITIAL MatrixName,UNSPECIFIED**

This Command causes all elements in the matrix to be set to the UNSPECIFIED state. Normally, you should place the initialization statement outside the Experiment, itself. This is what GPSS World does when it generates a Screening Experiment.

In addition to showing which data is missing, the use of the UNSPECIFIED state in the Results Matrix allows an Experiment to be saved and restarted without repeating runs that have already been completed. To be restartable, the Experiment should test the destination element for UNSPECIFIED before calling the Run Procedure. Automatically generated Screening Experiments use this method.

Here's another example of the creation and initialization of a Result Matrix.

```
MyResults MATRIX ,3,5,4,3  
INITIAL MyResults,UNSPECIFIED
```

*Figure 13-8. GPSS Statements Initializing a Result Matrix*

In this example a Result Matrix is created which can hold the results of an experiment with 4 factors, or one with 3 factors with replicates. Factor A can have up to 3 treatment levels, factor B can have up to 5, and so on,

The INITIAL Statement sets all elements in the Matrix to "UNSPECIFIED". This allows the ANOVA routine to detect elements in the matrix where no results are available.

### 13.3.4 ANOVA Library Procedure

The final step in a User Experiment is usually the analysis of results. By calling GPSS World's ANOVA library procedure, most of the work is done for you. The ANOVA Procedure can handle a multiway Analysis of Variance considering up to 6 factors and up to 3-way interactions of all combinations of main factors.

There are 3 arguments to the ANOVA Procedure. The first is the name of the Result Matrix, which is described in more detail, below.

The second argument is an optional dimension of the Result Matrix to be used for replicates. Each level in this dimension represents a run with distinct random number seeds. The ANOVA Procedure uses all information associated with the replicate dimension as part of the estimate of the Standard Error. This provides an estimate with more information and more Degrees of Freedom than had a replicate dimension not been used.

Some experimental designs do not use a replicate dimension. In that case, use a 0 as the second argument.

The third argument controls the factor interactions to be included in the statistical model. If the argument is 2, only 2-way interactions are included in the analysis. If the argument is 1, no interaction terms are included. A value of 3 or greater causes main effects, 2-way interactions, and 3-way interactions to be included in the statistical model. Four way interactions are not supported by GPSS World. The reason you would want to limit the interaction terms in the statistical model is this: when interaction terms are removed from the model, additional information and degrees of freedom become available for a better estimate of the Standard Error.

The main goal of the ANOVA Procedure is to create a standard ANOVA Table in the Journal, which indicates the resulting F statistic and its critical value. In addition, when you call the ANOVA Procedure from a PLUS routine, the Standard Error is returned when the ANOVA Procedure completes.

### The ANOVA Table

Here we have an ANOVA table generated automatically by the library procedure in GPSS World.

ANOVA					
Source of Variance	Sum of Squares	Degrees of Freedom	Mean Square	F	Critical Value of F (p=.05)
A	28.000	2	14.000	5.600	3.89
B	21.000	3	7.000	2.800	3.49
AB	11.000	6	1.833	0.733	3.00
Error	30.000	12	2.500		
Total	90.000	23			

Treatment Level A B	Count	Mean	Minimum	Maximum	95% C.I. (SE)
1 1	2	3.000	2.000	4.000	[ 0.564, 5.436 ]
1 2	2	1.000	1.000	1.000	[ -1.436, 3.436 ]

01/27/01 13:26:44      1.5811388

Figure 13-9. An ANOVA Table

Each factor and interaction in the statistical model is represented by a distinct line in the ANOVA table. In each line we have the basic calculations and the F statistic for that effect.

The larger the F value the stronger the effect. It's considered significant if it exceeds what is called "the critical value of F", which is calculated for you by GPSS World. The ANOVA table above shows the effect of factor A to be significant and the effects from factor B and the AB interaction to be not significant.

It's important to avoid Variance Reduction Techniques when using Analysis of Variance. They distort the variability introduced to simulate real-world randomness. Reducing the intrinsic variability of the observations causes the F statistics to be overstated.

When you do an Analysis of Variance, you should investigate the assumptions behind the analysis, such as normality and unchanging variance, which statisticians call homoscedasticity. GPSS World provides another feature for this purpose. If you create a GPSS Table Entity with a specific name, the ANOVA Procedure will fill it with the residuals of the Analysis of Variance. This is discussed in the next section.

The cell statistics report follows the ANOVA table, and it gives ranges and 95% Confidence Intervals for each Treatment Combination in the experiment, based on a pooled estimate of the Standard Error.

Although the Confidence Intervals associated with treatment combinations are not independent, the ranges can still be used as a check on the assumptions behind the ANOVA. When these vary wildly the homogeneity of variance in the residuals is in doubt. In that case, you may need to examine the data more closely before drawing your final conclusions. One way is to define a GPSS Table to collect the residuals calculated in the Analysis of Variance. The next section tells you how to do this.

### 13.3.5 The Table of Residuals

When the ANOVA Procedure performs its analysis, it searches for a GPSS Table Entity of a special name. It looks for a Table Name the same as the name of the Result Matrix except that the string "\_Residuals" is appended. If it finds a Table Entity of the derived name, the ANOVA Procedure tabulates the residuals of the analysis for the purpose of checking the assumptions used by ANOVA. For example, if you have a GPSS Table entity named MyResults\_Residuals defined when you call ANOVA with a Result Matrix named MyResults, GPSS World will clear the Table and fill it with the residuals from the Analysis of Variance.

Skew and outliers can sometimes be detected in the Table of Residuals, and the lack of homogeneous variance (heteroscedasticity), when it occurs, is often apparent here and in the descriptive statistics report which follows the ANOVA Table.

## 13.4 The Automatic Experiment Generators

GPSS World can generate either screening or optimization experiments for you. You can use them to rule out irrelevant factors or to find the best treatment combination. The process inserts a PLUS Experiment into your Model Object and optionally loads a Function Key with the appropriate CONDUCT Command. It takes just 4 easy steps:

1. Fill out the dialog (Edit Menu) and Click OK.
2. Edit the Run Procedure and Click OK.
3. Translate the model (Ctrl+Alt+S).
4. Press the Function Key.

That's all there is to it. The PLUS Experiment will run, reporting its status to the Journal. When it completes, it will create a full report, as well. In the case of an Optimizing Experiment, the report will include a mathematical description of the Response Surface around the optimum. Then, analyzing the Report is up to you.

Both types of generated PLUS Experiment contain invocation of a PLUS Run Procedure. It is your responsibility to ensure that this procedure sets up the run conditions appropriately. The Run Procedure is called each time the Experiment begins a new simulation run. Optionally, you can have the Experiment Generator create a template Run Procedure for you. Even so, you must still adapt it to your own conditions. The next section tells you how.

### 13.4.1 The Run Procedure

The Run Procedure is the flexible connection between the generated experiment and the user simulation. Both Screening Experiments and Optimizing Experiments use them. Generated Experiments repeatedly call the Run Procedure that the user names in the creation dialog in order to execute each run in the experiment.

Every experiment should have a Run Procedure written by you or generated by GPSS World. A Run Procedure is used to establish the measurement period during the simulation that is to be observed. It sets up each run in the Experiment according to the particular requirements of your simulation and is invoked once for each run in the experiment. Run Procedures are PLUS Procedures that are called by Experiments and therefore may contain DoCommand calls.

Normally the Run Procedure contains setup commands for the simulation such as RMULT Commands to control the Random Number Generators, and START and RESET Commands to set up the measurement period of the run.

Here are the most important things you should take care of in the Run Procedure:

- ◆ Use CLEAR OFF to remove old transactions between replication but still preserve the Results Matrix.
- ◆ If the Experiment does not set the treatment levels elsewhere, do it here based on arguments of the Run Procedure. This is not necessary for generated experiments. They set the treatment levels for you.
- ◆ Use RMULT to set the Random Number Generators according to an argument in the run procedure. If you have not defined a replicate dimension in the Result Matrix, you should "completely randomize" and use distinct random number stream seeds for each run.
- ◆ Use PLUS String Procedures to create the DoCommand argument.
- ◆ Use START to begin the start-up period.
- ◆ Use RESET to reset the statistics accumulators prior to the measurement period.
- ◆ Use START to begin the measurement period.

❖ Unless the Experiment stores results in the Result Matrix using a PLUS assignment statement, do it here based on arguments of the Run Procedure. This is not necessary for generated experiments.

When an experiment is generated automatically, the generator optionally creates a template Run Procedure which can be modified as desired. As an example, here is the default Run Procedure created by the Experiment Generators:

```
PROCEDURE RunProc(Run_Number) BEGIN  
  
    DoCommand("CLEAR OFF"); /* Must use OFF to preserve results. */  
  
    /* EXPAND THIS RMULT IF YOU HAVE MORE RNGs. */  
    /* All Random Number Streams must have new seeds. */  
    TEMPORARY CommandString;  
  
    /* Evaluate before passing to DoCommand. */  
    CommandString = Catenate("RMULT ",Run_Number#111);  
  
    /* DoCommand compiles the string in Global Context. */  
    DoCommand(CommandString);  
  
    /* SET UP YOUR OWN RUN CONDITIONS. */  
    DoCommand("START 100,NP"); /* Get past the Startup Period. */  
    DoCommand("RESET"); /* Begin the Measurement Period. */  
    DoCommand("START 1000,NP"); /* Run the Simulation. */  
  
END;
```

*Figure 13-10. The Default Run Procedure*

When you let GPSS World create a Run Procedure, it is up to you to customize it to fit your own situation. You can edit it before, or after, it is inserted into your Model Object.

### 13.4.2 Screening Experiments

Screening Experiments are used to look for factors which must be investigated more thoroughly or those which should be considered to be irrelevant. However, the combinatorial nature of measuring main effects and all the possible interactions often makes the complete set of runs, called a full factorial experiment, too lengthy and/or expensive to run. To make the procedure more manageable, we restrict the number of treatment levels to two per factor, and we look for ways to run only a fraction of the possible runs without sacrificing the most important information. The Screening Experiment Generator of GPSS World does most of the work for you.

You must decide which information you can give up in order to reduce the number of runs in the experiment. Normally it would be the higher-level factor interactions that are usually insignificant and can be ignored. Even if you suspect otherwise, remember that the purpose of the screening experiment is to identify the important factors to be studied in more depth later. When you ask GPSS World to reduce a full factorial screening experiment to a fractional set of runs, some effects get mixed with others to the point where you can't distinguish them in the Analysis of Variance. GPSS World will partition the set of all possible effects subsets called Alias Groups. Only the combined result of all effects in the group is reported in the Analysis of Variance. This means that if factor A and factor B are in an Alias Group, you will have lost any ability to tell which is responsible for the observed effect. Perhaps even worse than that, it is possible for two strong effects in the same Alias Group to cancel each other, thereby concealing the actual effects of each factor.

As you can see, it is extremely undesirable to have more than one main effect in an Alias Group. However, it is usually possible to arrange Alias Groups so that each main effect and the most important 2-way interactions are in Alias Groups of their own, with only higher order interactions as companions. GPSS World provides default Alias Groups which isolate main effects, if possible, and it makes it easy for you to change the Alias Groups if the defaults are not suitable. In any case, when you generate a GPSS World Screening Experiment, you should always examine the Alias Groups before you begin the runs. If you haven't separated the most important effects into distinct groups, the interpretation of results will be difficult.

## Generating A PLUS Screening Experiment

The first step in creating a Screening Experiment automatically is to open the dialog. You access it by clicking "Insert Experiment" in the Edit Menu of the Main Window, and clicking on the "Screening

Experiment ... " item. You must have an Open Model Object active in order to be able to do this.

Now just fill in the blanks and click OK when you're done. Figure 13-11 shows the Screening Experiment Dialog.

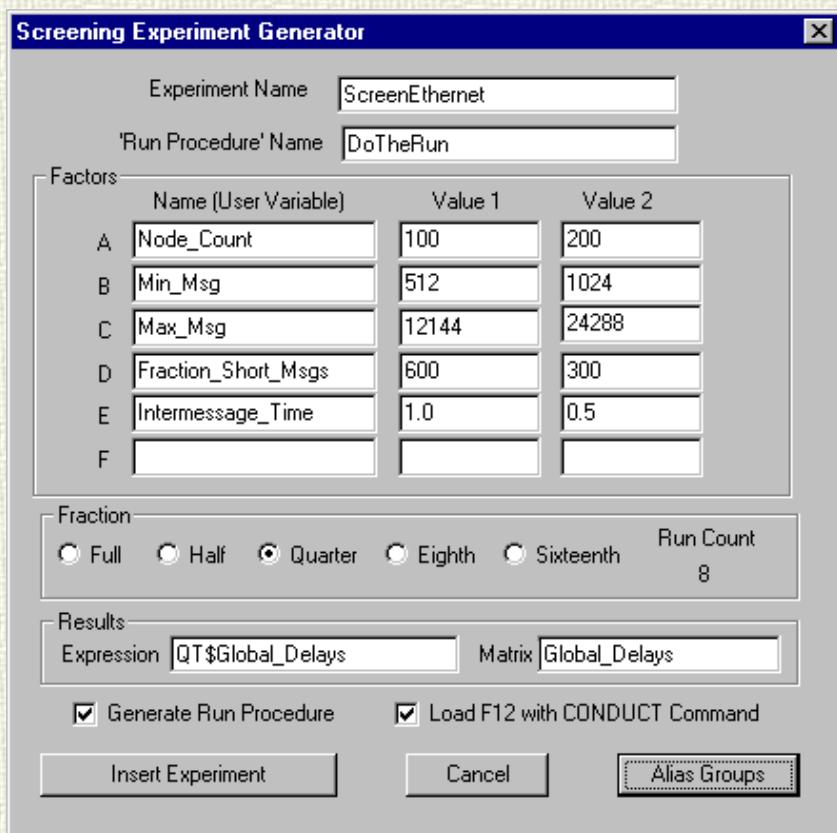


Figure 13-11. The Screening Experiment Generator Dialog

To specify the PLUS Experiment that is about to be generated, we simply fill in the fields of the Dialog. The "Experiment Name" and the "Run Procedure Name" fields are used as the PROCEDURE names in the generated experiments. The EXPERIMENT calls the Run Procedure repeatedly.

The Factors of the Experiment come next. Each factor name is actually the name of a User Variable and must obey the GPSS World naming conventions. It must begin with an alphabetic character and it must not clash with a keyword, SNA, or SNA class. Since screening experiments in GPSS World are full or fractional 2<sup>k</sup> experiments, there are two treatment levels to be specified for each factor. You must specify the names and two treatment levels for between 1 and 6 factors, inclusively. Factors must be specified consecutively starting with the A factor. The choice of treatment levels is crucial. You may want to do some preliminary experimentation so that the choices you make are good ones. Choose levels that are far apart to elicit a change of behavior and try to avoid levels where the effects are hidden by other factors.

The "Fraction" group is next. It allows us to specify what fraction of the full 2<sup>k</sup> experiment is to be run. The run count that will result is specified to the right. Choose a smaller fraction to decrease the run count.

The "Result Expression" is required. You must specify an expression to be evaluated as the metric of the simulation.

Next we have two checkboxes that allow us to select additional options. First, you can have GPSS World generate a template Run Procedure to go along with the generated experiment. You can edit this Run Procedure to suit your needs. Second, you can have GPSS World set up a setting which loads the appropriate CONDUCT Command into the setting for Function Key 11. If you choose this, after you create the Simulation Object, you only have to press the F11 Function Key to run the experiment.

Before we generate the experiment we should examine the Alias Groups. To do so, click the Alias Groups button to examine the consequences. This brings up the Alias Groups Dialog.

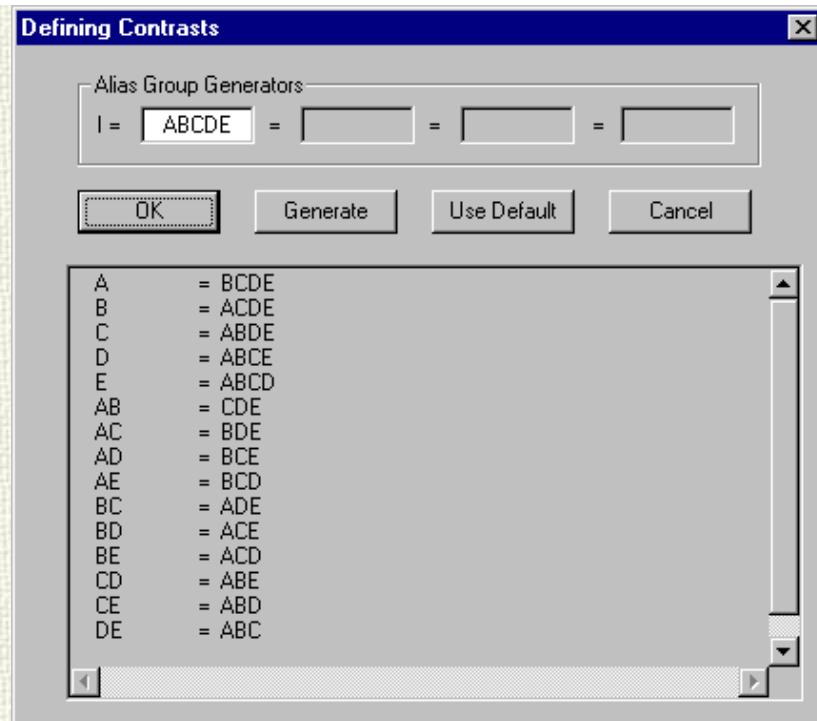


Figure 13-12. The Alias Group Dialog. Improved Experimental Design.

From the Alias Groups Dialog we can see that this experimental design isolates all the main factors, does not confound them with 2-way interactions, and it even isolates the 2-way interactions of the first few main effects.

Take a closer look the effects used as "Generators" in the Alias Group Dialog. We can change the partition of the effects into a different set of Alias Groups if we so desire by simply using a different set high level interactions as generators. To do this, change the generators and click on the Generate Button. Before you do this there are several things to know about the generators. First, you should know that when GPSS World creates Alias Groups it first expands the set of generators by forming all possible modulo 2 products and adding them to the set of generators, now called the "Expanded Set" of generators. For example if ABCD and BCDE are generators, their product, AE, will be in the Expanded Set of generators ( $ABCDE * BCDE = ABBCCDDE = A (BB) (CC) (DD) E = AE$ ). Note that any effect in the Expanded Set of generators will not appear in the report of effects. This means that you must not choose generators that result in an important effect occurring in the Expanded Set of generators. You can return to the default generators suggested by GPSS World by clicking on the Use Default Button. Click the Generate Button to create the new list of Alias Groups.

The next step is to insert the PLUS code comprising the experiment into your Model Object. When you have made the best compromise between the number of runs and the aliasing of effects, click the OK Button in the Alias Group Dialog and the Insert Experiment Button in the Screening Experiment Generator Dialog. If you have asked GPSS World to generate a Run Procedure, next we see a Dialog that allows you to personalize it. The details about personalizing the Run Procedure are in the previous section.

In any case, you may want to edit the generated experiment itself in your Model Window. You can change the Run Procedure there, as well. Just click OK and all the new GPSS World Statements comprising the Screening Experiment are placed at the bottom of your Model. A sample of a generated PLUS Experiment follows in the next section.

## The Generated Experiment

```
*****
*
* ScreenEthernet
* Fractional Factorial Screening Experiment
*****
```

```

INITIAL Global_Delays,UNSPECIFIED

EXPERIMENT ScreenEthernet() BEGIN

    /* Run 1 */
    Node_Count = 100;
    Min_Msg = 512;
    Max_Msg = 12144;
    Fraction_Short_Msgs = 600;
    Intermassage_Time = 1.0;
    IF (StringCompare(DataType(Global_Delays[1,1,1,1,1]),"UNSPECIFIED")EO)
    THEN BEGIN

        /* Run Procedure Call */
        DoTheRun(1);
        Global_Delays[1,1,1,1,1] = QT$Global_Delays;

    END;
    **** Runs 2-15 would go here ****

    /* Run 16 */
    Node_Count = 200;
    Min_Msg = 1024;
    Max_Msg = 24288;
    Fraction_Short_Msgs = 300;
    Intermassage_Time = 1.0;
    IF (StringCompare(DataType(Global_Delays[2,2,2,2,1]),"UNSPECIFIED")EO)
    THEN BEGIN
        /* Run Procedure Call */
        DoTheRun(16);
        Global_Delays[2,2,2,2,1] = QT$Global_Delays;
    END;

    /* Aliased Effects in Fractional Factorial Experiment */
    EFFECTS(Global_Delays,"I=ABCDE");

END;

*****
* Run Procedure *
*****
PROCEDURE DoTheRun(Run_Number) BEGIN

    DoCommand("CLEAR OFF"); /* Must use OFF to preserve results. */

    /* EXPAND THIS RMULT IF YOU HAVE MORE RNGs. */
    /* All Random Number Streams must have new seeds. */
    TEMPORARY CommandString;
    /* Evaluate before passing to DoCommand. */
    CommandString = Catenate("RMULT ",Run_Number#111);
    /* DoCommand compiles the string in Global Context. */
    DoCommand(CommandString);

    /* SET UP YOUR OWN RUN CONDITIONS. */
    DoCommand("START 100,NP"); /* Get past the Startup Period. */
    DoCommand("RESET"); /* Begin the Measurement Period. */
    DoCommand("START 1000,NP"); /* Run the Simulation. */

END;
*****

```

## Running the Experiment

CONDUCT is the GPSS World Command that is used to invoke an Experiment. The syntax is similar to that of a Procedure Call. But we have a better way. In the Experiment Generation dialog, we can choose to have GPSS World load the F11 Function Key with the appropriate CONDUCT Command. You can see this in the Model Object's Settings. (see Edit / Settings / Function Keys).

To start the execution of the experiment, we must do two things. First we must Translate the Model, thereby creating the Simulation Object. Click on Command / Create Simulation to do this, or just press Ctrl+Alt+S.

Then we press the F11 Function Key. This enters the CONDUCT Command. The Experiment goes to work, reporting status and output to the Journal Window of the newly created Simulation Object. Figure 13-13 shows the results of a Screening Experiment.

Alias Group	Effect	Sum of Squares	Degrees of Freedom	F - for Only Main Effects	Critical Value of F ( $p=.05$ )
A = BCD	0.030	0.002	1	0.000	10.13
B = ACD	-0.058	0.007	1	0.000	10.13
AB = CD	-0.178	0.063	1		
C = ABD	0.063	0.008	1	0.000	10.13
AC = BD	-0.014	0.000	1		
AD = BC	-0.109	0.024	1		
D = ABC	0.177	0.062	1	0.000	10.13
<b>Grand Mean</b>		<b>11.456</b>			
Total		1050.158	7		
Error		1050.079	3		

01/16/01 13:45:33 Experiment Ended.

Figure 13-13. Results of Screening Experiment.

The Screening Experiment Report shows the strength of each effect and, if it can, calculates an F statistic for those alias groups with main effects as members. From it you can plan future actions such as ignoring insignificant factors, and more fully exploring the important ones.

### 13.4.3 Optimizing Experiments and Response Surfaces

Optimization and the quantitative prediction of the behavior of a system are often the primary objectives of a simulation project. Both of these are supported directly by GPSS World's Optimizing Experiment Generator.

Response Surface Methodology is a set of statistical techniques for empirical model building and model exploration. Although most often used to estimate optimum treatment levels, it also can provide important information for general predictions of the yield expected under untested conditions. GPSS World supports several important Response Surface Methodologies in a way intended to make them easy to use. The simulation analyst sets up the initial conditions in a few User Dialogues and GPSS World then automatically creates a response surface experiment that seeks an optimum value. Ideally, you can start the experiment and let it run unattended until it finds the optimum value of the chosen metric. When you fill in the dialog windows you are, in effect, defining a box (actually a "hyperbox") called the Local Experimental Region that wanders over the set of permissible factor levels. If you make this box too small, the experiment will require more runs than necessary to move to the optimum. If you make it too large, the estimates of the optimum treatment levels will be less precise than they could be.

GPSS World applies several different methodologies as it explores the response surface of a GPSS World Simulation Object. The primary concept is that of a Local Experimental Region which is defined by you in the setup dialog. To define it, each of up to 5 factors to be studied are associated with two values defining the extent of the Local Experimental Region. Then, it is the whole Region that moves around the factor space. The goal, which is not always achieved, is to contain the optimum within the Local Region, perform a confirmation run, and then to report the optimum conditions and the mathematical model that was used.

GPSS World uses several experiments of increasing complexity to get the information it needs. As results are available, it attempts to fit either a linear model or a second order model (including two-

way interactions) to the data. It begins by using the Method of Steepest Ascent as it leaves its starting point. The direction is determined by the gradient of the first model that fits the data. If the goodness-of-fitness test fails for the first order model, the second order model is used.

Once a gradient is calculated, the center of the Local Experimental Region moves until it encounters either diminishing returns or a limit asserted in the setup dialog. Each step of the movement along the gradient never exceeds the width of the local experimental region, as defined in the Optimizing Experiment Generator Dialog. If the range of treatment levels is too narrow, it will take many unnecessary steps to climb the gradient. Each step in the move requires a simulation at the center of the Local Experimental Region, and its results are reported in the Journal. If, when movement stops, the local experimental region does not then contain an optimum, the experiment attempts to move in a new direction. This redirection can be suppressed by the Redirection Limit in the Optimizing Experiment Generator Dialog.

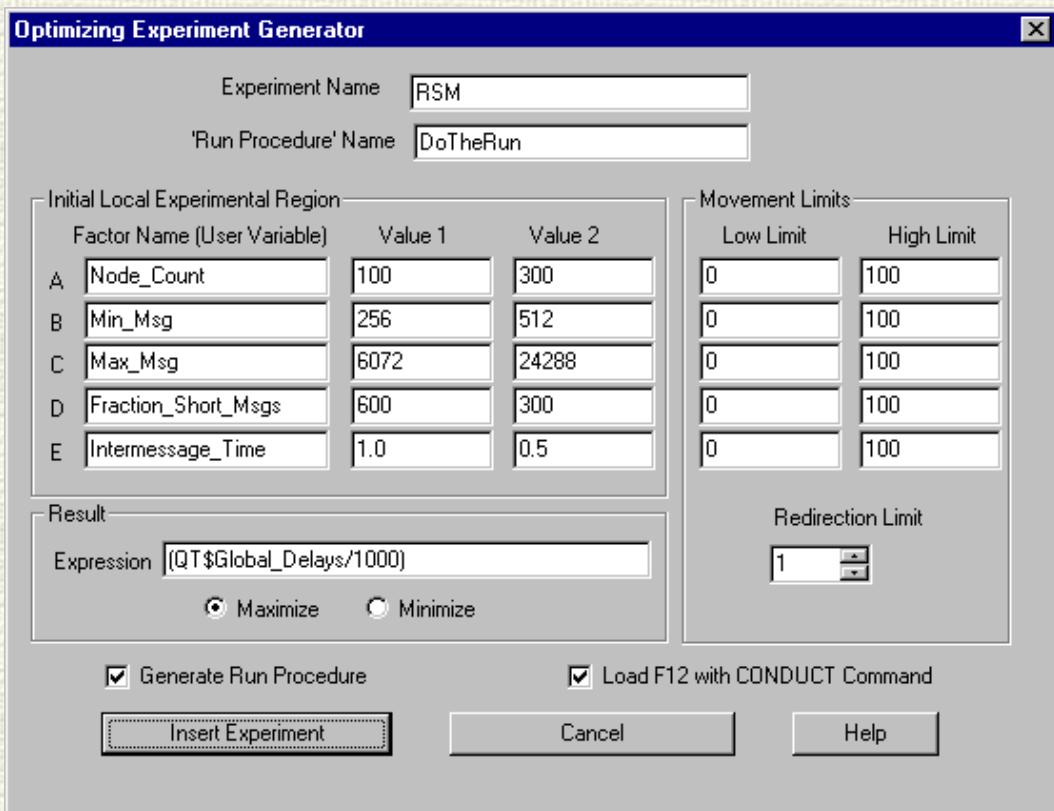


Figure 13-14. The Optimizing Experiment Generator Dialog.

Figure 13-14 shows the dialog used to generate Optimizing Experiments. Just as with the Screening Experiment Generator it is accessed through the Edit Menu of the Main Window.

To specify the PLUS Experiment that is about to be generated, we simply fill in the fields of the Dialog. The "Experiment Name" and the "Run Procedure Name" fields are used as the PROCEDURE names in the generated experiments. The EXPERIMENT calls the Run Procedure repeatedly.

The Factors of the Experiment come next. Each factor name is actually the name of a User Variable and must obey the GPSS World naming conventions. It must begin with an alphabetic character and it must not clash with a keyword, SNA, or SNA class. The two treatment levels of each factor define the Local Experimental Region which moves around searching for the optimum conditions. You must specify the names and two treatment levels for between 1 and 5 factors, inclusively. Factors must be specified consecutively starting with the A factor. The choice of treatment levels is crucial.

Several important trade-offs are implicit in the way you define the Local Experimental Region. First, it forms the starting point of the experiment. You should make it as close to the optimum as you can. Second, making it small usually makes the mathematical model fit better. Third, when the experiment moves one step toward the optimum it moves to an adjacent region. Therefore, a larger experimental region will take fewer steps. As you can see, the issues conflict with each other. It is up to you to choose compromises. Even further, don't overlook the fact that it is easy to modify a generated experiment after it has been inserted into your Model.. It's worthwhile to examine the generated PLUS code to look for ways to improve the experiment. You may want to do some preliminary experimentation so that the choices you make are good ones.

You should apply limits to the Optimizing Experiment. The section of the dialog for Move Limits gives you a chance to restrict the movement of the Local Experimental Region. If possible you should enter limits for each of the factors so that time won't be wasted on infeasible simulations. The Redirection Limit is another way to restrain the search. This value sets a limit to the number of times the experiment can start moving in a new direction in the factor space. This limit can prevent an unending cycle from occurring.

The "Result Expression" is required. You must specify an expression to be evaluated as the metric of the simulation.

Finally, we have two checkboxes that allow us to select additional options. First, you can have GPSS World generate a template Run Procedure to go along with the generated experiment. You can edit this Run Procedure to suit your needs. Second, you can have GPSS World set up a setting which loads the appropriate CONDUCT Command into the setting for Function Key 12. If you choose this, after you create the Simulation Object, you only have to press the F12 Function Key to run the experiment.

The next step is to insert the PLUS code comprising the experiment into your Model Object. When you have filled out all the appropriate items in the Optimizing Experiment Generator Dialog, click Insert Experiment to create the PLUS code and insert it into your model. If you have asked GPSS World to generate a Run Procedure, next we see a Dialog that allows us to personalize it. The details about personalizing the Run Procedure are in the beginning of this section.

In any case, you may want to edit the generated experiment itself in your Model Window. You can change the Run Procedure there, as well. Just click OK and all the new GPSS World Statements comprising the Optimizing Experiment are placed at the bottom of your Model.

## Running the Experiment

CONDUCT is the GPSS World Command that is used to invoke an Experiment. The argument to the CONDUCT Command is similar to a Procedure Call. But we have a better way. In the Experiment Generation dialog, we can choose to have GPSS World load the F12 Function Key with the appropriate CONDUCT Command. You can see this in the Model Object's Settings. (see Edit / Settings / Function Keys).

To start the execution of the experiment, we must do two things. First we must Translate the Model, thereby creating the Simulation Object. Click on Command / Create Simulation to do this, or just press Ctrl+Alt+S.

Then we press the F12 Function Key. This enters the CONDUCT Command. The Experiment goes to work, reporting status and output to the Journal Window of the newly created Simulation Object. Figure 13-15 shows the results of an Optimizing Experiment.

```

04/24/01 15:02:00 "Run 90.00000. Yield=95.94288. cat=158.0000; dog=19.76101;""
04/24/01 15:02:00 "Run 91.00000. Yield=100.0000. cat=160.0000; dog=20.00000;""
04/24/01 15:02:00 "Run 92.00000. Yield=95.94288. cat=162.0000; dog=20.23899;""
04/24/01 15:02:00 "Move ending."
04/24/01 15:02:01 "Run 93.00000. Yield=98.00000. cat=159.0000; dog=19.00000;""
04/24/01 15:02:01 "Run 94.00000. Yield=98.00000. cat=159.0000; dog=21.00000;""
04/24/01 15:02:01 "Run 95.00000. Yield=98.00000. cat=161.0000; dog=19.00000;""
04/24/01 15:02:01 "Run 96.00000. Yield=98.00000. cat=161.0000; dog=21.00000;""
04/24/01 15:02:01 "Run 97.00000. Yield=100.0000. cat=160.0000; dog=20.00000;""
04/24/01 15:02:01 "Run 98.00000. Yield=100.0000. cat=160.0000; dog=20.00000;""
04/24/01 15:02:01 "Run 99.00000. Yield=100.0000. cat=160.0000; dog=20.00000;""
04/24/01 15:02:01 "Run 100.0000. Yield=99.00000. cat=161.0000; dog=20.00000;""
04/24/01 15:02:01 "Run 101.0000. Yield=99.00000. cat=159.0000; dog=20.00000;""
04/24/01 15:02:02 "Run 102.0000. Yield=99.00000. cat=160.0000; dog=21.00000;""
04/24/01 15:02:02 "Run 103.0000. Yield=99.00000. cat=160.0000; dog=19.00000;""
04/24/01 15:02:02 Using Model:
04/24/01 15:02:02 Y = -300 +0 A +40 B
04/24/01 15:02:02 +0 A B
04/24/01 15:02:02 -1 A^2 -1 B^2
04/24/01 15:02:02 Predicted optimum yield is 100.
04/24/01 15:02:02 Optimum is in the local Experimental Region.
04/24/01 15:02:02 RSM_FitSurfaceToData() returns 4.
04/24/01 15:02:02 Experiment Ended.

```

Figure 13-15. Results of Optimizing Experiment.

When successful, the Optimizing Experiment reports the results of each run. From this can be seen the results of the sub-experiments and the movement of the local experimental region. The generated PLUS Experiment attempts to move the local experimental region so that it encloses the optimum conditions. The Experiment ends when this occurs, or when the redirection limit or a movement limit is reached.

When movement stops, the results of the goodness of fit test and the equation for the fitted response surface are also reported in the Journal Window. Then, if not already done, the PLUS Experiment attempts to verify the predicted optimum conditions by running a simulation.

## 13.5 Operating Tips

We end the chapter with some helpful hints that can be useful as well as some common pitfalls to be avoided while you are engaged in an experimentation project.

◆ GPSS World uses User Variables to take the treatment level of each factor as values. When you create names for your variables, they must not clash with existing keywords, SNAs, or SNA classes. Names with more than two letters, followed by at least one digit or underscore, are safe. The factor names in experiments are simple User Variable names and take on the indicated treatment levels as values during the experiment.

◆ By default, GPSS World uses the # symbol to denote multiplication and reserves \* for SNA indirection. If you prefer to use \* for multiplication, you can switch the role of \* and # by a checkbox in the first page of the Model Settings. This is accessed through the Edit Menu.

◆ When you create your own names, they must not be the same as existing keywords or SNA Classes. Names including underscores are safe.

◆ Settings are inherited from the parent Model Object, including Function Key assignments (see the Edit menu). Optionally, the Experiment Generators load a Function Key with the appropriate CONDUCT command. The Screening Experiment Generator optionally loads F11, the Optimizing Experiment Generator, F12. After creating the Simulation, to run the experiment, just press the Function Key.

◆ The DoCommand library procedure allows an Experiment, or any Procedure called by an Experiment, to execute any GPSS World Command, except the CONDUCT Command.

◆ DoCommand translates its argument string in global context. Don't use temporary variable names in it. If you need to pass the values of local variables or arguments to the DoCommand library

procedure, you should build the string first using PLUS string procedures. That will cause the local variables to be completely evaluated before entering the scope of the DoCommand library procedure. The bottom line is that you must use values, not names, of temporary variables in the string you pass to the DoCommand Procedure. Global Variable names may be used, however. For examples, see the Run Procedure in Section 14.3.2.

- ◆ You can place INCLUDE Statements anywhere but inside a PLUS Procedure. Commonly, you would want to place all Commands after your last INCLUDE Statement.
- ◆ If you have more than one metric describing your observations, you will need to edit any generated experiment and define additional Result Matrices and ANOVA invocations.
- ◆ If find that in an ANOVA Table the F Statistics could not be calculated, try reducing the level of interactions in the analysis if you can. Otherwise, you will have to add replicates.
- ◆ Simulations can have any number of Experiments. Use the CONDUCT Command to invoke an experiment.
- ◆ Do not pass Procedure arguments or Temporary Variables to DoCommand without evaluating them first. This is discussed above. The DoCommand argument string is compiled in global scope, which cannot access local variables. You can create the string before the call to DoCommand, and then, in a separate step, pass it as an argument.
- ◆ Don't forget that PLUS statements can only appear inside PLUS Procedures.
- ◆ Inserting a screening Experiment does not remove an existing Experiment of the same name from your Model Object, but it does replace it in your Simulation Object because being inserted at the end of your Model it is registered last in the Simulation Object.
- ◆ You can replace any Procedure, including Experiments, in an existing simulation object. Since they are longer than simple Commands, you would probably use an INCLUDE statement as a custom command, which points to a Text Object containing the Procedure.
- ◆ Initialize your Result Matrix to UNSPECIFIED. The default value of a GPSS Matrix Entity is 0. Initializing to UNSPECIFIED prevents 0 values from being used as results. Doing the initialization outside the PLUS Experiment allows your experiment to be restartable (see the next tip).
- ◆ If you want your Experiments to be restartable, inside your PLUS Experiment before each run you should test the Result Matrix element for the "UNSPECIFIED" Data Type. Only if that test is true would you call the Run Procedure for that run of the Experiment. Then, when the Experiment is restarted after being HALTED (and possibly SAVED), completed runs are skipped. To see an example of this, refer to any generated Experiment.
- ◆ Generated Experiments are inserted at the bottom of your Model. An experiment replaces any like-named experiment registered previously. You are responsible for removing any old generated experiments you no longer need.
- ◆ You should always complete a thorough test of the base simulation before beginning the experimentation phase.
- ◆ When you get compilation errors, use the "Next Error" command in the Search Menu. When you get "run time" errors, use the "Go To Line" command in the Search Menu to go to the location where the error was detected.

[\[Table of Contents\]](#)

# Chapter 14. Error Messages

## 14.1 Introduction

There are many conditions that can prevent the normal completion of an operation. They can occur from operation of the menus, during Translation of a Model, or during the running of a Simulation.

Errors that occur as you initiate Commands and environmental operations appear immediately and prevent the operation from beginning. They require you to correct a condition or select a different operation. The other two categories of error generally are fixed by alterations in your Model.

Syntax errors can occur when you translate a Model or Command. To fix ones that occur in a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If an error occurs in an Include-file, you can get better diagnostic messages by temporarily placing those statements in a Model Object.

Errors that occur within the Simulation are called Error Stops. They cause the Simulation to enter the Halted state, and if an Experiment is in progress it is aborted. Usually Error Stop messages tell you where the error is to be corrected in the Model, but the **Search / Next Error** feature is not available for them. Instead, the Include-file number (the Model is file number 0) and the text-line number is given in the error message. The **Edit / Goto** menu command can usually be used to find and correct these errors.

Sometimes you will need to do some investigation in order to find the error. If the source of the error is not apparent, you may want to stop the simulation at a specific place and then use the SHOW Command and open Windows and Snapshots to see what happened.

For troublesome Error Stops you may want to use the Blocks window and the debug toolbar. Then you can place a Stop condition that causes a stop just before when the error condition occurs. Then use interactive SHOW commands to evaluate all suspected operands. When a condition occurs late in a simulation, you may find it convenient to save the Simulation object just before the simulated time when the error is due to occur. This can save time when you must rerun a late part of the simulation a few times in the diagnostic process. You can keep reopening that saved Simulation instead of rerunning the simulation to the error point.

The major error messages are listed next in alphabetic order with an additional explanation. Where you see XXX in the message it means that the error message contains a phrase that depends on the details of the error condition. Admittedly some messages arise from more than one context. In that case the explanation is based on what we think is the most common way the message can occur.

## 14.2 Messages and Explanations

### "A Simulation Data Stream failed."

A Data Steam operation failed. The details have been given in a previous message. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

### "A Transaction is blocked on an impossible condition."

The Active Transaction is attempting to wait for a condition that can never occur. Such a Transaction would never be able to enter the Block. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. Do not attempt to block on an Integrated User Variable. Use the Transaction generation thresholds in the INTEGRATE Command instead.

### "A Transaction tried to seize or preempt its own Facility."

The Active Transaction already owns a Facility that it again tries to seize or preempt. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

### "A Transaction which owns a Facility is attempting to terminate."

The simulation is attempting to terminate a Transaction that owns one or more Facility Entities. Transactions must release Facilities before they can be destroyed. You must

change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"A Transaction which was preempted at a Facility tried to seize or preempt it."**

The Active Transaction is preempted at a Facility that it is attempting to seize or preempt again. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Access is denied."**

A read or write operation was disallowed because another process is using the file.

**"Add not defined for this data type."**

The calculation failed because an operand has taken on a value which cannot be coerced to an arithmetic value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Addition overflow."**

The result of the operation exceeds the maximum allowable value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Alias Group Generator XXX cannot be Identity (i.e. null)."**

You cannot use the given expression as an Alias Group Generator. See Chapter 13 of *The GPSS World Reference Manual* on how to specify them.

**"An attempt to enlarge the event calendar failed."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"An Experiment cannot be called by a Procedure."**

Although an Experiment is a kind of PLUS Procedure, you must use a CONDUCT Command to invoke one.

**"Argument B must be greater than Argument A."**

When the arguments were evaluated, Argument B was not greater than Argument A. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Argument cannot be used as a TEMPORARY MATRIX: XXX"**

You cannot use the same name for both arguments and local variables.

**"Argument cannot be used as a TEMPORARY: XXX"**

You cannot use the same name for both arguments and local variables.

**"Arithmetic overflow."**

The result of the operation exceeds the maximum allowable value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Array ordinal error."**

The element of the array does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Assembly Count was not positive."**

The evaluation of an Assembly Count resulted in a 0 or negative value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Attempt to release an idle Facility."**

The Active Transaction is attempting to release a Facility that it neither owns nor has been preempted from. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Attempt to release an unowned Facility."**

The Active Transaction is attempting to release a Facility that is not owned. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Attempt to release more storage than existed."**

The Active Transaction is attempting to cause a Storage Entity to have more available capacity than was defined in the STORAGE Command. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Block index is not positive."**

An incremental value used to calculate Block locations is 0 or negative. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Block index is too big."**

A Block location was calculated that exceeds the number of Blocks in the Model. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Block labels cannot be used elsewhere."**

Block labels cannot be used as the names of non-Block entities.

**"Block limit is too small."**

The Block limit location comes before the starting Block location. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Call stack overflow. Possible circular reference. Increase Stack Size in Settings."**

The nesting depth of Procedure calls exceeded the maximum allowed. The maximum can be set in the Settings Notebook. It is possible that one of your PLUS Procedures gets called by one of the Procedures it calls, forming an endless cycle.

**"Can't get local PLUS array."**

A memory request was denied. The maximum memory request can be set in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive.

**"Can't get memory for text string."**

A memory request was denied. The maximum memory request can be set in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive.

**"Can't get Numeric Group Element."**

A memory request was denied. The maximum memory request can be set in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive.

**"Can't get Parameter Block."**

A memory request was denied. The maximum memory request can be set in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive.

**"Can't get Transaction Parameter."**

A memory request was denied. The maximum memory request can be set in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive.

**"Can't get Transaction queuing element."**

A memory request was denied. The maximum memory request can be set in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive.

**"Can't get Transaction."**

A memory request was denied. The maximum memory request can be set in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive.

**"Cannot find that function in the specified module."**

A dynamic function call found the desired executable file but did not find the function within it. Please consult Chapter 8 for the details on using **Call()** and related library procedures.

**"Character error in statement."**

An invalid character was found in a text line. Please correct it with the text editing capabilities of GPSS World.

**"Circular reference in expressions."**

The nesting depth in expression evaluation exceeded the maximum allowed. The maximum can be set in the Simulation Page of the Settings Notebook. It is possible that the GPSS Variable Entities and Procedure calls form an unending sequence that you must prevent.

This error is usually caused by a circular reference during the evaluation of an expression. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. You might want to assign an intermediate value to a Savevalue or Transaction Parameter.

**"Comma needed between coordinates."**

Matrix element coordinates must be separated by commas.

**"Conjugate block is not a MATCH Block."**

The Active Transaction is attempting to test for a match condition in a Block that is not a MATCH Block. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Cumulative probability Must be 1.0 at the last point."**

The last cumulative probability in the Function Follower Statement must be 1.0.

**"Data Stream Numbers must be strictly positive."**

You cannot use a nonpositive number for a Data Stream. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Datum has not been initialized."**

You are attempting to use a variable in an operation before it has been given an initial value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Decrement not defined for this data type."**

The calculation failed because an operand has taken on a value which cannot be coerced to an arithmetic value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Divide not defined for this data type."**

The calculation failed because an operand has taken on a value which cannot be coerced to an arithmetic value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Division overflow."**

The result of an operation exceeded the maximum value allowed. The result of the operation exceeds the maximum allowable value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Division range error."**

One of the operands has a value for which the operation is not defined. An attempt to divide by zero can cause this error. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"DoCommand received an invalid command."**

The string passed to the **DoCommand()** library procedure did not contain a valid GPSS World Command. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Duplicate Alias Group Generators are not allowed."**

You cannot use the given expression as an Alias Group Generator. It is a duplicate. See Chapter 13 of *The GPSS World Reference Manual* on how to specify them.

**"Entity is not a BVariable."**

The Active Transaction caused an attempt to evaluate a Variable Entity that does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Entity is not a Variable."**

The Active Transaction caused an attempt to evaluate a Variable Entity that does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Error in derivative evaluation found while integrating: "**

An error was detected evaluating the derivative while attempting to integrate the User Variable given in the message. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Error in Operand A of QUEUE."**

The evaluation of Operand A of a QUEUE Block resulted in a nonpositive value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Error in threshold evaluation found while integrating: "**

An error was detected evaluating a Transaction-generation threshold while attempting to integrate the User Variable given in the message. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Error occurred opening the Simulation."**

An error occurred when GPSS World tried to read the file associated with the Simulation Object. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system. The file may be corrupt.

**"Error occurred saving the simulation."**

An error occurred when GPSS World tried to write the file associated with the Simulation Object. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system. The file may be corrupt.

**"Error placing data on Clipboard. Please Try again."**

A clipboard operation failed. Retry the operation. If it persists you may have a problem associated with the operating system.

**"Error priming GENERATE Block with its first Transaction."**

GPSS World was not able to initialize the GENERATE Block with its first Transaction. There is an error in the calculation of the arrival time, or in one of the other operands of the GENERATE Block.

**"Error reading plain text Include-file."**

An error occurred when GPSS World tried to read the file associated with the Text Object. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system. The file may be corrupt.

**"Error Stop."**

An error condition has stopped the operation. Additional details can be found in accompanying messages.

**"Error: corrupt Simulation Object."**

An error occurred when GPSS World tried to read the file associated with the Simulation Object. The data in the file is of unknown format.

#### "Error: decreasing cumulative probabilities."

The cumulative probability in the Function Follower Statement cannot be decreasing.

#### "Expecting =."

A syntax error occurred. In its present state, the Translator is expecting to find a decimal point or period. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting Label."

A syntax error occurred. In its present state, the Translator is expecting to find a label. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting `DO◆."

A syntax error occurred. In its present state, the Translator is expecting to find ◆DO◆. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting `THEN◆."

A syntax error occurred. In its present state, the Translator is expecting to find ◆THEN◆. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting a ◆/◆."

A syntax error occurred. In its present state, the Translator is expecting to find a forward slash. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting a GPSS Verb."

A syntax error occurred. In its present state, the Translator is expecting to find a Verb. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting a Label or GPSS Verb."

A syntax error occurred. In its present state, the Translator is expecting to find a label or a Verb. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting a PLUS Statement."

A syntax error occurred. In its present state, the Translator is expecting to find a PLUS Statement. See Chapter 8 of *The GPSS World Reference Manual*. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting a Y value."

A syntax error occurred. In its present state, the Translator is expecting to find a Y value in a Function Follower Statement. If this is a Model translation you can use **Search /**

**Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting a comma."

A syntax error occurred. In its present state, the Translator is expecting to find a comma. If this is a Model translation you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting a formal argument."

A syntax error occurred. In its present state, the Translator is expecting to find an argument in a Procedure call. If this is a Model translation you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting a numeric value or Identifier."

A syntax error occurred. In its present state, the Translator is expecting to find a numeric value or identifier. If this is a Model translation you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting a numeric value."

A syntax error occurred. In its present state, the Translator is expecting to find a numeric value. If this is a Model translation you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting a right parenthesis."

A syntax error occurred. In its present state, the Translator is expecting to find a  $\text{)}\text{}$  character. If this is a Model translation you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting an EXPERIMENT, PROCEDURE or GPSS Statement."

A syntax error occurred. In its present state, the Translator is expecting to find one of the items in the error message. If this is a Model translation you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting an expression factor."

A syntax error occurred. In its present state, the Translator is expecting to find a valid factor of an expression. The formal grammar is given in the Appendix of *The GPSS World Reference Manual*. If this is a Model translation you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting left parenthesis."

A syntax error occurred. In its present state, the Translator is expecting to find a  $\text{(}\text{)}$  character. If this is a Model translation you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting left square bracket ([]."

A syntax error occurred. In its present state, the Translator is expecting to find a  $\text{[}\text{]}$  character. If this is a Model translation you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred

in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting right parenthesis."

A syntax error occurred. In its present state, the Translator is expecting to find a  $\text{)}\text{}$  character. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting right square bracket (])."

A syntax error occurred. In its present state, the Translator is expecting to find a  $\text{}\text{]}\text{}$  character. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Expecting semicolon."

A syntax error occurred. In its present state, the Translator is expecting to find a  $\text{;}\text{}$  character. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Experiment aborted."

A simulation was interrupted while a PLUS Experiment was under way. Usually this means that the Result Matrix does not have all the data intended. You must complete the remaining runs and then the Analysis of Variance. The Experiments generated automatically by GPSS World test for the completion of runs and may be restarted with a CONDUCT Command.

#### "Experiments cannot be nested."

PLUS Experiments can only be called directly by a CONDUCT Command.

#### "Exponent too large."

The exponent exceeds the maximum allowable value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Exponentiation overflow."

The result of the operation exceeds the maximum allowable value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Expression memory is full."

There is no more room for memorized expressions. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Factors must be consecutive beginning with A."

Experimental factors are denoted consecutively A, B, C, D, E, and F.

#### "Failure loading the specified module."

A dynamic function call could not find the desired executable file. Please consult Chapter 8 for the details on using **Call()** and related library procedures.

#### "File close error."

An attempt to close a file failed. This can be caused by a hardware error.

#### "File error."

An error prevented a file operation from completing properly.

#### "Floating point exception during integration."

A floating-point arithmetic operation encountered an overflow or underflow. You must scale the values of your variables so they don't exceed numerical limits.

#### "GPSS Verb cannot be used as operator or operand."

Syntax error. The Translator is expecting an operator or operand, but instead, encountered a GPSS Verb.

**"GPSS World Student Version GPSS Block limit was exceeded."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Goodness of fit test fails."**

The preliminary experimental response surface did not fit the data. Normally, an experiment generated by GPSS World will attempt to acquire more data, and continue, or apply a higher order model. Otherwise, you will need to apply a more sophisticated analysis to the data.

**"Help system failure."**

A failure occurred initializing the online Help system. The GPSS World Help file may be missing.

**"Identifier has not been defined."**

A reference occurred to a name that has not been defined. You must add the appropriate definition to the Model and/or Simulation Object.

**"Illegal attempt to make Queue entity content negative."**

A Transaction tried to enter a QUEUE or DEPART Block, which would have caused the content of a Queue Entity to be negative. This is not allowed. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Illegal character."**

The Translator has encountered an invalid character. Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

**"Illegal combination of operands in ALTER block."**

You have used an illegal combination of operands in an ALTER Block. Please consult Chapter 7 of this manual for details on using ALTER Blocks.

**"Illegal combination of operands in REMOVE Block."**

You have used an illegal combination of operands in a REMOVE Block. Please consult Chapter 7 of this manual for details on using REMOVE Blocks.

**"Illegal combination of operands in UNLINK block."**

You have used an illegal combination of operands in a REMOVE Block. Please consult Chapter 7 of this manual for details on using REMOVE Blocks.

**"Illegal operator combination."**

You have used an illegal combination of operands in a Block. Please consult Chapter 7 of this manual for details on using Blocks

**"Illegal SNA class in COUNT or SELECT block."**

You have used an illegal SNA class in operand E of a COUNT or SELECT Block. Please consult Chapter 7 of this manual for details on using these Blocks.

**"Include-file files must be Text Objects (.txt)."**

Include-file files must be plain text files with a .txt filename extension.

**"Incompatible formats. The archive and this version of the GPSS World software are using different object formats."**

You are attempting to read a file associated with a GPSS World Object of a different format than the GPSS World software you are using.

For Simulation or Report Objects, you should recreate them from the Model Object using the new version of GPSS World.

For Model Objects, use the Windows clipboard with the old version of GPSS World to create plain text files for each Model. Then, use the Windows clipboard with the new version of GPSS World to create new Model Objects from the plain text files.

#### "Incorrect Keyword."

You have used a keyword that does not belong where you tried to use it. Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Incorrect Matrix shape."

GPSS World has encountered a Matrix of the wrong shape as it attempts to fit a response surface to data.

If the MATRIX Statement was generated by a GPSS World Automatic Experiment Generator and has not been altered, please report this problem. To do so, please complete an error report on the GPSS World Internet Web site at [MinutemanSoftware.com](http://MinutemanSoftware.com).

#### "Increment not defined for this data type."

The calculation failed because an operand has taken on a value which cannot be coerced to an arithmetic value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Indirect SNA not allowed here."

You cannot use a System Numeric Attribute with an indirect entity specifier as this operand. Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Indirect addressing is discussed in Section 3.4.2 of *The GPSS World Reference Manual*.

#### "Integer division overflow."

The result of the operation exceeds the maximum allowable value. Possible division by zero. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Integer division range error."

One of the operands has a value for which the operation is not defined. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. This error is caused by an attempt to divide by integer zero.

#### "Integration tolerance cannot be maintained for: XXX"

During the integration of the variable given in the error message, a reduction in integration ministep size was not sufficient to bring the local error below the error tolerance. You must either increase the Error Tolerance in the Simulate Page of the Settings Notebook, or you must create your own integration method using PLUS Procedures. It is possible that a derivative of integration variables is too large in magnitude.

#### "Integration variable was not initialized: XXX"

When an attempt was made to integrate the variable named in the error message it was found to have no initial value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Internal compiler error. Unknown source type."

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at [MinutemanSoftware.com](http://MinutemanSoftware.com).

#### "Internal depth error."

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at [MinutemanSoftware.com](http://MinutemanSoftware.com).

#### **"Invalid \$ character in name."**

Your Statement has a syntax error. An invalid character was detected in an identifier. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Names are discussed in Section 3.5 of *The GPSS World Reference Manual*.

#### **"Invalid attempt to enter a GENERATE Block."**

Transactions are not allowed to enter GENERATE Blocks, except for those created by the Block itself. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### **"Invalid attempt to skip a required operand."**

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid character in GPSS Statement."**

Your Statement has a syntax error. An invalid character was encountered. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid character in exponent."**

Your Statement has a syntax error. An invalid character was encountered. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid character in number."**

Your Statement has a syntax error. An invalid character was encountered. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid character."**

Your Statement has a syntax error. An invalid character was encountered. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid column coordinate. Must be name or integer."**

Your Statement has a syntax error. Matrix element indices must be names or integers. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid data type."**

The operation failed because an operand has taken on a value which cannot be coerced to an appropriate data type. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. .

#### **"Invalid decimal point in number."**

Your Statement has a syntax error. An extraneous decimal point was encountered. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid delimiter."**

Your Statement has a syntax error. An invalid delimiter character was encountered. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid delimiter in GPSS Statement."**

Your Statement has a syntax error. An invalid delimiter character was encountered. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid dollar character."**

Your Statement has a syntax error. An invalid  $\diamond \$ \diamond$  character was encountered. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid E character in number."**

Your Statement has a syntax error. An invalid  $\diamond E \diamond$  character was encountered. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid expression."**

An expression was not correctly formed using defined syntax. You must correct the syntax. The formal grammar is given in the Appendix of *The GPSS World Reference Manual*.

#### **"Invalid Function pair count."**

You have broken one of the rules regarding the definition of GPSS Function Entities. Please review Section 4.4 of *The GPSS World Reference Manual* and Chapter 6 on FUNCTION Statements and Function Follower Statements.

#### **"Invalid Function type."**

You have broken one of the rules regarding the definition of GPSS Function Entities. Please review Section 4.4 of *The GPSS World Reference Manual* and Chapter 6 on FUNCTION Statements and Function Follower Statements.

#### **"Invalid handle."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

#### **"Invalid identifier."**

Your Statement has a syntax error. An invalid identifier was encountered. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid indirect addressing."**

Your Statement has a syntax error. The indirect addressing is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Indirect addressing is discussed in Section 3.4.2 of *The GPSS World Reference Manual*.

#### **"Invalid indirect identifier."**

Your Statement has a syntax error. The indirect addressing is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Indirect addressing is discussed in Section 3.4.2 of *The GPSS World Reference Manual*.

#### "Invalid indirect identifier in GPSS Statement."

Your Statement has a syntax error. The indirect addressing is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Indirect addressing is discussed in Section 3.4.2 of *The GPSS World Reference Manual*.

#### "Invalid indirect integer in GPSS Statement."

Your Statement has a syntax error. The indirect addressing is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Indirect addressing is discussed in Section 3.4.2 of *The GPSS World Reference Manual*.

#### "Invalid indirect operand."

Your Statement has a syntax error. The indirect addressing is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Indirect addressing is discussed in Section 3.4.2 of *The GPSS World Reference Manual*.

#### "Invalid indirection character"

Your Statement has a syntax error. The indirect addressing is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Indirect addressing is discussed in Section 3.4.2 of *The GPSS World Reference Manual*.

#### "Invalid integer in GPSS Statement."

Your Statement has a syntax error. The integer is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Invalid keyword for this operand."

Your Statement has a syntax error. The keyword is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Invalid keyword."

Your Statement has a syntax error. The keyword is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid list sequence number."**

The List sequence number on the Function Follower Statement is incorrect. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid name in GPSS Statement."**

Your Statement has a syntax error. The name is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Names are discussed in Section 3.5 of *The GPSS World Reference Manual*.

#### **"Invalid negative number in GPSS statement."**

Your Statement has a syntax error. The negative number is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid number of Matrix indices."**

The dimension of the matrix implied by the number of indices in the matrix element specifier does not match that of the MATRIX Command. The two statements must imply the same Matrix dimension.

#### **"Invalid number of procedure arguments."**

The procedure invocation does not have the number of arguments required by the PLUS Procedure definition. The two statements must agree.

#### **"Invalid number."**

Your Statement has a syntax error. The number is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid operator."**

Your Statement has a syntax error. The operator is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid PLUS Procedure."**

Your Statement has a syntax error in the Procedure definition. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid procedure definition."**

Your Statement has a syntax error in the Procedure definition. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid procedure identifier."**

Your Statement has a syntax error. The procedure identifier is incorrect or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### **"Invalid quoted operator."**

Your Statement has a syntax error. The quoted operator is incorrect or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Invalid real number in GPSS statement."

Your Statement has a syntax error. A real number cannot be used in this position. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Invalid row coordinate."

Your Statement has a syntax error. Matrix element indices must be names or integers. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Invalid SNA Class."

Your Statement has a syntax error. The System Numeric Attribute Class is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

SNA Classes are discussed in Section 3.4 of *The GPSS World Reference Manual*.

#### "Invalid SNA entity specifier."

Your Statement has a syntax error. The System Numeric Attribute is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

SNAs are discussed in Section 3.4 of *The GPSS World Reference Manual*.

#### "Invalid SNA in GPSS Statement."

Your Statement has a syntax error. The System Numeric Attribute is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

SNAs are discussed in Section 3.4 of *The GPSS World Reference Manual*.

#### "Invalid SNA."

Your Statement has a syntax error. The System Numeric Attribute is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

SNAs are discussed in Section 3.4 of *The GPSS World Reference Manual*.

#### "Invalid string in GPSS Statement."

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Invalid string."

Your Statement has a syntax error. The string is invalid. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Invalid threshold arrival block found while integrating: "

An error occurred during the integration of the variable named in the message. A threshold crossing occurred but the destination Block number was not in the valid range. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Invalid use of SNA Class Specifier."

Your Statement has a syntax error. The System Numeric Attribute is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

SNAs are discussed in Section 3.4 of *The GPSS World Reference Manual*.

#### "Invalid XXX character in YYY."

Your Statement has a syntax error. The character given in the message is invalid or out of place in the expression. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Invalid XXX expression."

Your Statement has a syntax error. The expression given in the message is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Expressions are discussed in Section 8.2 of *The GPSS World Reference Manual*.

#### "Invalid XXX name."

Your Statement has a syntax error. The expression given in the message is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Names are discussed in Section 3.5 of *The GPSS World Reference Manual*.

#### "Invalid Y value."

The Y value in the Function Follower Statement is invalid or out of place. Function Follower Statements are discussed in Chapter 6.

#### "Line number must be a strictly positive integer."

Your Statement has a syntax error. The number is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Line numbers must begin in the first column."

Your Statement has a syntax error. The number is invalid or out of place. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "List Function argument is too large."

When the argument of a list function was evaluated it exceeded the size of the list. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Lock violation."

A file sharing violation has occurred. Retry the operation when the file is available.

#### "Logarithm range error."

The operand cannot be nonpositive. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"MAPI failure. Email was not sent."**

Your MAPI System could not be initialized or an error occurred when a message transmission was attempted. The message was not sent.

**"MAPI initialization failed. Email was not sent."**

Your MAPI System could not be initialized or an error occurred when a message transmission was attempted. The message was not sent.

**"Manual Simulation failure."**

The Manual Simulation Statement encountered an Error Stop. Additional details follow this message.

**"Matrix column cannot be system-assigned or Block label."**

The matrix column is not correctly specified. Change the Matrix element specifier in the offending Statement.

**"Matrix inversion failure."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"Matrix row cannot be system-assigned or Block Label."**

The matrix row is not correctly specified. Change the Matrix element specifier in the offending Statement.

**"Maximum depth exceeded: possible circular reference. Increase Max Depth in Settings."**

The nesting depth in an expression evaluation exceeded the maximum allowed. The maximum can be set in the Simulate Page of the Settings Notebook. It is possible that the GPSS Variable(s) and/or procedure calls form an unending sequence that you must change.

This error is usually caused by a circular reference during the evaluation of an expression. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. You might want to assign an intermediate value to a Savevalue or Transaction Parameter.

**"Memory error."**

A memory request was denied. The maximum memory request can be specified in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive(s).

**"Memory request denied."**

A memory request was denied. The maximum memory request can be specified in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive(s).

**"Minimum and maximum must be numeric."**

A minimum or maximum must have a numeric value. Please correct the Statement and retry the operation.

**"Maximum must be greater than minimum."**

You cannot have a minimum value greater than the maximum. Please correct the Statement and retry the operation.

**"Missing GPSS operand."**

Your Statement has a syntax error. The Translator is expecting to find the item in the message. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

**"Missing GPSS operator."**

Your Statement has a syntax error. The Translator is expecting to find the item in the message. In the Model Window, you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Missing SNA entity specifier."

Your Statement has a syntax error. The Translator is expecting to find the item in the message. In the Model Window, you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Missing Statement Label."

Your Statement has a syntax error. The Translator is expecting to find the item in the message. In the Model Window, you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Missing coordinates for MX Class SNA."

Your Statement has a syntax error. The Translator is expecting to find the item in the message. In the Model Window, you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Model translation failed; out of memory."

A memory request was denied. The maximum memory request can be specified in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive(s).

#### "Modulo division not defined for this data type. "

The calculation failed because an operand has taken on a value that cannot be coerced to an arithmetic value or an attempt was made to divide by zero. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Modulo division range error."

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Multiplication overflow."

The result of the operation exceeds the maximum allowable value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

#### "Multiple defined Block label definitions."

Your Statement has a syntax error. A Block Label can only be used for one Block. In the Model Window, you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Multiple label definitions: XXX"

Your Statement has a syntax error. The label given in the message was not uniquely defined. In the Model Window, you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Multiple argument definitions: XXX"

Your Statement has a syntax error. The argument given in the message was not uniquely defined. In the Model Window, you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

### **"Multiply not defined for this data type."**

The calculation failed because an operand has taken on a value which cannot be coerced to an arithmetic value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

### **"Multiway Experiment is Not Orthogonal. No F Statistic is available."**

The run results in the Result Matrix do not form an orthogonal experiment. You must either complete the experiment, or redefine it if the experiment itself is not orthogonal.

### **"Name has not been given a value."**

An uninitialized identifier was encountered. If you do not assign a value to a name in an EQU Statement, GPSS World assigns a distinct "system" number for use in defining GPSS Entities. Names with system assigned values may be used to name entities, but may not be used by themselves in expressions.

Do not confuse the value of a name with the result of an SNA. For example, if you define a Variable Entity named VAR1, you evaluate it by a reference to V\$VAR1 and not to VAR1. Here VAR1 serves as the entity specifier.

### **"Name too long."**

Your Statement has a syntax error. Names cannot exceed 200 characters. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

Names are discussed in Section 3.5 of *The GPSS World Reference Manual*.

### **"Negative A operand in SPLIT Block."**

The Active Transaction has caused the A Operand of a SPLIT Block to be evaluated with an invalid negative result. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

### **"Negative Storage request."**

When Operand B of an ENTER or LEAVE Block was evaluated, the storage amount requested was less than 0. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

### **"Negative time increment."**

The Active Transaction has caused a negative time increment to be calculated. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

### **"No data. An error occurred in the User Dialog."**

You must complete the required data field in the dialog.

### **"No room for conversion to floating point number."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

### **"Nonpositive List Function argument."**

When the argument of a list function was evaluated it was nonpositive. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

### **"Nonpositive modulus."**

The modulus must be positive. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

### **"Not enough memory for Model translation"**

An internal memory request was denied. Be sure you have enough swap space on your hard disk drive(s).

### **"Object code is corrupt."**

Please retranslate the Model. If the error persists please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"Operand B of QUEUE or DEPART is negative."**

The Active Transaction has caused the B Operand of a QUEUE or DEPART Block to be evaluated with an invalid negative result. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Operation has not been defined for this data type."**

The calculation failed because an operand has taken on a value which cannot be coerced to an arithmetic value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Optimum could not be predicted."**

A second order model could not be fitted to the data. Consequently the analysis has failed to predict an optimum value.

**"Out of memory."**

A memory request was denied. The maximum memory request can be set in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive.

**"Parameter number must be a positive integer."**

A nonpositive Transaction number was specified. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Path not found."**

An input or output operation attempted to use a path that does not exist in your file system. Please correct the situation and retry the operation.

**"Plot limit exceeded."**

You cannot add more plots to this Plot Window. You can open additional PLOT Windows.

**"PLUS Array has not been defined."**

A reference was made to PLUS Array that was not defined.

**"PLUS Procedure cannot be found."**

You must define a PLUS Procedure before you invoke it.

**"Probability cannot be less than 0."**

The probability cannot be less than 0 or greater than 1. Some probabilities are in parts-per-thousand. This is described in the associated Statement description in *The GPSS World Reference Manual*.

**"Probability cannot exceed 1.0."**

The probability cannot be less than 0 or greater than 1. Some probabilities are in parts-per-thousand. This is described in the associated Statement description in *The GPSS World Reference Manual*.

**"Procedure Statement limit exceeded."**

The Procedure is too large. You must break it into multiple procedures.

**"Procedure replaced. Multiple definition."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Reference to a non-existent Block entity."**

The Active Transaction has caused a reference to a Block entity that does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Reference to a non-existent Block entity."**

The Active Transaction has caused a reference to a Block entity that does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. You may have neglected to define the Block entity.

**"Reference to a non-existent Function entity."**

The Active Transaction has caused a reference to a Function entity that does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. You may have neglected to define the Function entity.

**"Reference to a non-existent Matrix entity."**

The Active Transaction has caused a reference to a Matrix entity that does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. You may have neglected to define the Matrix entity.

**"Reference to a non-existent Parameter."**

The Active Transaction has caused a reference to a Transaction Parameter entity that does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. Transaction Parameters are created in ASSIGN, MARK, SPLIT, and TRANSFER SUB Blocks.

**"Reference to a non-existent Savevalue entity."**

The Active Transaction has caused a reference to a Savevalue entity that does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. You may have neglected to define the Savevalue entity.

**"Reference to a non-existent Storage entity."**

The Active Transaction has caused a reference to a Storage entity that does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. You may have neglected to define the Storage entity.

**"Reference to a non-existent Table entity."**

The Active Transaction has caused a reference to a Table entity that does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. You may have neglected to define the Table entity with a TABLE or QTABLE Statement.

**"Reference to a non-existent Variable entity."**

The Active Transaction has caused a reference to a Variable entity that does not exist. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. You may have neglected to define the Variable entity with a VARIABLE, BVARIABLE, or FVARIABLE Statement.

**"REmove option was not specified."**

You have specified a destination Block without using the required RE option. Please consult the Block Statement description in Chapter 8 for more details.

**"REmove option was used with no destination."**

You must specify a destination Block location when you use the RE option. Please consult the Block Statement description in Chapter 8 for more details.

**"Saddle point detected. Critical point is not optimal."**

A second order model was fitted to the data but a saddle point was detected. Consequently, the analysis has failed to predict an optimum value.

**"Second order model fails. Failure to calculate optimum."**

A second order model could not be fitted to the data. Consequently the analysis has failed to predict an optimum value.

**"Second order model fails. Saddle point detected."**

A second order model was fitted to the data but a saddle point was detected. Consequently, the analysis has failed to predict an optimum value.

**"Setup error."**

An error occurred during GPSS World initialization. If it persists, please complete an error report on the GPSS World Internet Web site at [MinutemanSoftware.com](http://MinutemanSoftware.com).

**"Share.exe was not loaded, or a shared region was locked."**

This is an operating system error. If the problem persists, please consult the vendor of your operating system.

**"Sorry the Text Object Open failed."**

An error occurred when GPSS World tried to read the file associated with the Text Object. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system. The file may be corrupt.

**"Sorry, the Model open failed."**

An error occurred when GPSS World tried to read the file associated with the Model Object. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system. The file may be corrupt.

**"Sorry, the Model save failed."**

An error occurred when GPSS World tried to write the file associated with the Model Object. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system.

**"Sorry, the Report open failed."**

An error occurred when GPSS World tried to read the file associated with the Report Object. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system. The file may be corrupt.

**"Sorry, the Report save failed."**

An error occurred when GPSS World tried to write the file associated with the Report Object. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system.

**"Sorry, the Simulation has no Matrices defined."**

You cannot open a Matrix Window on the simulation because there are no Matrix entities in the Simulation Object.

**"Sorry, the Simulation has no Tables defined."**

You cannot open a Table Window on the simulation because there are no Table or Qtable entities in the Simulation Object.

**"Sorry, the Simulation open failed."**

An error occurred when GPSS World tried to read the file associated with the Simulation Object. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system. The file may be corrupt.

**"Sorry, the Simulation save failed."**

An error occurred when GPSS World tried to write the file associated with the Simulation Object. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system.

**"Sorry, the Text Object Save failed."**

An error occurred when GPSS World tried to write the file associated with the Text Object. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system.

**"Square root not defined for this data type."**

The calculation failed because an operand has taken on a value which cannot be coerced to an arithmetic value or which is negative. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Square root range error."**

The argument cannot be negative. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

### **"Standard Error is 0. Cannot perform F test."**

The calculation for the F test cannot be completed because the Standard Error of the experiment is 0. This makes all related statistical differences significant.

### **"Storage request exceeds total capacity."**

When Operand B of an ENTER Statement was evaluated, the storage request exceeded the total capacity of the Storage entity. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. You may wish to redefine the Storage entity.

### **"Subtract not defined for this data type. "**

The calculation failed because an operand has taken on a value which cannot be coerced to an arithmetic value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

### **"Subtraction overflow."**

The result of the operation exceeds the maximum allowable value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

### **"Syntax error in Label. "**

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

### **"Syntax error in Operand A. "**

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

### **"Syntax error in Operand B. "**

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

### **"Syntax error in Operand C. "**

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

### **"Syntax error in Operand D. "**

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

### **"Syntax error in Operand E. "**

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

### **"Syntax error in Operand F. "**

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

### **"Syntax error in Operand G. "**

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If

the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Syntax error in Operand H."

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Syntax error in Operator."

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Syntax error."

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Syntax error. XXX"

Your Statement has a syntax error in the expression given in the message. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Syntax error. Illegal use of Library Procedure name."

Your Statement has a syntax error. The name of a PLUS library procedure occurred in an invalid way. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Syntax error. Incorrect argument count."

Your Statement has a syntax error. A PLUS Procedure invocation has an argument count that does not match that of the Procedure definition. The name of a PLUS library procedure occurred in an invalid way. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Syntax error. Invalid PLUS Statement."

Your Statement has a syntax error in a PLUS Statement. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Syntax error. Invalid TEMPORARY MATRIX Identifier."

Your Statement has a syntax error in a PLUS TEMPORARY MATRIX Statement. An identifier is invalid. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Syntax error. Nonpositive dimension size."

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

#### "Syntax error. Too many matrix dimensions."

Your Statement has a syntax error. You have specified more dimensions in a Matrix entity than really exists. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error

occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

**"Syntax error. Unexpected end of file."**

The Translator expected to find more language elements but ran into the end of the Model. This error can occur when you have a string with unclosed double quotes.

**"Syntax error. Unidentified character."**

Your Statement has a syntax error. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

**"System error translating the interaction."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. A workable set for the principle block could not be found."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. An exception error occurred saving the Simulation."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. Bad DEQ call."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. Bad ENQ call."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. Bad FEC."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. Bad token string."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. Corrupt chain in DEQ."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. Corrupt chain in UnQQE."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. Dangling QQE."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. FEC corrupt."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. FUNCTION setup error."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. Internal exception."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. Internal overflow."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. Invalid indirect pointer."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. Invalid operand."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. No Next Block."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. No QQE."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"System error. QCB cache failure."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"Tab stops must be nonzero and increasing."**

Tab stops must be specified as positive strictly increasing integers.

**"Test expression does not have numeric or logic value. "**

You can only use a numeric value as the result of a TEST Block expression. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The XXX field is required. "**

Your Statement has a syntax error. The field specified in the message is invalid. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

**"The command is not valid. "**

Your Statement has a syntax error. Commands are discussed in Chapter 6 in *The GPSS World Reference Manual*.

**"The disk is full. "**

An attempt was made to write a file to a disk that did not have enough space.

**"The dynamic call must pass 0 or 1 argument."**

The dynamic call did not have the correct number of arguments. **Call()** and related library procedure invocations are described in Chapter 8 of *The GPSS World Reference Manual*.

**"The end of file was reached. "**

Your Statement has a syntax error. The Translator ran out of text before a statement was completed. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

This error can be caused when a string does not have a closing double quote character.

**"The entity label is missing."**

Your Statement has a syntax error. The Translator is expecting to find the item in the message. In the Model Window, you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

**"The exit code must be -1, 0 or +1."**

The Exit Command takes only these values as its argument.

**"The expression is not valid."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The expression must have a label."**

An expression cannot be viewed, plotted, or memorized without a label.

**"The file could not be found."**

An error occurred when GPSS World tried to read a file that the operating system could not find. The location of files is discussed in Chapter 2 of *The GPSS World Reference Manual*, and that associated with dynamic call invocations in Chapter 8 of *The GPSS World Reference Manual*.

**"The generators must be independent w.r.t. modulo multiplication. None can be the product of others."**

You have specified an Alias Group Generator that can be created from a combination of the other generators. They must be independent. Alias Groups are discussed in Chapter 13 of *The GPSS World Reference Manual*.

**"The j argument must be greater than the i argument."**

A procedure invocation used arguments that were not valid. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The location argument must be greater than 0.0."**

A procedure invocation used arguments that were not valid. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The Matrix has too many dimensions."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The Matrix index is not positive."**

The Active Transaction has caused a Matrix index to be evaluated that is 0 or is negative. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The Matrix index is too large."**

The Active Transaction has caused a Matrix index to be evaluated that exceeds the size of the dimension of that Matrix. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The Matrix offset is too big."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The max argument must be greater than the mode argument."**

A procedure invocation used arguments that were not valid. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The mean must be greater than 0.0."**

A procedure invocation used arguments that were not valid. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The mode argument must be greater than the min argument."**

A procedure invocation used arguments that were not valid. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The numeric conversion is not defined."**

An operation was attempted which required a numeric conversion that could not be done. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The nesting of Include-file files is too deep."**

Include-file files may be nested only to a maximum depth of 5.

**"The optimum type must be -1 for minimum or +1, for maximum"**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The p argument must be strictly between 0 and 1."**

A procedure invocation used arguments that were not valid. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The Random Stream number must be positive."**

A nonpositive Random Number stream was requested. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The redirection limit must be between 0 and 10, inclusively."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The replicate dimension is out of range."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The result Matrix has an invalid shape."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The result Matrix is incompatible with the Defining Contrasts."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The results of the Fractional Factorial Experiment are not complete."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The scale argument must be greater than 0.0."**

A procedure invocation used arguments that were not valid. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The shape argument must be greater than 0.0."**

A procedure invocation used arguments that were not valid. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The Simulation is still running and could not be saved."**

You must Halt the Simulation before saving it.

**"The Simulation was created by a GPSS World Commercial Version."**

The Student Version of GPSS World cannot open Simulation Objects created by the Commercial Version.

**"The Simulation was created by an upgraded version of GPSS World. This version cannot read it."**

You must create a new Simulation object with your current version of GPSS World.

**"The Simulation was not created by a Student Version."**

The Student Version of GPSS World cannot open Simulation Objects created by the Commercial Version.

**"The standard deviation must be greater than 0.0. "**

A procedure invocation used arguments that were not valid. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The string of Defining Contrasts is invalid. "**

An invalid argument was passed to the Effects procedure. If the string was created by a GPSS World generated experiment, please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"The system was unable to serialize part of the Simulation. "**

Internal error. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"The t argument must be greater than 0.0. "**

A procedure invocation used arguments that were not valid. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The upper count limit is too low."**

The Active Transaction has caused an operand to be evaluated that resulted in an entity number lower than the starting number. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"The User Key is not valid for this product. Please register to receive a new Key. "**

You need to acquire a valid User Key from Minuteman Software. The registration dialog can be accessed through **File / Register Software** in the menu of the Main Window of GPSS World.

**"There are no Transactions. Check Transaction limits and blocking."**

Either you have entered a statement that requires an Active Transaction and there is none, or all GENERATE statements have reached their creation limits.

If you limit the number of Transaction created by using Operand D in all GENERATE Statement returns to zero. You must provide another source of Transactions in the Simulation (SPLIT or GENERATE), you must increase the Termination Count in one or more TERMINATE Blocks, or you must reduce the Termination Count in the START Statement.

**"There are no errors in the list."**

The list of errors is empty. You may need to retranslate the model.

**"There are no more directory entries. "**

There are no more files in this directory (folder).

**"There are not enough runs defined in this Experiment. "**

The analysis program does not have enough data points to continue the analysis. You must complete the runs in the experiment. You must complete the runs in the experiment before analyzing the results.

**"There is no Stop here. Do you want to remove all Stops? "**

There is no stop condition on the currently selected Block. You have the option to remove all stop conditions, however.

**"There is no Transaction for the Parameter reference."**

A Statement was encountered which needed one or more Parameters of the Active Transaction for its evaluation. However, there is no Active Transaction.

**"There is no Transaction for this SNA evaluation."**

An SNA which a Transaction for its evaluation and there is none. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"There is no experiment. DoCommand() is Invalid."**

The **DoCommand()** library procedure cannot be invoked unless an Experiment is underway ♦ i.e. started by a CONDUCT Command. During an Experiment, even regular PLUS Procedures called directly or indirectly by the PLUS Experiment can invoke **DoCommand()**.

**"There is no such Transaction."**

The Transaction specified does not currently exist in the Simulation Object.

**"There is not enough computer memory to continue."**

A memory request was denied. The maximum memory request can be set in the Simulate Page of the Settings Notebook. Be sure you have enough swap space on your hard disk drive.

**"There must be an arrival block for the threshold expression."**

You must specify a Block that is to receive Transactions to be generated when the threshold is crossed.

**"There was a Simulation address translation error."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"There was a Simulation translation initialization Error."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at MinutemanSoftware.com.

**"There was a hardware error."**

Your operating system has reported that a hardware error prevented GPSS World from completing the operation.

**"There was an error opening XXX."**

An error occurred when GPSS World tried to read the file identified in the message. If it persists, you may have a hardware problem or one associated with the operating system. The file may be corrupt

**"There was an error processing XXX, in IO Stream Number YYY."**

An error occurred when GPSS World tried to read or write the file identified in the message on behalf of the Data Stream, also in the message. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system. The file may be corrupt

**"There was an error trying to set the file pointer."**

The operating system has reported that an error occurred when GPSS World tried to set a file pointer. Retry the operation. If it persists, you may have a hardware problem or one associated with the operating system. The file may be corrupt

**"This feature has not yet been implemented."**

The feature is not functional in this version of GPSS World.

**"Time range must be numeric greater than 0."**

The time range must be a positive number.

**"Tolerance must be between 0.000000001 and 0.1, inclusively."**

The integration tolerance must be in the given range.

**"Too many Block Entities for the Student Version."**

The Student Version supports only 150, or fewer, GPSS Blocks.

**"Too many open files."**

Your operating system is limiting the number of files that can be open at one time. You should close other applications and continue.

**"Too many significant digits."**

You have exceeded the maximum number of significant digits in a number. Real values are kept as double precision floating point numbers.

**"Trigonometric function domain error."**

The value of the operand is not in the range permitted for this function. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Truncate not defined for this data type."**

The calculation failed because an operand has taken on a value which cannot be coerced to an arithmetic value. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Unable to Stop the Simulation."**

Please close all other applications. Retry the operation. If this error persists, please complete an error report on the GPSS World Internet Web site at [MinutemanSoftware.com](http://MinutemanSoftware.com).

**"Unable to set up expression."**

An internal error has occurred. Please complete an error report on the GPSS World Internet Web site at [MinutemanSoftware.com](http://MinutemanSoftware.com).

**"Unclosed double quote."**

You must terminate the string with a closing double quote. Two consecutive double quotes within a string denote a double quote internal to the string.

If this is a Model translation you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

**"Undefined Label: XXX"**

Your Statement has a syntax error. The label in the message has not been defined. Check spelling. In the Model Window, you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

**"Unexpected Delimiter."**

Your Statement has a syntax error. In its present state the Translator expects to find a delimiter character such as a comma. In the Model Window, you can use **Search / Next Error or Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

**"Unknown token."**

The Translator has encountered a symbol it could not recognize. Possible corrupt file. If the problem persists, please complete an error report on the GPSS World Internet Web site at [MinutemanSoftware.com](http://MinutemanSoftware.com).

**"Use of a nonpositive entity number."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"Windows sockets initialization failed."**

GPSS World was unable to initialize its Internet interface. This could be a problem in your operating system.

**"You cannot use GENERATE in Manual Simulation. Use SPLIT."**

To create Transactions interactively you must use a SPLIT Block not a GENERATE Block.

**"You cannot use UNSPECIFIED to initialize a Logicswitch Entity."**

A syntax error occurred. Logicswitch Entities cannot take on a value of UNSPECIFIED. If this is a Model translation you can use **Search / Next Error** or **Search / Previous Error** to move the cursor directly to the faulty Statement. If the error occurred in an Include-file file, you can get better diagnostic messages by temporarily placing the statements in a Model Object.

**"You cannot wait on this SNA."**

The Active Transaction is attempting to wait for a condition that can never occur. Such a Transaction would never be able to enter the Block. You must change values, operands, or the flow of Transactions in the simulation to prevent this condition. Do not attempt to block on an Integrated User Variable. Use the Transaction generation thresholds in the INTEGRATE Command for that purpose.

**"You must Halt the Experiment first."**

The operation attempted requires that the current simulation be Halted first.

**"You must specify an SNA."**

You must specify an operand that is a valid System Numeric Attribute.

SNAs are discussed in Section 3.4 of *The GPSS World Reference Manual*.

**"You must specify arguments for this Experiment."**

You must change values, operands, or the flow of Transactions in the simulation to prevent this condition.

**"You must use a CONDUCT Command to invoke an Experiment."**

PLUS Experiments can only be started by a direct CONDUCT Command.

[\[Table of Contents\]](#)

# Appendix

This chapter presents a formal description of the commands and GPSS statements to be used with GPSS World. The first section contains the definitions of the elements of the language used in GPSS Statements the second contains the grammar for PLUS, the Programming Language Under Simulation, and the third a glossary of commonly used terms.

## 1.1. GPSS Grammar

*Null* ::= [ ]0,0

No entry. May be skipped.

*Uppercase* ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

Capital letters A through Z.

*Lowercase* ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

Lower case letters a through z.

*Alphabetic* ::= *Uppercase* | *Lowercase*

Either capital or lower case letters.

*Digit* ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Numerals 0 through 9.

*Comma* ::= ,

The comma symbol.

*Blank* ::= [ ]1,1

Blank. A space containing no printable character.

*Delimiter* ::= ; | *Comma* | *Blank*

A semicolon, comma, or blank. Delimiters are used to define the end of a field.

*NonnegInteger* ::= [Digit]1,15

Number of between 1 and 15 digits, inclusively.

*PosInteger* ::= [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9]1,1 [Digit]0,15

Number of between 1 and 15 digits, inclusively. First digit may not be 0.

*Fraction* ::= .[Digit]1,15

Decimal point followed by 1 to 15 digits inclusively.

*Sign* ::= Null | + | -

A plus or minus sign. If omitted, plus is assumed.

*Integer* ::= *Sign NonnegInteger*

An integer with an optional sign.

*NonnegNumber* ::= [Null | *NonnegInteger*]1,1 [nul | *fraction*]1,1

A nonnegative number which may include a fractional part.

*Exponent* ::= E *Sign* [*Digit*]1,3

E and an optional sign followed by 1, 2, or 3 digits.

*Number* ::= *NonnegInteger* [*Exponent*]0,1

A nonnegative number with an optional fractional part and optional exponent.

*SignedNumber* ::= *Sign Number*

A number which may be positive or negative, may have a fractional part, and may have an exponent. the sign need not be present.

*Name* ::= Alphabetic [ \_|Alphabetic | *Digit*]1,250 - [ Keyword | SNA | EntitySNAclass ]

An underscore, alphabetic character, or digit occurring 1 to 250 times and excluding reserved keywords. A name cannot be a verb or a valid SNA or SNA class. Lower case letters are automatically converted to upper case.

*StatementLabel* ::= *Name*

A statement label is a name used in the label field of a GPSS statement to give a name to a GPSS entity like a STORAGE or TABLE or to give a location name to a BLOCK.

*ResultMatrixName* ::= *Name*

A Result Matrix Name is passed to the ANOVA procedure to identify a Matrix Entity with special properties.

*AtomicSNA* ::= A1 | AC1 | C1 | M1 | PR | TG1 | XN1 | Z1

Atomic SNAs are System Numeric Attributes which do not need an entity specifier. For example, C1 is the relative system clock.

*EntitySNAclass* ::= [BV | CA | CC | CH | CM | CT | F | FC | FI | FN | FR | FT | FV | GN | GT | LS | MB | MP | MX | N | P | PR | Q | QA | QC | QM | QT | QX | QZ | R | RN | S | SA | SC | SE | SF | SR | SM | ST | SV | TB | TC | TD | V | W | X]1,1

Entity SNAs must be followed by an entity specifier in order to build a valid SNA. The entity specifier is the name or number of the entity, or when preceded by [\*], is the name or number of the parameter

of the Active Transaction containing the entity number. SNA class MX is excluded since it must also contain row and column specifiers (see *directmatrixSNA* and *indirectmatrixSNA* below).

*SimpleSNA* ::= *EntitySNAClass* [*PosInteger* | *\$Name*]1,1

A simple SNA is an entity SNA which does not use indirect addressing. The entity specifier must be a positive integer or a name preceded by *[ \$ ]*. SNA class MX is excluded since it must also contain row and column specifiers (see *directmatrixSNA* and *indirectmatrixSNA* below).

*DirectMatrixSNA* ::= MX[*PosInteger* | *\$Name*]1,1 ([[P]0,1 *Posinteger* | [P\$]0,1*Name*]1,1,[P]0,1*Posinteger* [P\$]0,1*Name*]1,1)

The MX is followed by a matrix entity specifier followed, in parentheses, by a row specifier and a column specifier. The row and column specifiers are positive integers, names or P class SNAs, separated by a comma. Indirect addressing is described below. The *[ \$ ]* is used as a separator when referencing a parameter by name in a row or column specifier.

*DirectSNA* ::= *AtomicSNA* | *SimpleSNA* | *DirectMatrixSNA*

A direct SNA is any SNA which doesn't use indirect addressing.

*IndirectMatrixSNA* ::= MX[[*PosInteger* | *\$Name* | \*[*PosInteger* | *\$*]0,1*Name*]1,1]1,1 ([[P]0,1*PosInteger* | [P\$]0,1*Name* | \*[*PosInteger* | *\$*]0,1*Name*]1,1 Comma [[P]0,1*PosInteger* | [P\$]0,1*Name* | \*[*Posinteger* | *\$*]0,1*Name*]1,1])1,1[*DirectMatrixSNA*]

The MX is followed by a matrix entity specifier followed, in parentheses, by a row specifier and a column specifier. The row and column specifiers are separated by a comma. At least one item must be specified using indirect addressing. If row and column specifiers are SNAs, they must be P class SNAs. A Setting is available for using the # character instead of the \*.

*SNA\*Parameter* ::= [BV | CA | CC | CH | CM | CT | F | FC | FI | FN | FR | FT | FV | GN | GT | LS | MB | MP | MX | N | P | PR | Q | QA | QC | QM | QT | QX | QZ | R | RN | S | SA | SC | SE | SF | SR | SM | ST | SV | TB | TC | TD | V | W | X]1,1 \*[*PosInteger* | *\$*]0,1*Name*]1,1

This is indirect addressing. The entity number is in a Transaction Parameter. The entity SNA class is followed by \* followed by the name or number of a parameter of the Active Transaction containing the entity number. A Setting is available for using the # character instead of the \*.

*IndirectSNA* ::= *SNA\*Parameter* | *IndirectMatrixSNA*

An SNA where the entity, row or column is specified by indirect addressing. A Setting is available for using the # character instead of the \*.

*MatrixSNA* ::= *DirectMatrixSNA* | *IndirectMatrixSNA*

Any MX class SNA.

*SNA* ::= *DirectSNA* | *IndirectSNA*

*Simpleterm* ::= *Name* | *SNA* | *SignedNumber*

Simplest items in an expression.

*MathFunction* ::= ABS | ATN | COS | EXP | INT | LOG | RND | SIN | SQR | TAN

Mathematical routines in the math library. these may be used to build expressions.

*MathFunctionTerm* ::= *MathFunction(Expression)*

A call to a math library subroutine with an expression as the argument.

*BinaryOperator* ::= + | - | # | / | \ | ^

Addition, subtraction, multiplication, division, integer division, and exponentiation. The [\*] symbol is reserved for indirect addressing. ]#] represents multiplication and ^ represents exponentiation.

*BlockName* ::= ADOPT | ADVANCE | ALTER | ASSEMBLE | ASSIGN | BUFFER | CLOSE | COUNT | DEPART | ENTER | EXAMINE | EXECUTE | FAVAIL | FUNAVAIL | GATE | GATHER | GENERATE | INDEX | INTEGRATION | JOIN | LEAVE | LINK | LOGIC | LOOP | OPEN | MARK | MATCH | MSAVEVALUE | OPEN | PLUS | PREEMPT | PRIORITY | QUEUE | READ | RELEASE | REMOVE | RETURN | SAVAIL | SAVEVALUE | SCAN | SEEK | SEIZE | SELECT | SPLIT | SUNAVAIL | TABULATE | TERMINATE | TEST | TRACE | TRANSFER | UNLINK | UNTRACE | WRITE

GPSS block names.

*LogicSwitchOp* ::= I | R | S

The operators used in the operator field of LOGIC blocks: I (invert), R (reset), or S (set).

*RelationalOp* ::= E | G | GE | L | LE | NE

The relational operators used in the operator field of certain GPSS blocks: E (equal), G (greater than), GE (greater than or equal to), L (less than), LE (less than or equal to), NE (not equal to).

*ConditionalOp* ::= E | G | GE | L | LE | MAX | MIN | NE

The conditional operators used in the operator field of certain GPSS blocks: E (equal), G (greater than), GE (greater than or equal to), L (less than), LE (less than or equal to), MAX (maximum), MIN (minimum), NE (not equal to).

*GateOp* ::= FNV | FV | I | LS | LR | M | NI | NM | NU | SE | SF | SNE | SNF | SNV | SV | U

The operators used in the operator field of GATE blocks. The operators specify a test condition and an entity type. The operators are:

- ◆ FNV Facility must be not available.
- ◆ FV Facility must be available.
- ◆ I Facility must be preempted.
- ◆ LS logicswitch must be set.
- ◆ LR logicswitch must be reset.
- ◆ M MATCH block must have a Transaction of the same assembly set as the Active Transaction.
- ◆ NI Facility must not be currently preempted.
- ◆ NM MATCH block must NOT have a Transaction of the same assembly set as the Active Transaction.
- ◆ NU Facility must not be in use.
- ◆ SE Storage must be empty.
- ◆ SF Storage must be full.

- ◆ SNE Storage must be not empty.
- ◆ SNF Storage must be not full.
- ◆ SNV Storage must be not available.
- ◆ SV Storage must be available.
- ◆ U Facility must be in use.

*LineNumber* ::= *Number*

*CommandName* ::= BVARIABLE | CLEAR | CONDUCT | CONTINUE | EQU | EXIT | FUNCTION | FVARIABLE | HALT | INCLUDE | INITIAL | INTEGRATE | MATRIX | QTABLE | REPORT | RESET | RMULT | SHOW | START | STEP | STOP | STORAGE | TABLE | VARIABLE

GPSS Commands do not define block entities. They set the conditions of the simulation and define other GPSS entities.

*Verb* ::= *BlockName* | *CommandName*

Statement specific syntax is discussed in Chapters 3, 6, 7, and 8.

## 1.2. PLUS Grammar

*Procedure* ::= **PROCEDURE** *ProcedureName* ( *FormalArgumentList* ) *Statement*

*Experiment* ::= **EXPERIMENT** *ProcedureName* ( *FormalArgumentList* ) *Statement*

*Statement* := CompoundStatement | IfStatement | IfElseStatement | WhileStatement | AssignStatement | GotoStatement | ReturnStatement | ProcedureCallStatement | LabeledStatement | TempDeclarationStatement

*LabeledStatement* ::= *Label* *Statement*

*Label* ::= *Name*

*TempDeclarationStatement* ::= *TempVarDeclareStatement* | *TempMatrixDeclareStatement*

*TemporaryVarDeclareStatement* ::= **TEMPORARY** *NameList* ;

*TempMatrixDeclareStatement* ::= **TEMPORARY MATRIX** *Name*[ *IntegerList* ] ;

*CompoundStatement* ::= **BEGIN** *StatementSequence* **END** ;

*FormalArgumentList* ::= *Name* [ , *Name* ] ... | NULL

*StatementSequence* ::= *Statement* [ *Statement* ] ... | NULL

*AssignStatement* ::= *Lvalue* = *Expression* ;

*IfStatement* ::= **IF** ( *Expression* ) **THEN** *Statement*

*IfElseStatement* ::=

**IF** ( *Expression* ) **THEN** *Statement* **ELSE** *Statement*

*WhileStatement*: **WHILE** ( *Expression* ) **DO** *Statement*

*GotoStatement* ::= **GOTO** *LabelName* ;

*ReturnStatement* ::= **RETURN** [ *Expression* ] ;

*ProcedureCallStatement* ::= *ProcedureCall* ;

*ProcedureCall* ::= *ProcedureName*( *ExpressionList* )

```

ProcedureName := LibraryProcedureName | Name

FormalArgumentList := Name [ , Name ] ... | NULL

StatementSequence := Statement [ Statement ] ... | NULL

ExpressionList := Expression [ , Expression ] ...

ExperimentName := Name

ModelName := Name

StringConstant := " [Character]..." 

ParenthesizedExpression := ( Expression )

Expression := Expression | SuperResult

:= SuperResult

SuperResult := SuperResult & InterResult

:= InterResult

InterResult := InterResult = SubResult

:= InterResult /= SubResult

:= SubResult

SubResult := SubResult < Comparator

:= SubResult > Comparator

:= SubResult <= Comparator

:= SubResult >= Comparator

:= Comparator

Comparator := Comparator + Term

:= Comparator - Term

:= Term

Term := Term # Factor

:= Term / Factor

:= Term \ Factor

:= Term @ Factor

:= Factor

A Setting is available for using the * character instead of the #.

Factor := Factor ^ Factor

:= - Factor

:= + Factor

:= Number

:= GenericDatum

:= TextString

:= ParenthesizedExpression

:= ProcedureCall

:= FunctionCall

:= SNA

```

```

GenericDatum := Name  

    := ArrayElement  

ArrayElement := Name [ ExpressionList ]  

LValue := Name  

    := Name [ ExpressionList ]

```

## Glossary

**Active Transaction** - That GPSS Transaction in a simulation which is at the Head of the Current Events Chain and is the next to attempt entry into a GPSS Block.

**Command File** - A Secondary Model File intended for interactive use. A Function Key loaded with an INCLUDE Statement can run a Control File with a single keystroke.

**Data Stream** - A sequence of text lines identified by a unique positive integer.

**Entity Label** - A Named Value used in the Label field of a GPSS Statement.

**GPSS Statement** - A GPSS Block Statement or a Command occurring in a single Text Line.

**Immediate Command** - A Command which when sent to a Simulation Object is performed immediately and is not placed on the Command Queue. HALT and SHOW are the only Immediate Commands. A HALT Command interrupts any running simulation and deletes all remaining Commands from the Command Queue.

**Internal Model File Number** - the 0-relative ordinal of the Model File in the order encountered by the Translator. The Model Object is assigned Model File Number 0.

**Interactive Statement** - A Model Statement sent to an existing simulation.

**Manual Simulation Statement** - an Interactive Block Statement, causing a temporary Block to be created, and causing the Active Transaction to attempt entry.

**Model** - A sequence of Model Statements.

**Model File** - A file containing Model Statements.

**Model Statement** - A GPSS Statement or a PLUS Procedure definition.

**Named Value** - A user created name used in a model.

**PLUS Procedure** - A PLUS PROCEDURE or EXPERIMENT Statement obeying the Syntax rules of the PLUS Language.

**Primary Model File** - A Model File brought into a Model Window of a GPSS World Session.

**Queued Command** - A Command which when sent to a Simulation Object is placed on a Command Queue behind all other Commands waiting to be performed. All Commands except HALT and SHOW are Queued Commands.

**Secondary Model File** - A Text Object Translated as a result of an INCLUDE Command.

**Session** - The Sequence of Events from the Opening to the Closing of the GPSS World process.

**Simulation** - The Result of Translation of a model, whose state can be advanced by a Simulation Object.

**Statement** - A GPSS Statement or a PLUS Statement.

**String** - A sequence of characters.

**Termination Count** - The state variable in each simulation that, when it becomes nonpositive, causes the simulation to end.

**Text Line** - A Sequence of up to 250 print characters, including blanks and tabs, terminated by, but not including, a CR LF sequence.

**Trace Indicator** - A Transaction state variable that causes a Trace Message to be created by each Block entry.

**Translator** - That part of the GPSS World Control Process which converts a model into a simulation.

**User Variable** - A Named Value not occurring in any GPSS Statement Label field.

[[Table of Contents](#)]