Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

A TUTORIAL ON THE SIMPLE_1 SIMULATION ENVIRONMENT

PHILIP COBBIN
SIERRA SIMULATIONS & SOFTWARE
303 ESTHER AVENUE
CAMPBELL, CALIFORNIA 95008

Overview:

SIMPLE_1 is an integrated modeling environment for interactive simulation using the IBM PC, XT, AT and true compatibles. The system is composed of a full screen editor, file management routines, compiling and run time systems for processing models written in SIMPLE_1. This tutorial will overview the modeling environment and provide an introduction to the SIMPLE_1 modeling language with an emphasis on the animation and advanced statistics features for simulation of manufacturing systems.

The SIMPLE_1 programming language supports modeling discrete and continuous systems world views using a network modeling orientation. Features of the language include the ability of the user to declare variables and statistics requirements, perform I/O operations on files and to animate simulation results in real time easily utilizing built in features of the language. SIMPLE_1 utilizes a repetitive approach to run control to facilitate goal seeking modeling. The language has features particularly suited for modeling assembly operations in manufacturing due to SIMPLE_1's unique approach to managing groups of entities in models of discrete systems.

At the time of presentation this tutorial will be augmented with slides to illustrate features of the SIMPLE_1 modeling environment.

SIMPLE_1 an integrated modeling environment:

Simulation projects inherently involve the integration of many activities and analysis skills to accomplish study objectives. In addition to the obvious requirement to construct, execute and analyze simulation results, data collection and analysis, model validation, and convincing decision makers as to the merits of simulation are important issues in simulation that emerging simulation software must address.

SIMPLE_1 has been implemented as an integrated modeling environment to facilitate simulation related activities by organizing the software into a set of integrated modules for performing the tasks of:

1) Editing models via a full screen text editor coupled to the compiler and run time system's error detection routines.

2) Managing disk files with a function key driven operating system to manage disk directories, active path names and drives, etc.

3) Collection and analysis of data via "toolbox" programs written in SIMPLE_1 to accomplish histogram generation, runs testing for correlation of data sets, Mann-Whitney U test, etc.

4) Interactive compilation of models with errors reported back to the full screen text editor.

5) Interactive execution of models with disk or keyboard input of program variables.

6) Animation of simulation results using SIMPLE_1 language elements to animate results as they occur during the simulation.

7) Interrupt model execution to allow an analyst to review and alter program variables during the simulation. This feature is particularly useful for verifying model execution.

8) On-line tutorials to facilitate learning the system and accessing key documentation quickly. Syntax and theory of operation data on language elements is available on-line.

Design requirements for the software included the avoidance of special hardware. Accordingly, SIMPLE_1 has no special hardware requirements for graphics adapters or special monitor requirements. The software runs on the IBM PC, XT, AT and like compatibles such as the AT&T PC 6300 equipped with either a monochrome or a color monitor.

SIMPLE_1 is an integrated modeling environment: hence it is more than a compiler and run time system for models written in SIMPLE_1. Basically, how SIMPLE_1 works is as follows: Upon execution of SIMPLE_1 the software displays an initial banner then a screen for displaying file related information and executing file commands. Disk directories can be reviewed and the default disk drive and DOS path name for model files can be changed. Files are loaded into the system for editing or compiling by pressing the F1 Function key and inputing the file name for the model. Editing, compiling and execution of the model are controlled using the key board's built in function keys. Figure 1 is a reproduction of a typical SIMPLE_1 environment display.

All of the various elements of the SIMPLE_1 modeling environment include banners at the top of the screen to show the user how to use the function keys. Much of the software and language documentation is available on-line via tutorial screens. At the top of Figure 1 the function keys: F1..F10, X, C, A, and ,D keys are listed along with the function associated with each key. To obtain the directory listing of files on the disk displayed in the figure, the "D" key was depressed. To load a file called "TV_IO.MDL" into memory the F1 was depressed and the file name entered. Once loaded into memory the file could be edited, or compiled & executed by depressing the F7, F9 and F10 keys.

```
SIERRA SIMULATIONS & SOFTWARE:          DATE:8/18/86      TIME: 12.41:46 pm

F1 -    GET FILE | F5 -   DELETE FILE | F9 - COMPILE MODEL | X - EXIT SYSTEM
F2 - REVISE FILE | F6 -   RENAME FILE |F10 -     RUN MODEL | A - CHANGE
F3 -   SAVE FILE | F7 -     EDIT FILE |  C - CHANGE DRIVE  |     DIRECTORY
F4 -  COPY FILES | F8 -      TUTORIAL |  D - LIST DIRECTORY|

          .            ..      TV_IO.MDL      CRANE.MDL  GT_EXMPL.MDL
   DATA.MDL      HISTO.INP    ROCKET.MDL   CONVEYOR.MDL    H_GRAM.MDL
  PILOT.EJT   SHORT_TV.MDL    TESTER.MDL   ANIM_TRL.MDL      DISK.MDL
MULTIPLE.SCH  ELEVATOR.MDL     WAFER.MDL   MULTIPLE.MDL   ABC_MFG.MDL
TRNSFR_L.MDL  MACH_BRK.MDL  MULTIPLE.DTA   Q_THEORY.MDL    SPRING.MDL
   CAFE.MDL        CPU.MDL      SHORT.IF      MODEL.BAT      CASH.DTA
   TEST.MDL       CASH.MDL      SHIP.MDL       TIME.DTA    TV_IO.REP

Disk Free Space: 24 K-Bytes

USING DISK DRIVE :C AVAILABLE MEMORY:  443872 DIRECTORY: \models13
FILE IN MEMORY :TV_IO.MDL
COMMAND:
```

Figure 1 - SIMPLE_1 main environment display.

The text editor is a full screen text editor coupled to the compiler and run time systems. When the compiler or run time system detects an error the editor is called after displaying a descriptive message of the problem encountered. Figure 2 is a sample reproduction of an editor display. From the initial environment the model was loaded into memory and the editor accessed by subsequently pressing the F7 key.

Information on various aspects of SIMPLE_1 are available through on-line tutorial screens. This feature facilitates debuging and learning the language quickly. Information on syntax and language elements are available through extensive on-line tutorials. Table 1 is a listing of SIMPLE_1 block types. The block summarized in the table can be used to open and close files, buffer keyboard input and perform discrete/continuous modeling of systems. In addition general purpose concepts like an IF-THEN-ELSE and a WHILE loop construct are included in the language. SIMPLE_1 also contains a number of built in function to perform arithmetic operations and access statistics and internal SIMPLE_1 variables. Table 2 is a listing of the functions built into the language.

When an error is detected by the SIMPLE_1 compiler or run time system a message describing the nature of the error is displayed. After displaying the error message SIMPLE_1 returns control to the editor with the cursor initially at the problem area. Once returned to the editor the usual routine is to consult with the on-line tutorials to check syntax or language concepts. Once the error is isolated and fixed in the the editor the user exits and re-compiles the revised model. SIMPLE_1's coupling of a full screen text editor with the compiler, run time, and tutorial systems provides an effective mechanism for program development and speeds up the learning process for beginners.

SIMPLE_1 simulations are interruptible. When a key is depressed during the simulation a menu is displayed at the top of the screen. The SIMPLE_1 interrupt sub-system allows the user to:

1) Halt/Continue the simulation
2) List the global variables in the model
3) Change or Look at the values of global variables
4) Review statistics on block utilization.

Figure 3 illustrates the interrupt menu and a listing of the global variables defined for a model.

Reviewing program variables during execution is particularly useful for program debugging and validation. In addition the interrupt feature can be used to allow the analyst to interrupt the simulation and change the values of variables to introduce problems into the model and to see the reaction of the system.

```
            BLK STRT:     0 COL:   0
FILE :TV_IO.MDL  LINE:    1 COL:   1 INSERT   SCRN LINE:     1 COL:  1
            BLK END :     0 COL:   0
F1 FIND STR | F4 DEL BLK    | F7 COPY BLK |     F10 WRITE BLK |CTRL F3 EDITOR
F2 DEL LINE | F5 MK BLK TOP | F8 MOVE BLK |Ctrl F1 TOP        |        TUTOR
F3 QUIT     | F6 MK BLK BOT | F9 READ BLK |CTRL F2 BOTTOM     |
DECLARE;
  GLOBALS: TIME_IN_SYSTEM OBSERVE_STATS;
  ENTITIES: TV(I):_CONTROL(1);
  DEF_SCREEN: PICTURE,1,1,80,16,YES;
+
                         TIME :
            +<-------------------------------------------+
            :                                             :
            :  !! TV INSPECT/ADJUSTMENT EXAMPLE !!        :
            :                                           *****
            :                                  +---------*  *
            :              INSPECT TV          :        *****
            :                                  :       ADJUSTOR
            :            **************        :       STATION
CREATE      :            *  NO.       *        :
TV --->-----+----------->* INSPECTORS *------>+------> PACKING
TOTAL:                   * BUSY =     *                TOTAL:
                         **************
```

Figure 2 - Full screen text editor display.

```
           HALT = Y     CONTINUE SIMULATION      = C
     LIST VARIABLES = L     CHANGE/LOOK AT VARIABLES = V
   LIST BLOCKS INFO = B     --- PRESS APPROPRIATE LETTER ---
    VARIABLE LABEL: TYPE
         STOP_TIME  SCALAR
        KILL_COUNT  SCALAR
         STEP_SIZE  SCALAR
    RELATIVE_ERROR  SCALAR
    ABSOLUTE_ERROR  SCALAR
          VELOCITY  SCALAR
            HEIGHT  SCALAR
           WT_FUEL  SCALAR
           BURNING  SCALAR
                 K  SCALAR
                 G  SCALAR
              RATE  SCALAR
        MAX_HEIGHT  SCALAR
            THRUST  SCALAR
         WT_ROCKET  SCALAR
              DRAG  SCALAR
```

Figure 3 - SIMPLE_1 interrupt menu and listing of variables for a sample program.

SIMPLE_1 "Toolbox" programs:

The SIMPLE_1 language and environment support development and use of a "Toolbox" approach to systems analysis. Programs can be written in SIMPLE_1 to collect and analyze data: real or synthetic data. Examples of programs written in SIMPLE_1 to provide a basic "tool kit" include programs to:

1) Collect timing data using the keyboard.
2) Construct Histograms of data sets (with or without a runs test to check for correlation in the data set).
3) Perform the Mann-Whitney test on two data sets.

The histogram program reads a data set and automatically sets histogram cell parameters. Using SIMPLE_1's character based graphics scheme the program interactively displays a histogram of the data set and allows the user to alter histogram parameters. This program illustrates the capabilities of the software for modeling and analyzing systems. The ability of the user to build "tool box" programs in SIMPLE_1 provides an open ended means of expanding the capabilities of the system. The open ended nature of SIMPLE_1 is a direct consequence of merging simulation language concepts with general purpose programing language concepts common in BASIC, Pascal, C, or FORTRAN.

SIMPLE_1: The Language

SIMPLE_1 employs a number of unique approaches to simulation from a language design point of view. The code is structured into five segments, one of which is a declaration phase. The other four phases of SIMPLE_1 describe the discrete and continuous nature of the model and run control aspects of model execution.

SIMPLE_1 uses a repetitive approach to run control employing a PRERUN and POSTRUN code sections to set initial conditions and analyze run results. Figure 4 illustrates SIMPLE_1's approach to running the user's model. The PRERUN section of the model is executed first to establish model parameters and run control limits such as the stopping time for the simulation. After execution of the PRERUN code the DISCRETE and/or CONTINUOUS sections of the model are processed. Using SIMPLE_1's repetitive approach to run control one can look at the results of a simulation to base decisions for parameter values of the next run.

Discrete event aspects of the model are defined using an activity on node network structure. The Continuous aspects of the system model are described using algebraic state equations which define variables overtime via first order differential equations. The Continuous aspects of the model are simulated using a Runge Kutta fourth order fixed step procedure with the step size assignable by the modeler. The discrete aspects of the model are processed via an event scheduling mechanism to sequence the flow of entities through blocks in the network model.

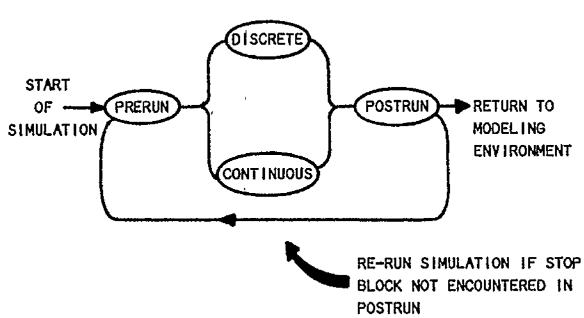


Figure 4 - Schematic of run control in SIMPLE_1

SIMPLE_1 is a declarative language in that the user can define variables. SIMPLE_1 variable identifiers can have up to 20 significant characters including the underscore to facilitate self documentation of the model. The language supports the declaration of the following classes of data structures:

1) Globally scoped reals: scalars and arrays with single or double subscripts.

2) Entities: Entities are declared by name with each type having their own unique number of attributes.

3) Screens: Windows and an associated character schematic to define a background for model animation.

4) Files: File variables to control reading and writing to files and logical devices.

Statistics on globally scoped variables of an observation or time persistent nature are collected automatically by appending key words to the variable declaration. When statistics are declared for arrays the statistics are collected for each element in the array; accordingly SIMPLE_1 models can collect extensive statistics on model variables.

Screens can be declared in SIMPLE_1 which define a character schematic to be used as a background over which animation of the model state is to be performed. SIMPLE_1's approach to formating of screen images emphasizes a "quick and dirty" approach to minimize modeling effort and overhead.

Entities are created by name in SIMPLE_1 and have their one unique attributes. Entities with identifiers like: CPU_BOARD and CHIP_SET can be declared in SIMPLE_1 each with differing attribute requirements. CPU_BOARD can be declared to have one attribute and CHIP_SET entities can have say five attributes associated with them. Entities are created by name in SIMPLE_1 models and can be brought together into groups. Entities formed into groups do not lose any of their attributes in SIMPLE_1. Manipulation of entity attributes by name simplifies referencing entities traveling in groups and tends to improve the self documentation aspects of SIMPLE_1 models.

The body of a SIMPLE_1 model is composed of five sections: DECLARE, PRERUN, DISCRETE, CONTINUOUS, and POSTRUN. Figure 5 illustrates the organization of SIMPLE_1 model code, the sequences of the segments describes the data structures first, followed by the code segments in their relative order of execution. The DECLARE section is used to define key model variables such as entities, screens, and so forth. The PRERUN and POSTRUN sections execute in a basic subroutine like manner much like BASIC or FORTRAN.

SIMPLE_1 employs seven (7) basic block types to define discrete and continuous models. The brevity of language concepts for discrete system modeling is due to the flexibility of the SIMPLE_1 CONDITIONS block. The network representation, syntax, and brief description of the CONDITIONS and the other SIMPLE_1 basic block types are summarized in TABLE 1.

Discrete system models involve construction of networks defining the flow in time of entities. Conceptually, entities are distinct individual objects that flow through blocks in the network model. Typically, entities are used in models to represent real objects: tools, parts, people, and so forth. The network model is used to define the interrelationship between entities and other elements of the system. In the most basic form, network models describe the processes to:

1) CREATE entities in the model

2) QUEUE entities (in waiting lines) until specified CONDITIONS are met.

BLOCK TYPE — SYMBOL — SYNTAX/FUNCTION

ACCEPT

LABEL ACCEPT,X,Y,VAR,L,H;

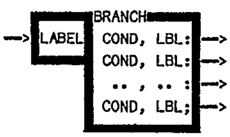
ACCEPT at a screen location a variable value.

ACTIVITY

LABEL ACTIVITY DURATION;

Engage arriving entities in an activity for a duration of time.

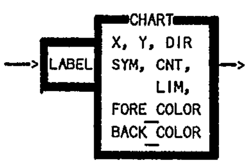
BRANCH



LABEL BRANCH
CONDITION/PROBABILITY,LBL1:
" " ,LBL2:
" " ,LBLN;

Route entities using a conditional or probabilistic criteria.

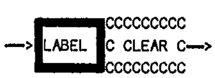
CHART



LABEL CHART,X,Y,DIR,SYM,CNT,LIM,
FORE_COLR,BACK_COLR;

CHARTS variable or expression value using ASCII characters to written CNT time to represent variable values graphically.

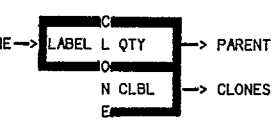
CLEAR



LABEL CLEAR;

CLEAR all statistics

CLONE



LABEL CLONE QTY,LBL;

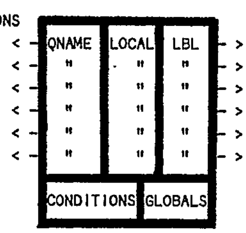
CLONE arriving entities and route them to block with label LBL.

CLOSE

LABEL CLOSE,FILEVAR1: ... :
FILEVARN;

CLOSE a file.

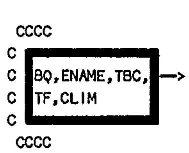
CONDITIONS



CONDITIONS,
GLOBAL,QNAME,LOCAL,LBL:
QNAME,LOCAL,LBL;

Monitor system state until specified conditions are met. CONDITIONS controls flow of entities from QUEUEs to other blocks in the model.

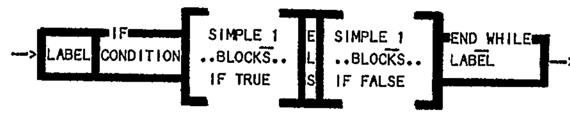
CREATE



CREATE,BQ,ENAME,TBC,TF,CLIM;

Creates groups of entities of entities of type ENAME.

IF-THEN-ELSE;



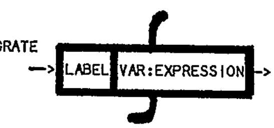
SYNTAX & FUNCTION:

LABEL IF CONDITION THEN;
....
SIMPLE_1 BLOCKS
....
LABEL ELSE;
....
SIMPLE_1 BLOCKS
....
LABEL END_IF;

IF CONDITION TRUE execute body of SIMPLE_1 blocks up to ELSE statement

IF CONDITION FALSE execute body of SIMPLE_1 blocks AFTER ELSE STATEMENT

INTEGRATE



Define Rate of change for VAR as EXPRESSION.

KILL

LABEL KILL,KILL_INCR;

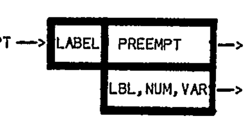
Eliminate entities from system.

OPEN

LABEL OPEN,FILEVAR A FILE EXT;

OPENs a file for subsequent read or write operations.

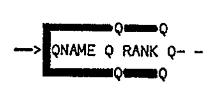
PREEMPT



LABEL PREEMPT,LBL,NUM,VAR;

PREEMPT entities engaged in the LBL labeled activity.

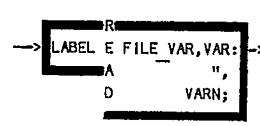
QUEUE



QNAME QUEUE,RANKING;

Hold entities in QUEUE until CONDITIONS are met.

READ



LABEL READ,FILEVAR,VAR1: ...:
VARN;

READ data from files.

REPORT

LABEL REPORT;

Generate a standard REPORT on simulation results.

RESET

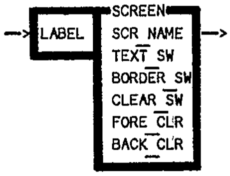
LABEL RESET;

Destroy all existing entities in the model.

TABLE 1 - SUMMARY OF SIMPLE_1 BLOCK TYPES

BLOCK TYPE	SYMBOL	SYNTAX/FUNCTION

SCREEN



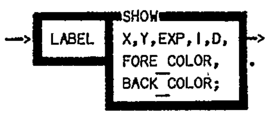
LABEL SCREEN,SCR_NAME,TEXT_SW,
BORDER_SW,CLEAR_SW,
FORE_COLR,BACK_COLR;

Activates SCR_NAMEd SCREEN and optionally resets the foreground and background colors in use on color systems.

SET

LABEL SET VAR:=EXPRESSION: ... :
: ... : VAR:=EXPESSION;
SET entity attribute and global variable values.

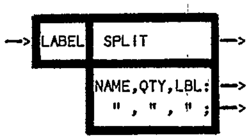
SHOW



LABEL SHOW,X,Y,EXP,I,D,
FORE_COLR,BACK_COLR;

SHOWs variable or expression value at the specified area of the active screen.

SPLIT



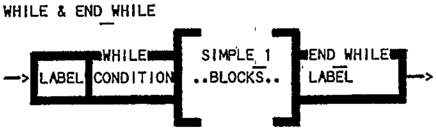
LABEL SPLIT:ENAME,QTY,LBL:...:
:...:ENAME,QTY,LBL;
SPLIT entities from arriving group and route to block with the label LBL.

STOP

LABEL STOP;

STOP simulation processing and return to modeling environment.

WHILE & END_WHILE

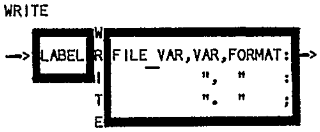


SYNTAX & FUNCTION:

LABEL WHILE,CONDITION;
....
SIMPLE_1 BLOCKS;
....
LABEL END_WHILE;

Executes a WHILE loop until the CONDITION expression is false

WRITE



LABEL WRITE,FILEVAR,VAR1,I,D:...
VAR2,I,D:...
VARN,I,D;

WRITE numeric or string constants to file.

TABLE 1 - CONTINUED

FUNCTION NAME	DESCRIPTION
ARITHMETIC:	
ABS	Absolute value
ARCOS	Arc cosine
ARCSIN	Arc sine
ARCTAN	Arc tangent
COS	Cosine
EXP	e taken to some power
LOG	Base 10 log
LN	Natural log
MAX	Maximum of two arguments
MIN	Minimum of two arguments
MOD	Modulus
ROUND	Round value to integer portion
SIN	Sine
SQRT	Square root
TAN	Tangent
BLOCK STATISTICS:	
AVE_NUM	Average activity level
COUNT	Count on number of times block encountered
MAX_NUM	Maximum activity level
MIN_NUM	Minimum activity level
NUM	Current number of entity groups in block
STD_NUM	Standar deviation for activity level
ENTITY GROUP ACCESS:	
NUM_ENTITY	Number of entities of a given type in group
VAL_ENTITY	Value of an attribute for non-pole entities
SET_ENTITY	Set function for non-pole entity attributes
INTEGRATED VARIABLES:	
LAST_STATE	Last state value
DERIV	Current derivative
LAST_DERIV	Last derivative
RANDOM NUMBERS:	
UNIFORM	Uniform distribution
NORMAL	Normal "
EXPON	Exponential "
TRIAG	Triangular "
LOGNORMAL	Lognormal "
POISSON	Poison "
SEED	Seed setting function
DISC_STEP	Discrete values & probabilities passed by an array
VARIABLE STATISTICS:	--- Observational statistics ---
OBSERVE_AVE	Average
OBSERVE_MIN	Minimum
OBSERVE_MAX	Maximum
OBSERVE_N	Number of observations
OBSERVE_STD	Standard deviation
VARIABLE STATISTICS:	--- Time persistant statistics ---
TIME_AVE	Average
TIME_STD	Standard deviation
TIME_MIN	Minimum
TIME_MAX	Maximum
TIME RELATED:	
STIME	Simulation time
SYS_TIME	Real system time
RUN CONTROL:	
KILL_COUNT	Termination count for run
STEP_SIZE	Integration step size
STOP_TIME	Stopping time for run
FILE RELATED:	
EOF	Return end of file status of a file

TABLE 2 - SUMMARY OF SIMPLE_1 FUNCTIONS

3) ACTIVITY: activities are undertaken by entities and involve the passage of time.

4) BRANCH: branching of entities between alternative pathways through the network model.

5) KILL: Disposal of entities in the system when they are no longer needed in the model.

6) SET variable values to describe changes in system state or entity attributes.

```
DECLARE;

   DECLARATION OF USER-DEFINED VARIABLES:
     1) GLOBAL variables
     2) ENTITIES
     3) SCREENS
     4) FILES

END;
PRERUN;

    INITIALIZE RUN:
      1) READ/WRITE information to files
      2) SET user defined variable values
      3) SET run limits: stopping time, entity termination limits
      4) CLEAR statistical accumulators

END;
DISCRETE;

   MODEL OF DISCRETE PROCESSES IN SYSTEM
     1) Statements based on an activity on node network scheme
     2) Character animation of simulation results
     3) READ/WRITE data to disk, keyboard, monitor etc.

END;
CONTINUOUS;

   MODEL OF CONTINUOUS PROCESSES IN SYSTEM:
     Statements use network scheme to define differential
     equation models of continuous system elements.

END;
POSTRUN;

   ANALYSIS OF RUN RESULTS/RUN CONTROL
     1) standard/custom reports on simulation results
     2) output of simulation results to files or devices
     3) CLEARing of statistical accumulators
     4) RESETing of model state
     5) Calculation of revised run parameters.
     6) STOPping program execution

END;
```

Figure 5 - Schematic Diagram of SIMPLE_1 Code

These six concepts plus a set of advanced modeling concepts comprise the basic building block processes used in SIMPLE_1 discrete system models. Detailed descriptions of all Simple_1 block concepts are available through the on-line tutorials.

CONDITIONS block: A key language element

The CONDITIONS block defines the state conditions required for entities to leave queues. In a basic queue/server relationship a CONDITIONS block is used to associate a specific QUEUE with an ACTIVITY block. Figure 6 illustrates a fragment from a network model describing the processing of computer mother boards through an insertion activity. A parameter of the CONDITIONS block in Figure 6 specifies that the number of active INSERTION activities must be less than one (idle) in order for a board to be released from the QUEUE labeled CPU_BOARDS.

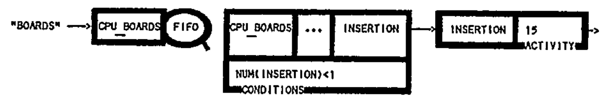


Figure 6 - Basic Queue/Server SIMPLE_1 network fragment for CPU assembly process model.

In most situations you start off modeling the main processes and add embellishments to capture additional constraints on system operation. In a model of a CPU assembly process we would start modeling with a basic simulation of the CPU's mother board flow through the production process.

The assembly aspects of system operation can have a dramatic bearing on the performance of the system and SIMPLE_1 has features especially useful for modeling assembly constraints in models of manufacturing processes. After construction of the initial model of the mother board's processing additional details can be added to the program to model assembly processes. Taking the basic queue/server code, a slight modification to the CONDITIONS block will model the assembly of the CPU_BOARD with a CHIP_SET entity. To add in an assembly constraint for the operation we would add a queue to store the required chip sets and augment the conditions block. The revised network fragment is illustrated in Figure 7. In the revised situation an entity must be in the CPU_BOARDS queue and the CHIP_SETS queue as well as an idle INSERTION activity in order for the CONDITIONS block to route the entities to the INSERTION activity. When the criteria for releasing the queues is met the conditions block routes the board and chip set entities to the insertion activity as a group. In the created group the board and chip set entities travel together and keep their unique attribute values, (they do not give up any attributes as a result of traveling together as a group). Figure 8 schematically illustrates the resultant entity group that is ultimately routed to the INSERTION activity.

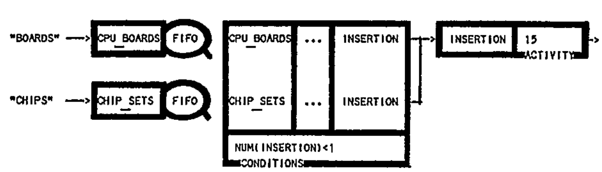


Figure 7 - Revised Queue/Server SIMPLE_1 network fragment to model assembly of CHIP_SET and CPU_BOARD entities.

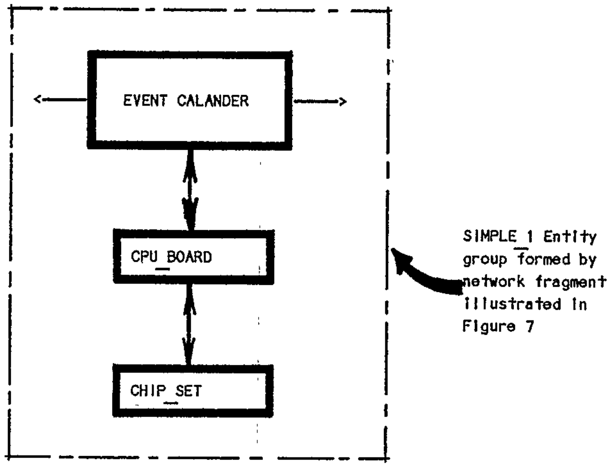


Figure 8 - Schematic representation of SIMPLE_1 entity group concept using the CPU assembly process as an example.

In addition to the basic modeling block types SIMPLE_1 models can employ blocks to manipulate groups of entities created with CONDITIONS blocks. The SPLIT block allows splitting specific entity types from a group and re-route them elsewhere and the CLONE block is useful for creation of exact duplicates of entity groups. As the name implies, the PREEMPT block is used to preempt the completion of activities by entities.

Notably absent in the SIMPLE_1 language is the concept of a resource. The reason SIMPLE_1 does not employ resources is that by it's nature, the CONDITIONS block can be used to model simplistic and complex resource situations. Key system resources in SIMPLE_1 models are typically modeled as entities that are grouped with "customer" entities while in use and SPLIT from the customer and routed to a queue when the resource entity becomes idle. The advantage inherent in modeling resources as a separate type of entity in SIMPLE_1 models is the ability to model explicitly the decision making processes of the resource. SIMPLE_1's handling of complicated resource situations is in a fashion a highly generalized version of the selector node concept for resource modeling employed in INS.

SIMPLE_1 employs four specialized blocks for run control purposes. A CLEAR block is used to control clearing statistical accumulators and a RESET can be used in the POSTRUN to eliminate all entities in existence in the discrete portion of the model. A standard report on system performance can be obtained using the REPORT block in the POSTRUN. The key run control block in SIMPLE_1 is the STOP block. The STOP block is used in the POSTRUN to halt model execution and return to the main SIMPLE_1 environment.

An original GPSS example of a basic TV inspection and adjustment situation illustrates how SIMPLE_1 code is written. In this example we have TV's arriving to be inspected by one of two available inspectors. After inspection good sets are routed to shipping and defective sets are routed to an adjusting station. At the adjusting station the sets are re-aligned by a single adjustor and routed back to the inspectors for re-testing. Using Schriber's GPSS TV inspection and adjustment example the SIMPLE_1 code for the model would be:

```
DECLARE;
   GLOBALS: TIME_IN_SYSTEM OBSERVE_STATS;
   ENTITIES: TV(1);
END;
PRERUN;
  SET STOP_TIME:=1440;
END;
DISCRETE;
          CREATE,1,TV,UNIFORM(3.5,7.5,1);
          SET TV(1):=STIME;
WAIT_INSP QUEUE,FIFO;
          CONDITIONS,
          NUM(INSPECT)<2,WAIT_INSP,,INSPECT;
INSPECT   ACTIVITY UNIFORM(6,12,1);
          BRANCH 0.85,PACK:
                 0.15,WAIT_ADJ;
WAIT_ADJ  QUEUE,FIFO;
          CONDITIONS,
          NUM(ADJUST)<1,WAIT_ADJ,,ADJUST;
ADJUST    ACTIVITY UNIFORM(20,40,1);
          BRANCH,WAIT_INSP;
PACK      SET TIME_IN_SYSTEM:=STIME-TV(1);
          KILL;
END;
CONTINUOUS; END;
POSTRUN;
  REPORT;
  STOP;
END;
```

The global variable TIME_IN_SYSTEM is declared with the key word OBSERVE_STATS appended to signal collection of statistics. When the set block near the bottom of the code assigns the value of TIME_IN_SYSTEM with the expression:

TIME_IN_SYSTEM:=STIME-TV(1)

The creation time for the TV and the current simulation time (STIME) are used to calculate the time in the system for the exiting TV. As a side affect of the the assignment SIMPLE_1 updates observatioanal statistics for TIME_IN_SYSTEM.

The CONDITIONS blocks in this model employ a built in function NUM which returns the current number of entity groups currently at a block in the model. NUM is one of an extensive number of built in SIMPLE_1 functions available to the modeler. Built in functions of the language provide access to arithmetic functions, random number generators etc. Table 2 is a summary of SIMPLE_1 functions.

<u>Input, Output and Animation:</u>

The SIMPLE_1 simulation language has input and output concepts for both file I/O and screen animation with the screen being updated while the model is running. SIMPLE_1 supports I/O operations using specialized block constructs. The input and output operations supported in the language are for two types of operations. Block constructs in the language control I/O to the screen or keyboard and to DOS. Screen I/O constructs include mechanisms for writing ASCII characters and numbers coupled with template images. The character and number based display formats of SIMPLE_1 combined with screen generation features of the language form a character based animation capability. In summary, SIMPLE_1 supports file and screen I/O Operations associated with:

1) SCREEN activation to display a text background.
2) SHOW block to display numeric values on a screen.
3) CHART block to display characters on a screen.
4) ACCEPT block for reading variable values from the keyboard.
5) READ and WRITE blocks for file input/output.
6) OPEN and CLOSE blocks for managing files during model execution.

VIDEO COLOR FIELDS

The screen I/O blocks: SCREEN, SHOW, and CHART have two optional fields to select the foreground and background colors to use on machines with a color monitor. The fields are optional and specify the foreground and background color to use when writing to the screen. Integer numbers are used to turn on specific colors as defined by the color numbers:

0: Black	6: Brown	11: Light Cyan
1: Blue	7: Light Gray	12: Light Red
2: Green	8: Dark Gray	13: Light Magenta
3: Cyan	9: Light Blue	14: Yellow
4: Red	10: Light Green	15: White
5: Magenta		

Revising the TV inspection and adjustment example illustrates the I/O concepts of SIMPLE_1 for both character animation of the simulation and generation of disk files. A screen will be used to form a schematic of the TV inspection system. SHOW and CHART blocks will be used to animate the state of the system using the schematic diagram of the system as a background. Figure 9 is a listing of the revised code for the TV repair model.

In the DECLARE section a CONTROL entity type has been added for managing the animation of the screen on 10 time unit intervals. The screen named PICTURE is associated with a schematic of the system A FILES declaration is made in the DECLARE section to define file variable OUT1. OUT1 will be used to store time in system observations.

During the PRERUN phase an OPEN block will open "HISTO.INP". When a TV completes processing the length of time spent in the system by the TV will be written to the file for post processing with the histogram analysis program implemented in SIMPLE_1 that is supplied with the software. A CLOSE block is used in the POSTRUN to close the disk file when the model is finished. Prior to returning to the SIMPLE_1 modeling environment menu a standard report on run results is obtained using the REPORT block. The REPORT block at run time allows reports to be written to the screen or to file.

A CONTROL entity is created every 10 time units in the DISCRETE section to manage updating the screen. The CONTROL entity executes a series of SHOW and CHART blocks. The SHOW blocks are employed to write numbers for the time, queue sizes etc. The CHART blocks are used to write ASCII characters. The number of ASCII characters written by the CHART block is used to graphically represent the number of busy inspectors and adjusters in the system. In effect, the animation of simulation results using the CONTROL entity causes "SNAP SHOTS" of the system to be taken on fixed time intervals. Alternatively, SHOW and CHART blocks can be inserted between ACTIVITY and QUEUE blocks to update the screen as specific portions of the system change state. This alternative method produces screen results that are generally more active and representative of the activities being simulated however, additional coding overhead is generally required.

```
DECLARE;
  GLOBALS: TIME_IN_SYSTEM OBSERVE_STATS;
  ENTITIES: TV(1): CONTROL(1);
  DEF_SCREEN: PICTURE,1,1,80,16,YES;
+
                         TIME :
           +<------------------------------------------+
           :                                           :
           :  !! TV INSPECT/ADJUSTMENT EXAMPLE !!      :
           :                                         *****
           :                                 +--------*   *
           :             INSPECT TV          :        *****
           :                                 :       ADJUSTOR
           :            **************       :       STATION
  CREATE   :            *   NO.      *       :
  TV --->----+--------->* INSPECTORS *----->+-----> PACKING
  TOTAL:                * BUSY =     *               TOTAL:
                        **************

+
FILES: OUT1,WRITE;
END;
PRERUN;
    OPEN,OUT1 AS HISTO INP;
    SCREEN,PICTURE,,,,15,0;
    SET STOP_TIME:=1440;
END;
DISCRETE;
 CREATE,1,CONTROL,10,10;
   SHOW,36,2,STIME,7,2,12,0;
   CHART,27,11,4,177,NUM(WAIT_INSP),12;
   CHART,57,6,4,177,NUM(WAIT_ADJUST),10;
   CHART,38,13,4,178,NUM(INSPECT),2;
   CHART,61,7,4,178,NUM(ADJUST_OP),2;
   SHOW,65,13,COUNT(PACK),3,0;
 KILL;
            CREATE,1,TV,UNIFORM(3.5,7.5,1);
            SET TV(1):=STIME;
WAIT_INSP   QUEUE,FIFO;
             CONDITIONS,NUM(INSPECT) < 2,
             WAIT_INSP,,INSPECT;
INSPECT     ACTIVITY UNIFORM(6,12,1);
            BRANCH 0.85,PACK:
                   0.15,WAIT_ADJUST;
WAIT_ADJUST QUEUE,FIFO;
            CONDITIONS,NUM(ADJUST_OP) < 1,
            WAIT_ADJUST,,ADJUST_OP;
ADJUST_OP   ACTIVITY UNIFORM(20,40,1);
            BRANCH ,WAIT_INSP;
PACK        SET TIME_IN_SYSTEM := STIME-TV(1);
            WRITE,OUT1,TIME_IN_SYSTEM,8,3:/;
            KILL;
END;
CONTINUOUS;
END;
POSTRUN;
  CLOSE,OUT1;
  REPORT;
  STOP;
END;
```

Figure 9 - Revised SIMPLE_1 code for TV repair model.

Running this example will produce the file: HISTO.INP which contains the individual time in system observations for TVs. Using a histogram program written in SIMPLE_1 a runs test was performed on the data and histogram generated. The histogram results are illustrated in figure 10. The report generated by the REPORT block was saved to a disk file and is reproduced in Figure 11.

```
RELATIVE      ENTER 1 TO RETURN TO MENU: ?          CELL UPPER   #    FREQUENCIES
FREQUENCY                                            NO. LIMIT OBS.  REL.    CUM.
  0.2897 :     #                                      1   6.000   0 0.0000 0.0000
  0.2607 :     #                                      2   9.000  57 0.2262 0.2262
  0.2317 :     #                                      3  12.000  73 0.2897 0.5159
  0.2028 :   # #                                      4  15.000  51 0.2024 0.7183
  0.1738 :   # # #                                    5  18.000  26 0.1032 0.8214
  0.1448 :   # # #                                    6  21.000  11 0.0437 0.8651
  0.1159 :   # # #                                    7  24.000   3 0.0119 0.8770
  0.0869 :   # # # #                               #  8  27.000   0 0.0000 0.8770
  0.0579 :   # # # #                               #  9  30.000   0 0.0000 0.8770
  0.0290 :   # # # # #                             # 10  33.000   0 0.0000 0.8770
  0.0000 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 11  36.000   1 0.0040 0.8810
          1 2 3 4 5 6 7 8 91011121314151617181920  12  39.000   0 0.0000 0.8810
                   --- CELL NUMBER ---             13  42.000   0 0.0000 0.8810
                                                   14  45.000   2 0.0079 0.8889
                   ---  STATISTICS ---             15  48.000   1 0.0040 0.8929
          AVERAGE .............. :    24.0069      16  51.000   0 0.0000 0.8929
          STD DEVIATION ........ :    40.5338      17  54.000   2 0.0079 0.9008
          MINIMUM .............. :     6.1620      18  57.000   0 0.0000 0.9008
          MAXIMUM .............. :   367.6200      19  60.000   0 0.0000 0.9008
          NUMBER OF OBSERVATIONS :        252      20       1  25 0.0992 1.0000
```

Figure 10 - Histogram generated from data created by TV model. Results were obtained using a 160 line program written in SIMPLE_1.

```
                              SIMPLE_1

                     SIERRA SIMULATIONS & SOFTWARE

                  (C) Copyright 1985 Phillip Cobbin
                         All Rights Reserved

SUMMARY REPORT FOR: tv_io.MDL

GENERATED ON: 8/16/86  10.48:46 pm

COMMENT: Samplo standard SIMPLE_1 summary report for TV model

              SUMMARY REPORT:  BLOCK STATISTICS

              SIMULATED TIME: STIME =   1.4400000000E+03
              STATISTICS CLEARED AT :   0.0000000000E+00

      BLOCK LABEL        TYPE    AVERAGE    STD DEV  MIN  MAX CRNT  CNT
-------------------+-----------+----------+----------+----+----+----+----+
         WAIT_INSP:      QUEUE:     0.574:     0.772:   0:   4:   2: 299:
           INSPECT:   ACTIVITY:     1.847:     0.366:   0:   2:   2: 297:
       WAIT_ADJUST:      QUEUE:     1.163:     1.123:   0:   4:   2:  43:
         ADJUST_OP:   ACTIVITY:     0.834:     0.372:   0:   1:   1:  41:
              PACK:        SET:     0.000:     0.000:   0:   1:   0: 252:
-------------------+-----------+----------+----------+----+----+----+----+

              SUMMARY REPORT: OBSERVATIONAL STATISTICS

              SIMULATED TIME: STIME =   1.4400000000E+03
              STATISTICS CLEARED AT :   0.0000000000E+00

    VARIABLE LABEL           TYPE  AVERAGE  STD DEV   MIN   MAX CRNT NO.
-------------------+---------------+--------+--------+-----+-----+----+----+
    TIME_IN_SYSTEM: SCALAR         :  24.007:  40.534: 6.2:367.6:10.6: 252:
-------------------+---------------+--------+--------+-----+-----+----+----+
```

Figure 11 - SIMPLE_1 standard summary report generated by TV model.

SIMPLE_1 will model continuous systems definable as a set of first order differential equations. A simple rocket model illustrates SIMPLE_1's approach to continuous modeling. The height of the rocket attained over time will be integrated and is based upon the initial fuel load of the rocket. In this example we would define velocity height, weight etc. in the declare section. The SIMPLE_1 key word INTEGRATED follows the declaration of variables whose values are obtained by numerical integration. SIMPLE_1 integrates continuous variables using a Runge-Kutta fourth order fixed step procedure. The SIMPLE_1 code for this example is illustrated in Figure 12.

```
DECLARE;
  GLOBALS:
    VELOCITY INTEGRATED: HEIGHT INTEGRATED: WT_FUEL INTEGRATED:
    BURNING: K: G: RATE: MAX_HEIGHT: THRUST: WT_ROCKET: DRAG;
  ENTITIES: CONTROL(2);
  DEF_SCREEN: PICTURE,1,1,80,23,YES;
+
                           TIME :
    15 :
    14 :                                          ROCKET MODEL
    13 :
    12 :                                          VELOCITY     :
    11 :                                          HEIGHT       :
  H 10 :                                          MAX HEIGHT   :
  E  9 :
  I  8 :                                         INITIAL FUEL
  G  7 :                                         WT (500-1500) :
  H  6 :
  T  5 :
     4 :
     3 :
     2 :
     1 :
       +-----+-----+-----+-----+-----+-----+-----+-----+
            20    40    60    80   100   120   140   160

                    --- Time ---

+
END;
PRERUN;
  SET STOP_TIME    := 150:   STEP_SIZE :=  1.0:  WT_ROCKET  :=  300:
      BURNING      :=  20:   K         := 0.05:  G          := 9.81:
      HEIGHT       :=   0:   VELOCITY  :=    0:  MAX_HEIGHT :=    0:
      THRUST       :=3500;
  INTEGRATE WT_FUEL:0;   INTEGRATE VELOCITY:0;   INTEGRATE HEIGHT:0;
  SCREEN,PICTURE,1,1,1,15,0;
  SCREEN,PICTURE,0,0,0,12,0;
  ACCEPT,65,11,WT_FUEL,500,1600;
END;
DISCRETE;
CREATE,1,CONTROL,2.0;
  SHOW,36,2,STIME,7,0,11,0;     SHOW,66,6,VELOCITY,7,1;
  SHOW,66,7,HEIGHT,7,1;         SHOW,66,8,MAX_HEIGHT,7,1;
  CHART,7+STIME/4,18-ROUND(HEIGHT/1000),1,35,1,1,12,0;
KILL;
END;
CONTINUOUS;
  SET MAX_HEIGHT  :=  MAX(MAX_HEIGHT,HEIGHT):
      DRAG        :=  K*VELOCITY*ABS(VELOCITY):
      WT_FUEL     :=  MAX(0,WT_FUEL):
      THRUST      :=  THRUST*(WT_FUEL>0);
  INTEGRATE WT_FUEL   : -BURNING*(THRUST>0);
  INTEGRATE VELOCITY  : G*(THRUST-DRAG)/(WT_ROCKET+WT_FUEL)-G;
  INTEGRATE HEIGHT    : VELOCITY;
END;
POSTRUN;
  STOP;
END;
```

Figure 12 - SIMPLE_1 model of a simple ROCKET.

A benefit of SIMPLE_1's DECLARE section is the ability to define and use variables with identifiers related to the physics of the problem such as height, velocity, drag, etc.

In this model the PRERUN establishes the initial state variables prior to the run. A discrete section is used to periodically update the monitor to display the rocket's state over time both numerically and using the character graphics capabilities of the language. Figure 13 illustrates the information displayed on the monitor while execution of the model is progressing.

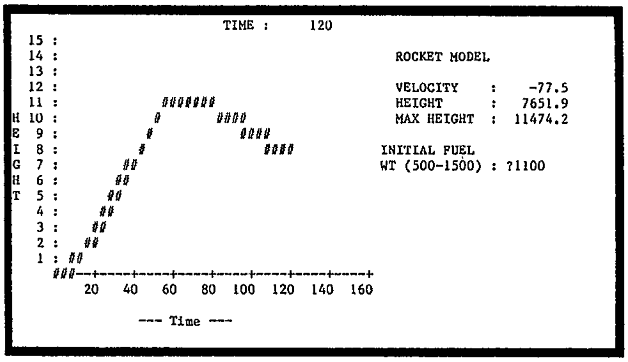


Figure 13 - Screen display during execution of rocket model.

Applications of SIMPLE_1:

Since announcement of SIMPLE_1 at the 1985 Winter Simulation Conference held in San Francisco, SIMPLE_1 has been applied in manufacturing, academia, and by the United States Military. Applications of SIMPLE_1 to date have ranged from manufacturing systems, robotics justification, health care systems, emergency planning, and analysis of logistic support systems.

Summary

SIMPLE_1 has a number of innovative features not found in current simulation software. The system combines a full screen editor with compilation and run time systems to speed up the edit-debug cycles involved in model building. The language supports a "tool box" ability whereby support programs can be written in SIMPLE_1 to post process simulation data. SIMPLE_1 utilizes a built in capability to animate simulation results using a character graphics methodology which stresses a "quick and dirty" approach to model animation. The language supports reading and writing of data sets via standard ASCII text files in addition to the animation and key board data input capabilities. SIMPLE_1 is not just a pretty picture: the language support extensive collection of statistics. Statistics collection capabilities of SIMPLE_1 include the ability to easily obtain statistics on user defined arrays.

The implementation of SIMPLE_1 combines the compilation and run time systems of the software into an integrated environment. The SIMPLE_1 environment includes on-line tutorials and full screen editor coupled to the compiler and run time system. Errors detected by the compiler or run time system initiate a call to the editor to isolate the error and speed up the edit-compile-debug cycle of modeling.

References

Cobbin, Philip, "SIMPLE_1: A simulation environment for the IBM PC", Modeling and Simulation on Microcomputers, Claude, C. Barnett, Editor, Society for Computer Simulation, La Jolla, 1986, pp 243-248.

Cobbin, Philip, "Applying SIMPLE_1 to manufacturing systems", Summer Computer Simulation Conference, July 28-30 1986, Reno, Nevada, Roy Crosbie and Paul Luker, Editors, Society for Computer Simulation, La Jolla, pp 724-730.

Cobbin, Philip, " Modeling tote stacker operation as a WIP storage device" To be published in: Winter Simulation Conference proceedings, December 1986, Washington D.C.

Sierra Simulations & Software: SIMPLE_1 User's guide and reference manual, 1985.

Starr, Patrick, Skrien,Douglas, and Meyer, Robert,"Simulating schedule recovery strategies in manufacturing assembly operations" To be published in: Winter Simulation Conference proceedings, December 1986, Washington D.C.

Philip Cobbin is the owner of Sierra Simulations & Software and is the developer of SIMPLE_1. Phil has developed and taught simulation to undergraduates as an adjunct professor of Industrial Engineering at San Jose State University. He holds a Master of Science in Industrial Engineering from Purdue University, a Bachelor of Science in Industrial Engineering and Operations Research from the University of Massachusetts at Amherst, and an Associate in Science degree in Manufacturing Engineering Technology from Waterbury Connecticut State College. Phil is a native of Los Angeles and has been previously employed by the General Products Division of the International Business Machines corporation performing simulation modeling and material handling engineering activities.