SHARE PROGRAM LIBRARY AGENCY

PROGRAM NUMBER

032019

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA
(305) - 284-6257

SHARE PROGRAM LIBRARY SUBMITTAL FORM

SHARE Program Library Agency
Triangle Universities Computation Center
P. O. Box 12175
Research Triangle Park, N. C. 27709

COSMIC CONTROL NUMBER :

This form should be completed and submitted with the program package to the SHARE Program Library Agency at the address shown above. Standards and instructions for submitting programs are in the "SHARE Program Library Standards Manual".

(1) Program Number (to be filled in by COSMIC). 360D-03.2.014

(2) System Type (machine) S/360, S/370

(3) Search Key . . . . . Compiler /and/executes/written/ in/the SIMSCRIPT II Programming Language

(4) Programming Language . . . . . 360ASALF ASM F

(5) Author's Name and Address . . . . . Philip J. Kiviat
CTEC, Inc.
7777 Leesburg Pike
Falls Church, VA 22043

(6) Direct Inquiries to Name and Address . . . . . (if different than Author)

(7) Title of Program . . . . . The SIMSCRIPT II Programming Language

(8) Submitter's Installation Membership Code. . . . . N

(9) Submitter's Own Program Identification and Suffix (optional). . . . . .

(10) Primary Subject Code. 03 2

(11) Operating or Monitor System Required OS/360

(12) New or Revision Code (if revision, show prior Program Number in Item 1). . . . . R

(13) Year Completed 69

(14) Date of Submittal. 06 07 72

(15) Documentation (number of original pages submitted) 47

(16) Abstract (should contain sufficient information for a reader to determine the value of the program). Listed on the reverse side of this form are subjects which may serve as a guide for a descriptive abstract.

SHARE PROGRAM LIBRARY SUBMITTAL FORM

Subject Guide:

a. Purpose
b. Programming Language used
c. Version and modification level or release number
d. Field of application
e. Type of routine (main program, subroutine, etc.)
f. Specific description of machine requirements

ABSTRACT

The SIMSCRIPT II compiler tranlates source language inputs into assembly programs which are assembled by an OS multiple-assembler into link-editable modules. The compiler itself is written in SIMSCRIPT II. It will run under MVT, MFT, or PCP.

The program should be stored in the user's load library and called out later by the compile procedures. Compilation requires core storage of at least 150 K bytes.

SIMSCRIPT II is described completely in THE SIMSCRIPT II PROGRAMMING LANGUAGE by P. J. Kiviat, R. Villanueva, and H. M Markowitz, Pretice-Hall, Englewood Cliffs, New Jersey, 1969.

(Please attach additional pages if necessary). . . . . Total pages attached ____

Permission to Publish

"I hereby give the SHARE Program Library Agency permission to reprint, reproduce, and distribute this program."

(17) Signature of Submitter and Date Donald W. Kosy 6-7-72

(18) Signature of Installation Addressee Donald W. Kosy

SIMSCRIPT II System Tape Summary

This 9 track, 800 bpi, labeled tape, 003381, contains 8 files. Each file is EBCDIC, has a logical record length of 80, and a block size of 800.

File 1: SYS1.SIM2LIB--SIMSCRIPT II object-time and run-time library.
logical record count = 979

File 2: SYS1.SIM2--SIMSCRIPT II compiler and multiple assembler interface.
logical record count = 2015

File 3: SYS1.SIM2GEN--JCL procedures to generate SIMSCRIPT II from this tape (003381) to disk.
logical record count = 19

File 4: SYS1.SIMPROC--JCL procedures to run SIMSCRIPT II.
logical record count = 78

File 5: SYS1.SIMSAMP--Sample SIMSCRIPT II program.
logical record count = 13

File 6: SIM2.COMPILER--Compiler source code.
logical record count = 6089

File 7: SIM2.LIBRARY--Library source code.
logical record count = 8885

File 8: SIM2.SUPPORT--Compiler support programs (grammar processor, etc.).
logical record count = 987

RM-5777-1-PR
MARCH 1972

The SIMSCRIPT II Programming Language: IBM 360 Implementation

P. J. Kiviat, D. W. Kosy, H. J. Shukiar,
J. B. Urman and R. Villanueva

This research is supported by the United States Air Force under Project Rand—Contract No. F44620-67-C-0045—Monitored by the Directorate of Operational Requirements and Development Plans, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of Rand or of the United States Air Force.

Rand
SANTA MONICA, CA. 90406

PREFACE and SUMMARY

This Memorandum contains information on the implementation of the SIMSCRIPT II computer language on Rand's IBM 360 Model 65 computer system. It should be of general interest to IBM 360 users, and to persons undertaking the task of implementing SIMSCRIPT II on other computers. Both the implementation and this report have been revised to eliminate errors discovered since the report was first released (July 1969) and to bring out a few new features.

Principal changes include:

1. List statements, automatic entity checking, lefthanded functions, monitored variables, and BEFORE and AFTER statements now supported.
2. BETA.F, ERLANG.F, GAMMA.F and WEIBULL.F statistical functions corrected and a new gamma variate generator, GAMMAJ.F, added for use with small shape parameters.
3. The value of SEED.V(1) changed to provide a better initial sequence of random numbers for stream 1. Note that results of runs made using this new release will therefore differ from runs using the previous release. (To use the original stream-1 random-number sequence, LET SEED.V(1) = 524287 before beginning the simulation.)
4. JCL procedures slightly modified to eliminate need for the SIMIN dataset and to produce less output from the assembly step.
5. A normal approximation for values of mean MU greater than 6 for the POISSON.F random variate generator. This is faster for large values of MU and eliminates an error condition that could occur in the old function. Note that results of runs made using this new release differ from those runs using the old POISSON.F function.
6. The addition of routine SNAP.R, to allow the user to temporarily regain control from the monitor upon detection of a fatal error at run-time.

Because of these modifications, the user must do the following to run programs previously compiled by earlier versions of the compiler: either (1) insert a control card to the linkage editor, preceding the object decks of the form bENTRY MAIN or (2) recompile the MAIN routine to obtain a new object deck.

SIMSCRIPT II is described completely in P. J. Kiviat, R. Villaneuva, and H. M. Markowitz, *The SIMSCRIPT II Programming Language*, The Rand Corporation, R-460-PR, October 1968 (also published as a commercial book by Prentice-Hall, Englewood Cliffs, New Jersey, 1969). A companion-piece to the present Memorandum offers a compact reference listing of the syntax and semantics of SIMSCRIPT II, designed for professional programmers already familiar with the language. See P. J. Kiviat and R. Villaneuva, *The SIMSCRIPT II Programming Language: Reference Manual*, The Rand Corporation, RM-5776-PR, October 1968. The system itself is available through the SHARE/COSMIC library.

CONTENTS

TABLES

I. IMPLEMENTATION CONSIDERATIONS AND RESTRICTIONS

This section details the IBM System/360 implementation of the language defined in Rand Report R-460-PR (see Preface). Suggestions are made for improving program efficiency in cases where such comments are appropriate. All references are to section numbers in the defining report. Features of the language not yet implemented are also listed.

Section 1-02

Because *SIMSCRIPT II* ignores trailing periods (see *Section 1-09*), numbers followed by a period are treated as integers (e.g., 55. is interpreted as the integer 55). A zero must be included for the constant to be considered a real (decimal) number. Notice that this convention is quite different than that of FORTRAN and other languages. Whether a constant is integer or real is sometimes important, e.g., when used as an argument to a function (see *Section 2-19*).

Section 1-08

The statements

```
ADD e TO v          and
SUBTRACT e FROM v
```

are translated into

```
LET v = v + (e)     and
LET v = v - (e)
```

before they are compiled, giving rise to three potential problems:

(1) If v is a subscripted variable with complex subscript references, it is more efficient to compute the subscripts in a separate statement than to have the compiler compute them twice. For example,

```
ADD 1 to X(Y*(AB-2),DIFF**N)
```

translates to

```
LET X(Y*(AB-2),DIFF**N) = X(Y*(AB-2),DIFF**N) + 1
```

causing the subscripts Y*(AB-2) and DIFF**N to be evaluated twice.

To conserve storage space and computer time, the statement should be written as:

```
LET I = Y*(AB-2)
LET J = DIFF**N
ADD 1 TO X(I,J)
```

(2) Unforeseen difficulties can arise if a subscript is or contains a function, particularly one that has side effects, e.g., calls the random number generator. For example,

```
ADD 1 TO TABLE(UNIFORM.F(A,B,1))
```

is translated to

```
LET TABLE(UNIFORM.F(A,B,1)) = TABLE(UNIFORM.F(A,B,1)) + 1
```

before it is compiled, causing two random numbers to be generated, and most probably two different elements of TABLE to be accessed. The statement is more properly written as

```
LET I = UNIFORM.F(A,B,1)
ADD 1 TO TABLE (I)
```

(3) Duplicate error messages may be produced because of intermediate translations as in (1) and (2) above.

Section 1-09

The special characters ¢ and ! do not print.

Section 1-13

In the SKIP e OUTPUT LINES statement:

if e < 0, it is set to 0
if e > LINES.V, it is set to LINES.V

Section 2-03

The code generated for the statement RESERVE v(*) AS e, and similar statements, uses v(*) twice. The statement generates a subprogram call in which v(*) is used once as a GIVING and once as a YIELDING argument. For example, the statement

RESERVE ARRAY(I,*) AS 15

generates the statement

CALL RES.R(ARRAY(I,*),2,15) YIELDING ARRAY(I,*)

for the unpacked, two-dimensional array ARRAY.

If I is a function with side effects, the result can be incorrect. The same steps should be taken in each of these cases as in *1-08*.

In statements of the form

RESERVE *array pointer list* AS e

the expression e is evaluated once for each array named in the list. Arrays of more than one dimension also have the expressions in the BY e clauses reevaluated.

Section 2-05

The upper bound expression (e_2) of a FOR loop is evaluated each time the loop is repeated. This also applies to the BY expression (e_3), if any.

Section 2-09

Improper index evaluations in the statements GO TO label(e) and GO TO $label_1$ OR $label_2$ OR ... OR $label_n$ PER e do not always have the same consequences. Assume that some or all of the labels, label(i), i = 1, ..., n have been defined.

e = 0	Program terminates with a meaningful error message.
e < n label(e) undefined	Program terminates with a meaningful error message
e > n and transfer is to a valid nonbranching instruction	Program terminates a meaningful error message.
e > n and transfer is to a data word	Program terminates with an operating system error message.
e > n and transfer is to a valid branching instruction	Program continues, most likely in error, with no warning given.

Section 2-16

The maximum size of a subprogram (main routine or routine) is 12,288 bytes.

12,288 = 3 base registers @ 4,096 bytes/register

The first seven letters of each routine name must be unique, e.g., a program cannot contain routines named PROCESSA and PROCESSB. External references can only contain eight letters; SIMSCRIPT II prefixes the letter L (left) or R (right) to the first seven letters of each routine name in constructing the external references. In addition, all periods are converted to dollar signs. All references to routines in object decks, assembly listings, or memory maps are of this form, e.g., the right-handed routine JOB.OVER.NOW is converted to RJOB$OVE.

Section 2-26

The form line of a PRINT statement cannot appear in the i lines following a SUBSTITUTE THESE i LINES FOR *word* statement.

Section 3-04

The RELEASABLE declaration is not necessary when operating under PCP or MFT II. The statement is not available under MVT.

Section 3-05

Overlay is available through normal OS/360 procedures. None of the features in the section--LOAD, SAVE, DYNAMIC, OVERLAY, or IS LOADED--is implemented.

Section 3-07

If a statement is of the form COMPUTE v AS THE *statistic* OF exp.

and

statistic is MAX or MIN and
v is INTEGER and
exp. is INTEGER

or is of the form COMPUTE v AS THE *statistic* OF exp. and

statistic is MAX or MIN and
v is INTEGER and
exp. is INTEGER

significance can be lost due to double conversion between INTEGER and REAL during evaluation. This will only happen if some value of exp. or e is greater than 2^{24}.

Section 3-08

An ALPHA variable or constant can hold at most four characters. ALPHA literals are left-adjusted if less than four characters.

Section 3-09

In a WRITE statement, an ALPHA literal (character string) is limited in length only by what is expressible on a single card.

In the statement USE d FOR INPUT and USE d FOR OUTPUT, d is an arithmetic expression. If it is REAL, it is rounded to INTEGER during execution.

The "standard" d (device) numbers are:

input care reader = 5

output line printer = 3
punch = 2

JCL DD cards are needed to define additional I/O devices (see Sec. V).

During output, on those devices for which it is relevant, column 0 is used for carriage control. Ordinarily, SIMSCRIPT II takes care of setting its value. It can be addressed as OUT.F(0) and thus set or tested by the programmer. The carriage control characters and their functions are:

Printer		Punch	
blank	single space	n	select stacker n
0	double space		
-	triple space		
1	new page		
2-12	skip to channel i		
+	no space		

Since OUT.F is an ALPHA function, the control characters must be expressed as ALPHA literals, e.g., LET OUT.F(0) = "1".

SECTION 3-10

The ADVANCE and BACKSPACE statements are not implemented.

Section 3-10-2

Whenever a REWIND statement is executed and the unit designated is the current input (output) unit, the current input (output) unit is changed to the card reader (printer).

Examples:

	Statements	System Variables
(1)	USE 8 FOR INPUT READ X AS D(10,2)	READ.V = 8
	⋮	
	REWIND 8	READ.V = 5
(2)	USE 8 FOR OUTPUT WRITE X AS D(10,2)	WRITE.V = 8
	⋮	
	REWIND 8	WRITE.V = 3

The consequences of these actions are that every REWIND statement must be followed by a USE statement or USING phrase before the rewound unit can be read from or written on.

To read a tape or disk repeatedly one must write:

```
'READ' REWIND d
       USE d FOR INPUT
       READ ...
            ⋮
       GO READ
```

and not

```
       USE d FOR OUTPUT
'READ' REWIND d
       READ ...
            ⋮
       GO READ
```

REWIND destroys the integrity of the "current unit."

Section 3-11

The system variables RCOLUMN.V and WCOLUMN.V are set to -1 by the system after the unit to which they refer has been USED. If a programmer wants to use them before the first read or write has been executed, he must take this into consideration.

Section 3-12

Automatic page turning and accounting for the variables LINES.V, LINE.V and PAGE.V are done for the standard line printer, unit number 3, only. For other devices, OUT.F(0) must be used.

Section 4-01

The first five letters of each entity name must form a unique word. Permanent entity names are prefixed with the characters RC$ to form external names for their "create routines"; temporary entity names are prefixed with the characters GW$ to form external names for the variables containing the length of their entity records.

Section 4-02

The first five letters of each set name must form a unique word. Three characters are prefixed to these letters to form eight-letter external references for the seven possible set routines. These three letters are an R plus the characters listed in Table 14 of R-460-PR.

Section 4-07

Every global variable or attribute not assigned to a word or array must have its first seven letters form a unique word. Such variables are accessed by external references, and must therefore follow their restrictions (see *Sec. 2-16*). When possible, word or array specification should be used, as it generates more efficient code.

ALPHA variables can only be quarter field and intra-packed.

The default condition for bit and quarter field packing of integers is unsigned. Signed can be specified when necessary.

REAL variables cannot be packed.

Section 4-12

Implied subscripts cannot be used when reading attributes of permanent entities. If AGE is an attribute of the permanent entity MAN, the form FOR EACH MAN, READ AGE(MAN) must be used. READ AGE is assumed to be a free form read of array AGE.

Section 4-17

Arguments of a subscripted monitored variable are automatically converted to INTEGER (as for any subscripted variable) before being passed to the monitoring routine. Note that the monitoring routine cannot be CALLed directly. If *variable* is monitored, the statement CALL *variable* is undefined.

Section 5-01

The first five letters of every event name must form a unique word. This is because event names represent both temporary entities (event notices) and routines and must follow the conventions of each (see *Sec. 2-16* and *Sec. 4-01*).

External events cannot be assigned to unit zero in an EXTERNAL EVENT UNITS statement.

Section 5-02

Statements of the form

SCHEDULE AN *event* CALLED $expression_1$ IN $expression_2$ DAYS

are compiled into the statements

```
CREATE AN event CALLED expression1
LET TIME.A(INT.F(expression1)) = TIME.V + (expression2)
CALL A.EV.S(INT.F(expression1), I.event)
```

Similar code is generated for SCHEDULE statements that use NOW or AT clauses. Since $expression_1$ is evaluated three times, steps similar to those discussed in *Sec. 1-08* should be taken to guard against unwanted side effects and to improve efficiency.

The SCHEDULE...NOW statement is implemented in a way that could cause difficulties for the rare case where a programmer chooses to manipulate event notices in the timing sets (EV.S) directly. Users who deal with events only through the normal SIMSCRIPT II statements and mechanisms (SCHEDULE, CANCEL, START SIMULATION) will have no difficulties with the statement.

A statement of the form

SCHEDULE THIS *event* NOW

is compiled into the statements

```
LET TIME.A(INT.F(event)) = -RINF.C
CALL A.EV.S(INT.F(event), I.event)
```

Similar code is generated for other variations of the SCHEDULE statement (see above).

The large negative value of TIME.A insures that the event so scheduled will be the first of its class to be removed from its timing set, EV.S(I.*event*). If several events of different classes are scheduled NOW within the same event routine, each appears at the top of its respective timing set and the appropriate one is selected first according to the implicitly or explicitly specified priority.

It is therefore possible for events scheduled to occur at positive simulation times to have negative event times stored in their event notices. Programmers should bear this in mind if they search or operate on timing sets themselves.

When an event is selected to occur, i.e., to become the "next" event, its TIME.A attribute is tested to see if it is -RINF.C. If it is, it is set to the current value of TIME.V before control is passed to the event routine. If it is not, TIME.V is set to its value. In this way, when an event occurs, the TIME.A attribute of its event notice always contains the correct current simulation time.

STATEMENTS NOT YET IMPLEMENTED

The following statements and language features are not implemented in the current version of the compiler.

Level 3

CLOSE, ADVANCE, BACKSPACE
Column repetition for the PRINT statement
LOAD, SAVE and associated features

Level 4

LIST attributes
All TEXT features

Level 5

BREAK TIES
Event arguments in SCHEDULE and EVENT
ACCUMULATE, TALLY, DUMMY, and RESET
RANDOM variables
ORIGIN.R routine and the conversion functions that depend on it

HELPFUL HINTS

The following are a number of frequently made programmer errors.

Compilation

(1) Routines not having their first seven characters form a unique word introduce *undetectable* duplicate names. Extreme care should be used in naming routines.

(2) Permanent entity, temporary entity, event notice, or global variable names not having their first seven characters form a unique word introduce duplicate names. These errors are flagged during the assembly step of the preamble and are corrected by changing the names.

(3) Misspelling the name of a variable or function results in an implicitly defined local variable. Execution will proceed with this "new" variable initialized to zero. Such an error will be detected only if the variable is subscripted and an attempt is made to change its value.

Execution

(1) Disagreements in mode between arguments in calling statements and corresponding definitions in routines may be difficult to discover as effects are subtle, e.g., an INTEGER number used as a REAL and taken to be a zero.

(2) Arrays not reserved before they are used generally result in error message 204 or 205 (see Table 3).

(3) Subscript limits exceeded often result in error message 204 or 205, but may give no indication of error.

(4) Incorrect entity identification number of index used to access an attribute has the same effect as exceeding a subscript limit and also often results in error message 206.

(5) Incorrect JCL cards produce system completion code such as 013 or F13.

(6) Running out of core storage space produces system completion code 80A.

II. RULES AND DECK SETUP FOR COMPILATION, ASSEMBLY, AND EXECUTION

A SIMSCRIPT II program is compiled by putting its preamble before all its source language subprograms, prefacing and following this card deck with appropriate control cards, and submitting the complete deck to the SIMSCRIPT II compiler (see Table 1).

CONTROL CARDS

Table 1

JOB CONTROL CARDS USING CATALOGUED PROCEDURES

COMPILE-ASSEMBLE-LINK EDIT-EXECUTE

```
//jobname     JOB     appropriate parameters
//  EXEC   SIM2
//SIM.SYSIN   DD   *

     SOURCE DECKS

/*
//LKED.SYSIN   DD   *
                                    }
     OBJECT DECKS                   }      Optional
                                    }
/*
//GO.SYSIN   DD   *

     DATA DECKS: MAY BE NONE

/*
```

COMPILE-ASSEMBLE

```
//jobname    JOB    appropriate parameters
//  EXEC   SIM2CA
//SIM.SYSIN  DD   *

     SOURCE DECKS

/*
```

LINK EDIT-EXECUTE

```
//jobname    JOB    appropriate parameters
//  EXEC   SIM2LG
//LKED.SYSIN  DD  *

     OBJECT DECKS

/*
//GO.SYSIN   DD   *

     DATA DECKS: MAY BE NONE

/*
```

Within JCL cards, blanks are meaningful. Punch all cards exactly as shown. The //GO.SYSIN DD * card and accompanying /* card are always needed, even if there is no input data.

NOT USING THE CATALOGUED PROCEDURES

Section IX shows how a SIMSCRIPT II program deck is embedded in the JCL that integrates the various SIMSCRIPT processing steps. As with the catalogued procedures, the program, object, and data decks follow the SIM.SYSIN, LKED.SYSIN, and GO.SYSIN DD * cards respectively.

DEFAULT CONDITIONS

The compile and execute phases have default region sizes of 150K and 52K, respectively. When compilations require more space, the 150K default can be overridden by putting a REGION.SIM=nK phrase on a compile step EXEC card as follows:

```
// EXEC  SIM2,REGION.SIM=180K
// EXEC  SIM2,REGION.SIM=250K
// EXEC  SIM2CA,REGION.SIM=168K
```

Larger execution phase regions are specified by a REGION.GO=nK phrase on an appropriate EXEC card:

```
// EXEC  SIM2,REGION.SIM=170K,REGION.GO=100K
// EXEC  SIM2,REGION.GO=150K
// EXEC  SIM2LG,REGION.GO=100K
```

When REGION.SIM and REGION.GO phrases appear on the same EXEC card, the REGION.SIM phrase must be listed first.

SIMSCRIPT II always produces an assembly listing and an object deck unless otherwise specified. To inhibit these, one can either pass the appropriate parameters to the assembler or DUMMY the assembler output units. This card

```
//ASM.SYSPUNCH  DD  DUMMY
```

inserted right after the program deck, prohibits punching the object deck while

```
//ASM.SYSPRINT  DD  DUMMY
```

cancels the assembly listing. To delete both, the SYSPUNCH card is placed before the SYSPRINT card; both are inserted immediately following the program deck.

RECOMPILATION

When the preamble is compiled, tables are constructed that define the program's variables and structure. These tables are used during compilation of the subprograms. The preamble also generates a routine

named PRMB[†] and routines that support the entity-attribute-set and event declarations.

Individual subprograms can be compiled separately, for either correction or expansion, by preceding them with the preamble. To inhibit the generation of a duplicate set of support routines, the word OLD should be put before PREAMBLE. PRMB is always generated.[†]

While certain preamble modifications require the recompilation of entire programs, others do not. New entities, attributes, and sets can be added to programs without the need for complete recompilation; if specific locations (words or arrays) are not specified, external references are used to maintain compatibility. Certain program changes require recompilation.

Subprograms must be recompiled if variables within them:

- have their packing factors changed;
- are equivalenced differently;
- have their array or word specification changed;
- are affected by addition or deletion of a TALLY or ACCUMULATE statement;
- have their monitor status changed;
- are affected by addition or deletion of a BEFORE or AFTER statement.

Subprograms must be recompiled if sets within them:

- are affected by addition or deletion of a BEFORE or AFTER statement;
- have their ranking order changed;
- have the properties of their set attributes changed.

Subprograms must be recompiled if events within them:

- are affected by addition or deletion of a BEFORE or AFTER statement.

[†]If a preamble defines more than 80 external references, a routine PRMB1 is generated. If it defines more than 160 external references, a routine named PRMB2 is generated. As many PRMB1 routines are generated as are necessary.

III. COMPILATION AND ASSEMBLY

Processing of a SIMSCRIPT II program occurs in two phases. In phase one, the source program is read, a listing of it annotated with error messages (if there are any) is printed, and an assembler language program is produced. In phase two, the program is assembled.

The compilation phase listing is almost self-explanatory, as it looks exactly like the programmer's source deck (except that DOUBLE lines are listed as one). When there are language errors, a line is printed identifying the source statement containing the error and the word in question as well as a code. The codes are described in Table 2. Most often, errors can be detected at the source language level, i.e., the assembly listing is not required.

At times, the assembler output must be consulted. It looks as follows:

> Each SIMSCRIPT II source statement appears as a remarks card, i.e., there is an asterisk following the statement number.
>
> When a source statement is "scripted," i.e., decomposed by the compiler into simpler SIMSCRIPT II source statements, these statements follow the remarks card and are preceded by two or more asterisks.
>
> The generated assembly code follows the remarks cards. Two examples illustrate each case:

(1) A source statement not requiring scripting.

Compiler listing:

```
LET EVENT(5,1) = 0
```

Assembly listing:

```
145*      LET EVENT(5,1) = 0
146  L    11,=V(GEVENT)
147  L    11,0(11)
148  L    11,20(11)
149  LE   0,=E'0'
150  STE  0,4(11)
```

(2) A source statement requiring scripting.

```
Compiler listing:
          READ N.ELEVATOR
Assembly listing:
151*        READ N.ELEVATOR
152**       CALL RFI.R YIELDING N.ELEVATOR
153    L    15,V(RRFI$R)
154    BALR 14,15
155    L    11,LV+52(6)
156    L    13,=V(GN$ELEVA)
157    ST   11,0(13)
```

If DEFINE TO MEAN or SUBSTITUTE statements have been used, the substitutions they invoke are seen in the remarks cards, though not in the phase one program listings.

At times, compiler-produced error messages will refer to words that are not contained in source statements. This will happen when a statement is scripted in such a way that words are generated and these words are in error. For example, in compiling the statement DESTROY DOG CALLED D, where DOG is an entity that belongs to a set named KENNEL, the word M.KENNEL is scripted when code is compiled to check if an entity that belongs to a set is being destroyed. If for some reason M.KENNEL triggers an error, the error can only be found by checking the message against the assembly listing.

Table 2

MESSAGES PRODUCED DURING COMPILATION

Number	Explanation
1	Word does not conform to statement syntax, or statement is not allowed in this part of the program.
2	Missing) in arithmetic expression or subscript inserted.
3	Missing terminal " or | in literal; literal deleted.
4	More formats than variables or expressions; WRITE (ignored), PRINT (zeroes inserted).
5	More variables or expressions than formats; excess ignored.
6	Conflicting or redundant properties in DEFINE statement; statement ignored.
7	Number of subscripts inconsistent with definition or first use; definition or first use employed.

Table 2--continued

Number	Explanation
8	ELSE without a matching IF ignored.
9	IF without a matching ELSE; error branch constructed.
10	Previous definition or use of this name precludes its use in this context.
11	Attempt to assign a value to this in-line function; value assigned to dummy location.
12	Array number of attribute greater than 724.
13	This should be the name of a routine; statement ignored.
14	This form of RETURN statement not allowed in event or left-handed routine; expression ignored.
15	LOOP without a matching DO ignored.
16	Common attribute without a subscript. Implied subscripting not allowed; unaccessible local variable created.
17	Number of GIVING arguments inconsistent with definition; definition employed.
18	Multiple use of label name ignored.
19	This label should be subscripted; subscript of 1 assumed.
20	Name repeated in parameter list of ROUTINE or EVENT statement.
21	Undefined label; error branch constructed.
22	DO without a matching LOOP; error branch constructed.
23	RETURN used in MAIN routine; STOP substituted.
24	Missing END statement supplied.
25	Missing text in DEFINE TO MEAN or SUBSTITUTE; statement ignored.
26	Inappropriate mode and/or dimensionality for this implied subscript due to local definition.
27	Attempt to place an attribute in the first five words of an event notice; specification ignored.
28	Unsubscripted SUBPROGRAM variable expected here; in CALL statement ignored, in indirect function value of zero used.
29	Too many internal labels generated in this routine; reduce size of routine or number of labels.
30	Temporary entity more than 1023 words long; delete attributes or specify packing.
31	Subscripts not allowed with this variable; subscripts ignored.
32	The subscript of this temporary attribute should be INTEGER; conversion made. Attribute may be implicit.

Table 2--continued

Number	Explanation
33	Negative number (with this magnitude) used as a subscript or entity reference; absolute value used.
34	Unsubscripted label expected; subscript ignored.
35	THEN IF without a preceding IF; word THEN ignored.
36	Missing) in a logical expression inserted.
37	DIV.F used with non-INTEGER argument; ordinary division assumed.
38	Number of YIELDING arguments inconsistent with definition; definition employed.
39	Non-function used as an attribute of a "mixed" compound entity; function assumed.
40	Attempt to equivalence function attributes.
41	Missing) in equivalence group inserted.
42	Attempt to pack a function ignored.
43	Attempt to pack an unsubscripted system attribute ignored.
44	Illegal packing requested for this name; packing ignored.
45	Packing of form (*/n) used with temporary attribute; (1/n) assumed.
46	BELONGS clause with compound entity ignored.
47	Attempt to define non-local variables as having local property, e.g., SAVED,RECURSIVE, ignored.
48	Incorrect mode specification for packed variable; mode set to INTEGER, packing kept.
49	Attempt to define a set not previously mentioned in an EVERY statement ignored.
50	This statement should be preceded by a FOR, WHILE or UNTIL phrase.
51	Output format used in READ; format ignored.
52	Illegal or out-of-place * used as a subscript or argument with this name; for subscript all asterisks assumed to follow, for argument function assumed zero.
53	This has not been defined as a set; statement ignored.
54	Set lacks attribute(s) needed for this statement; statement ignored.
55	This should be the name of a permanent entity; statement ignored.

Table 2--continued

Number	Explanation
56	Assume a DO before this statement (after ALSO group).
57	This should be a temporary entity; statement ignored.
58	GROUP used without column repetition.
59	This should be the name of an event; statement ignored.
60	Suppression in midst of column repetition group (suppressed).
61	FOR phrase expected after PRINTING; PRINTING ignored.
62	IN GROUPS OF phrase expected; column repetition ignored.
63	IN GROUPS OF 0 used; assume 1.
64	Unfinished heading; END statement added.
65	Unfinished report; END statement added.
66	PRINT 0 LINES specified; statement ignored.
67	Not enough print lines provided; excess ignored.
68	Owner or member missing for this set.
69	This automatically defined attribute of a common set should have been mentioned in an EVERY statement.
70	Mode of this variable or function conflicts with its automatic definition.
71	Number of subscripts or GIVING arguments of this name conflicts with its automatic definition.
72	Explicit definition of this name conflicts with its automatic definition.
73	This ranking attribute should have been mentioned in an EVERY or THE SYSTEM statement.
74	This type of FILE statement illegal with a ranked set; statement ignored.
75	Number of GIVING arguments greater than 254; reduced to 254.
76	Number of YIELDING arguments greater than 254; reduced to 254.
77	Number of subscripts greater than 254; reduced to 254.
78	Subscript of label should be between 1 and 3000; this label ignored.
79	Too many recursive local variables in this routine; reduce size of routine.
80	Subscripted variable expected here; ignore this variable.
81	YIELDING arguments not allowed with left-handed or monitoring routines; YIELDING list ignored.

Table 2--continued

Number	Explanation
82	ENTER statement allowed only in left-handed routine; ignored.
83	Monitoring not allowed in local DEFINE statement; monitoring ignored.
84	Monitoring routine has incorrect number of GIVING arguments.
85	This MOVE statement not allowed in this routine; ignored.
86	BEFORE CREATING or AFTER DESTROYING not allowed; ignored.

IV. EXECUTION

The TRACE statement generates a backtrack of all subprograms currently called, including those called by the system. Its output looks like:

```
AT LOCATION     00002E40
CALLED FROM     0000D314
CALLED FROM     000000B2
```

To determine the source language statements that correspond to these locations, which are relative to the program load point,

(a) Using the link edit memory map, locate the routines containing the locations;

(b) Subtract (in hexadecimal) the first location of each routine from its corresponding TRACE location to get the relative address of the location within the routine;

(c) Using the assembly listings, find the source statements that correspond to the relative addresses within the routines.

When the SIMSCRIPT II system detects a program error during execution it calls TRACE, prints an appropriate message, calls SNAP.R, and terminates the program. The messages produced by the system during execution are listed in Table 3.

TRACE also prints the current values of the following system variables:

TIME.V	the current simulation time
READ.V	the number of the current input unit
WRITE.V	the number of the current output unit
RCOLUMN.V	the rightmost column of the last data field read
WCOLUMN.V	the rightmost column of the last data field written
EOF.V	the current value of the end-of-file control variable
EVENT.V	the class number of the next event to be executed

The library contains a default SNAP.R routine of the form

```
ROUTINE  SNAP.R  RETURN  END.
```

It can be replaced at any time by a user-written SNAP.R routine. This can be used, for example, to display values of global variables, arrays, attributes, etc., when an execution error is discovered. This routine should not be used to restart the simulation since recovery from error conditions is not possible from SNAP.R.

Table 3

MESSAGES PRODUCED DURING EXECUTION

Number	Explanation
1	Zero to a negative power.
2	Zero used as second argument of MOD.F.
3	Negative number raised to a real power.
4	Invalid I/O unit.
5	Negative expression in SKIP INPUT statement.
6	Attempt to file an entity in a set it is already in.
7	Attempt to file BEFORE or AFTER an entity that is not in the mentioned set.
8	Input requested from output device.
9	Attempt to REMOVE from an empty set.
10	Attempt to REMOVE an entity that is not in set.
11	Invalid stream number for random draw.
12	Output requested on an input device.
13	Attempt to cause an event already scheduled.
14	Attempt to cancel an event not scheduled.
16	No memory space available; increase region size if possible.
17	Negative argument in ITOA.F.
18	Argument > 9999 in ITOA.F.
20	Input from an unopened device.
21	Attempt to use a unit for input that is in the output state.
22	Attempt to use a unit for output that is in the input state.
24	Output to an unopened device.
28	Formatted READ goes beyond the end of input record.
32	Negative field width in input format.
36	Negative field width in output format.
40	Mixed binary/EBCDIC record.
41	Invalid character while reading "C" format.
42	Too many characters to read and store in "C" format.
44	Output format field width greater than record size.
45	Attempt to WRITE or PRINT an ALPHA string longer than output record.

Table 3--continued

Number	Explanation
46	Attempt to transfer to missing LOOP statement--see compiler message.
48	Input format field width greater than record size.
52	"S" format width exceeds buffer size.
56	Negative field width in "S" format.
60	Zero or negative subscript specification in RESERVE statement.
64	Reference to an unreserved array in DIM.F.
68	End of file encountered during read operation while EOF.V = 0.
72	"B" format width ≦ 0 on input.
76	"B" format width ≦ 0 on output.
80	Real number too large to be converted to integer.
84	Invalid character in "I" format during input.
88	Integer number too large for input.
92	START NEW PAGE statement inconsistent with output unit definition.
96	Unsuccessful open--missing DD card likely.
100	Attempt to transfer to undefined unsubscripted label--see compiler message.
104	Wild transfer in computed or subscripted GO statement.
108	Transfer to missing ELSE statement--see compiler message.
112	Parameter 2 negative in "D" or "E" format.
116	Parameter 2 > Parameter 1 in "D" or "E" output format.
122	Parameter 2 > Parameter 1 in "D" or "E" input format.
124	Real number too large for input.
128	Invalid character in "D" or "E" format during input.
131	OUT.F(0) cannot be used with non-carriage control unit.
132	Mean in EXPONENTIAL.F call ≤ 0.
133	Mean in ERLANG.F call ≤ 0.
134	Number of stages in ERLANG.F ≤ 0.
135	Mean in LOG.NORMAL.F call ≤ 0.
136	Standard deviation in LOG.NORMAL.F call ≤ 0.
137	Standard deviation in NORMAL.F call ≤ 0.

Table 3--continued

Number	Explanation
138	Mean in POISSON.F call ≤ 0.
139	Second parameter less than first in RANDI.F call.
140	Second parameter less than first in UNIFORM.F call.
141	Number of trials in BINOMIAL.F call ≤ 0.
142	Probability in BINOMIAL.F call ≤ 0.
143	Shape parameter ≤ 0 in WEIBULL.F call.
144	Scale parameter ≤ 0 in WEIBULL.F call.
145	Mean in GAMMAJ.F ≤ 0.
146	Shape parameter in GAMMAJ.F ≤ 0.
147	First parameter in BETA.F call ≤ 0.
148	Second parameter in BETA.F call ≤ 0.
150	Absolute value of SIN.F,COS.F argument $\geq \pi.2^{18}$.
151	Value of EXP.F argument > 174.673.
152	Value of LOG.E.F,LOG.10.F argument ≤ 0.
153	Value of ARCSIN.F,ARCCOS.F argument > 1.
154	Value of ARCTAN.F arguments = (0,0).
155	Value of SQRT.F argument < 0.
156	Absolute value of TAN.F arguments $>\pi.2^{18}$.
157	Value of TAN.F argument too close to a singularity.
160	Negative time expression in call of WEEKDAY.F.
161	Negative time expression in call of HOUR.F.
162	Negative time expression in call of MINUTE.F.
201	Attempt to execute unassigned or unavailable instruction.
202	Attempt to execute a privileged instruction.
203	Attempt to have an EXECUTE instruction execute another EXECUTE instruction.
204	Attempt to access protected memory area.
205	Invalid address specified.
206	Many errors possible. See message for completion code 0C6 in IBM manual.
207	Data stored incorrectly for operation.
208	Fixed-point-overflow encountered.

Table 3--continued

Number	Explanation
209	Fixed-point-divide yields excessive quotient; may be division by zero.
210	Decimal-overflow encountered.
211	Decimal-divide yields excessive quotient.
212	Floating-point exponent overflow encountered.
213	Floating-point exponent underflow encountered.
214	Floating-point addition or subtraction yields all-zero fraction.
215	Floating-point division by zero attempted.
216	Attempt to release a routine.
217	Negative argument to OUT.F.
218	Argument to OUT.F exceeds buffer length.
219	No more storage available for recursion.†
220	Simulation time decrease attempted.
221	Name in name field of external event data card (record) is not an external event name.
222	Non-ALPHA word encountered in name field of external event data card (record).
223	ALPHA field encountered in time field of external event data card (record).
224	Calendar-time format not yet implemented.
225	LIST statement attempted while a previous LIST is in progress.
226	Attempt to destroy an entity that is in a set.

†Additional local variable storage can be obtained by writing a small assembly program of the following form and linkediting it to override the normal system routine.

```
XLV   CSECT
      DC   A(LAST)
      DS   1536F
LAST  EQU  *-1
      END
```

A number greater than 1536 in the "DS" card allocates additional full words to the local variable storage area.

V. DEFINING ADDITIONAL DATA SETS

GENERAL RULES

To make use of additional data sets (tape or disk) during execution, additional data definition (DD) cards must be included with the deck. These follow standard OS Job Control Language and are subject to the following rules:

1. The card must begin in column one as follows:

   //GO.SIMUXX

   where XX is the unit number referenced in the program. For example,

   //GO.SIMU08

   will refer to unit number 8.

2. Unit numbers must be from 01 to 15 and should exclude 02, 03, and 05; these are reserved for the punch, printer, and card reader, respectively.†
3. For all data sets, DCB parameters must be included; there are no defaults. The three essential parameters are RECFM, BLKSIZE, and LRECL.

RECORD FORMATS

SIMSCRIPT II supports either fixed-length or variable-length records.† When using fixed-length records, each time a record is written or read, the same number of bytes (characters) are transferred, regardless of how full the record is. With variable records, only the amount of data in the buffer is actually transferred to the input/output unit. With fixed-length records, the unused portion of the buffer is padded with blanks. Thus, where a great variety in record

---

†Units 1-5 are reserved for fixed length records only.

lengths is required, variable-length formats should be used; if all records are the same or about the same length, or if binary records are used, fixed-length format should be used. Variable-format records require somewhat more time during execution but can save storage space.

To specify that a unit is to be fixed-length, use the following rules concerning the DCB parameter:

1. RECFM=FB
2. LRECL = the number of bytes in the largest record to be written.
3. BLKSIZE = a number of bytes equal to some multiple of the LRECL parameter; the larger the multiple, the more core used up, but the faster I/O operations will be handled.

NOTE: When writing as BINARY, fixed block records should be used, and the BLKSIZE and LRECL parameters should be some multiple of four, e.g., 100.

To use variable-length record format, follow the following rules:

1. RECFM=VB.
2. LRECL = the number of bytes in the largest record plus four.
3. BLKSIZE = a number of bytes equal to some multiple of the LRECL parameter plus four.

Some examples:

1. Scratch disk data sets

```
//GO.SIMU09  DD  UNIT=DISK,SPACE=(400,(1000)),
//  DCB=(RECFM=FB,BLKSIZE=400,LRECL=400)
```

This data set has fixed-length 400 byte records; the blocking factor is one, and enough space is provided to hold 1000 blocks; the data set could be used for a binary file.

```
//GO.SIMU09  DD  UNIT=DISK,SPACE=(400,(1000)),
//  DCB=(RECFM=FB,BLKSIZE=4000,LRECL=400)
```

This is similar to the first example, except each block will contain ten records; enough space is provided for 1,000 blocks or 10,000 records.

```
//GO.SIMU12  DD  UNIT=DISK,SPACE=(408,(100)),
//  DCB=(RECFM=VB,BLKSIZE=412,LRECL=204)
```

Here, variable blocked records are used; note that the largest record possible is 200 bytes since LRECL is four larger than the data portion of a record; there are two records per block ((2*LRECL) + 4); there is enough space to hold 100 blocks or 200 records.

2. Permanent disk data sets:

```
//GO.SIMU06  DD  UNIT=DISK,SPACE=(400,(1000)),
//  DCB=(RECFM=FB,BLKSIZE=400,LRECL=400),DISP=(NEW,CATLG),
//  DSNAME=MYDATA
```

This example is identical to the first one except the data set will be cataloged (the DISP parameter) and its name will be MYDATA (the DSNAME parameter).

```
//GO.SIMU07  DD  UNIT=DISK,SPACE=(244,(1000)),
//  DCB=(RECFM=VB,BLKSIZE=244,LRECL=24),DISP=(NEW,CATLG),
//  DSNAME=TEST,VOLUME=SER=222222
```

The data set is variable blocked format, each record will be no larger than 20 data bytes (LRECL = data bytes + 4), and each block will contain ten records; there is space for up to 1000 blocks, the data set will be cataloged, its name is TEST, and it will reside on the volume with serial number 222222.

```
//GO.SIMU10  DD  DSNAME=TEST,DISP=OLD
```

With this DD card the cataloged data set named TEST is to be referenced; since it is cataloged, the DCB, VOLUME, and UNIT parameters will be provided by the system; thus, the data set created in the previous example could be referenced in another job by the above DD card.

3. Tape data sets:

```
//GO.SIMU15  DD  UNIT=TAPE7,LABEL=(1,NL),VOLUME=SER=SCRTCH,
//  DCB=(RECFM=FB,LRECL=100,BLKSIZE=3000,DEN=2,TRTCH=C)
```

In this example an unlabeled tape named SCRTCH is to be used for a temporary data set; the records are each 100 bytes long and there are 30 records per block; note that there is no SPACE parameter, but the LABEL parameter has been added (NL indicates an unlabeled tape);

also note the additional DCB parameters of DEN and TRTCH; these are used to indicate tape density and recording mode, respectively; this example shows parameters for an 800 bpi, seven-track tape with the translate feature off and the data conversion feature on. The TRTCH parameter is only applicable to seven-track tapes.

```
//GO.SIMU12  DD  UNIT=TAPE9,LABEL=(1,NL),VOLUME=SER=SCRTCH,
//  DCB=(RECFM=VB,BLKSIZE=112,LRECL=54,DEN=3)
```

Here, the data set is variable blocked, the density is 1600 bpi, the tape is unlabeled, and it is nine-track; note that there is no TRTCH parameter.

```
//GO.SIMU09  DD  UNIT=TAPE9,LABEL=(,SL),VOLUME=001234,
//  DCB=(RECFM=FB,LRECL=100,BLKSIZE=100,DEN=3),DISP=(NEW,KEEP),
//  DSNAME=TAPEDATA
```

This is an example of a nine-track, 1600 bpi, standard labeled tape with volume serial number 001234; the data set name is TAPEDATA and it will be kept.

```
//GO.SIMU04  DD  UNIT=TAPE9,LABEL=(,SL),VOLUME=SER=001234,
//  DSNAME=TAPEDATA,DISP=OLD
```

This DD card could be used to read the data set created in the above example; note that there is no DCB parameter since the DCB information is kept in the tape label. If the tape were unlabeled, the DCB parameters would have to be provided.

4. Other data sets:

```
//GO.SIMU07  DD  DUMMY,DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//GO.SIMU08  DD  SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
```

Dummy and Sysout data sets always require DCB parameters. For card inputs; however,

```
//GO.SIMU04  DD  *
```

the system supplies DCB information--although a block size can be optionally assigned as in

```
//GO.SIMU04  DD  *,DCB=BLKSIZE=800.
```

VI. CALLING ASSEMBLER LANGUAGE ROUTINES

This section describes the conventions used in all SIMSCRIPT II routines; it discusses how an assembler language program must be written to interface with a SIMSCRIPT II calling routine. The discussion is divided into four parts: routine prologue, routine body programming conventions, returning control to the calling program, routine epilogue.

THE PROLOGUE

The general form of a prologue is shown in Fig. 1. Shown first is the routine name, which is the name used in the CALL statement or function reference, after the following conversions have been applied to it:

(a) The name is truncated to seven characters

(b) An R is prefixed to the truncated name[†]

(c) All decimal points are converted to dollar signs.

For example:

```
RANDOM.F becomes RRANDOM$
SIN.F    becomes RSIN$F
```

The variables L and X are computed by the formulae:

$$L = 52 + 4*(g + y + r)$$
$$X = 4*(y + r)$$

where g and y are the number of *giving* and *yielding* arguments in the routine's calling sequence, and r is the number of *recursive* local variables[††] used by the routine.

[†] An L is prefixed if the routine is used as a left-handed function.

[††] Assuming that each recursive local variable is four bytes long.

```
1     Rname     CSECT
2               USING *,15
3               USING H,7,8,9
4               B     F
5               DC    AL2(L)
6               DC    AL2(0)
7               DC    AL2(X)
8               DC    AL2(T-*+10)
9     F         L     2,0(1)
10              DROP  15
11              BALR  0,2
12    H         EQU   *
```

Fig. 1--Prologue format

The BALR instruction (line 11) passes control to XREC, a SIMSCRIPT II system routine that establishes a *save area*, and returns the address of the save area's first byte in general register 6. The save area, illustrated in Fig. 2, is divided into four sections.

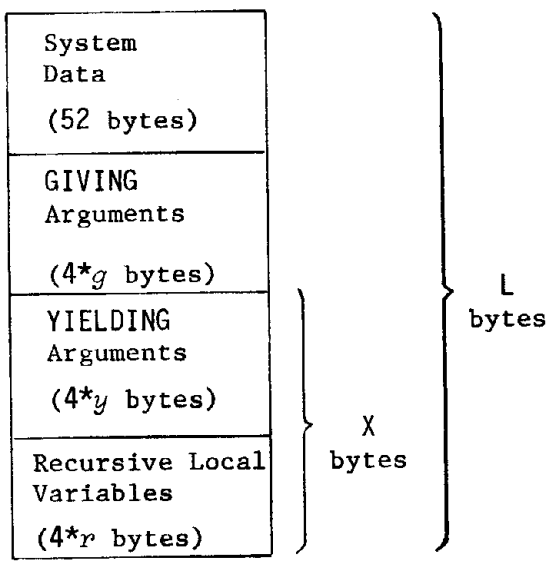


Fig. 2--Save area format

The first section, labeled System Data, contains the following (in order): the values of general registers 6 through 14, and floating point registers 0, 2, and 4, saved by XREC at the time of entry to this routine. It should not be used by the programmer. The second section, labeled GIVING Arguments, contains the *values* of the routine's giving

arguments. The third section, labeled YIELDING Arguments, is the area in which yielding argument *values* are stored prior to return to the calling routine. The fourth section, labeled Recursive Local Variables, is the area in which all recursive local variables of the routine are located. When a routine is called, XREC zeroes out the yielding and recursive areas.

THE BODY

The routine body must conform to several conventions concerning:

(a) accessing GIVING argument values
(b) returning YIELDING argument values
(c) accessing recursive local variables
(d) using registers.

GIVING Arguments

General register 6 points to the first byte of the save area. To access the value of the first GIVING argument, the programmer must refer to the first four-byte word after the System Data section of the save area, i.e., the word whose address is 52 bytes greater than the contents of general register 6. The first GIVING argument is at 52(6), the second at 56(6), etc. The i-th giving argument is addressed as [52 + 4*(i - 1)](6).

As an example consider the calling sequence:

```
CALL GET.DATA GIVEN I,J AND K YIELDING M AND N
```

The value of I is stored at 52(6), the value of J at 56(6), and the value of K at 60(6).

YIELDING Arguments

The first YIELDING argument follows the last GIVING argument; in general, the address of the i-th YIELDING argument value is [52 + 4*g + 4*(i - 1)](6). In the above example, the value that will be stored in M when control returns to the calling routine is at 64(6). The value that will be stored in N is at 68(6).

Recursive Local Variables

The remainder of the save area is used to store recursive local variable values. The programmer is free to organize this section in whatever manner he chooses. The only restriction is that the area must be an integral number of four bytes.

Registers

General registers 1, 6, 7, 8, and 9 are reserved for SIMSCRIPT II use and should not be used by the programmer for purposes other than those described. Register 1 points to the "master array" (called XMAS) that contains the SIMSCRIPT system variables READ.V, TIME.V, LINES.V, etc., and the global variables defined by the user to be "IN ARRAY *i*". Register 6 points to the current save area. Registers 7, 8, and 9 are used as base registers; 8 is needed only if a routine is more than 4,096 bytes long; 9 is needed only if a routine is longer than 8,192 bytes. If 1 and 6 are used, their contents should be saved immediately after the prologue and restored before returning to the calling routine.

If the programmer-written routine is to be used as a right-handed function, the function value must be returned in floating-point register 6 if the function is real or in general register 5 if it is otherwise. If the programmer-written routine is to be used as a left-handed function, the "ENTER WITH" value is found in floating-point register 6 or general register 5 depending on the mode of the function.

Readability

A simple technique can be used to improve a routine's readability. By putting EQU statements after the prologue's CSECT statement, the programmer can refer to arguments by their mnemonic representation rather than by their byte displacement. For example, in a routine written for the calling statement above, one could write:

```
I EQU 52
J EQU 56
K EQU 60
M EQU 64
N EQU 68
```

and then use M(6) rather than 64(6).

RETURNING CONTROL TO THE CALLING PROGRAM

To return control to the calling routine, two return sequence statements must be executed:

```
L  15,4(1)
BR 15
```

These statements pass control to XRET, which returns the save area to the free storage pool and returns to the calling routine.

THE EPILOGUE

```
  LTORG
T EQU   *
  END
```

T is the address of the last byte in the routine; it insures that the entire routine is releasable. (Note that T is referenced in the prologue.)

Library routines can be created most simply by insuring that they use only local variables or defined system variables. When several library routines have to communicate with one another, this can be done by compiling them with their own preamble, making sure that all global names are unique,† and putting the object decks of this "library preamble" and its library routines together. As this preamble has the same name (PRMB) as that of the preamble of the program it will be loaded with, the name of the library preamble must be changed. This is done by changing the assembly cards of the compiled "library preamble" in the following way:

```
The first assembly card is:  PRMB  CSECT
It must be changed to:       name  CSECT
```

where *name* is unique. To be safe, *name* should not start with G, I, R, or L.

---

†We suggest that a person constructing library routines use global variable names that the SIMSCRIPT II programmer is urged to avoid, e.g., names that start with a letter followed by a period, or that end with a period followed by a letter, or that do not appear in Secs. IV-VII of P. J. Kiviat and R. Villanueva, *The SIMSCRIPT II Programming Language: Reference Manual*, The RAND Corporation, RM-5776-PR, October 1968.

VII. STORAGE ALLOCATION DURING EXECUTION

PERMANENT ENTITIES

Permanent entities have their attributes stored as arrays. Attributes that are interpacked are reserved and released with the same statement.

TEMPORARY ENTITIES

When temporary entities of a particular record size are initially created, a GETMAIN is executed to get the correct number of bytes from the operating system. If an odd number of words is requested, twice the number asked for is delivered and two odd-size records split from them.

When temporary entities are destroyed they are put in sets that contain records of the same size. As long as records of a particular size are available they are taken from these sets rather than from the system. Records of two words or more are "branded" when they are destroyed with the hexadecimal characters EFOFEFEF in the second word. Before an entity record is destroyed it is checked to see if it already has a brand; if it has, it is not put in the set of available records of its size. This assures that in spite of a programming error that destroys the same entity twice, the "available record" set integrity is maintained.

If at some time during the execution of a program a CREATE or RESERVE statement is executed, and for any reason there is no core available, all *even-size* records (words, not bytes) are returned to the system by FREEMAIN commands. This space is then used for the allocation. Odd-size records are not returned. Since this makes it possible for a program to run out of core while there are free odd-size records available, programmers with extremely large problems may choose to define all entity records to have an even number of words. One word may be wasted with each CREATE, but there will be no residual odd-size record problem.

USER-DEFINED GLOBAL VARIABLES

Global variables declared by statements of the form,

THE SYSTEM HAS A *variable* IN ARRAY i

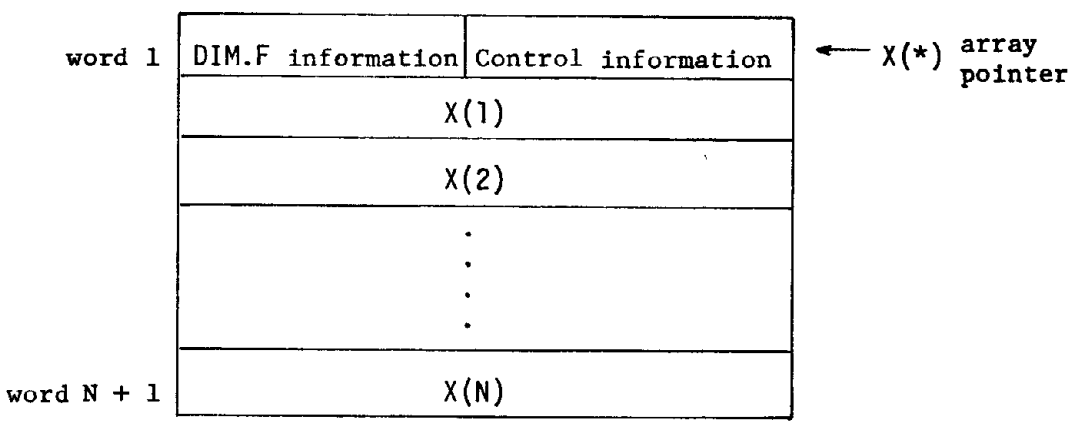
are displaced 1196 + 4*i bytes from general register 1. Variables not placed in a particular word can be found by looking at the assembly listing of the preamble, i.e., the PRMB listing. For a variable called X the code

```
        ENTRY GX
GX      DC    F'0'
```

is generated. The relative location of GX within PRMB is used along with the link-edit map to find the absolute core location of X during execution.

ARRAYS

All arrays are stored as vectors; multidimensional arrays are constructed of linked vectors. Each array vector is of the form:



The first half of the first word contains the size of the vector as defined in a RESERVE statement. The second half contains two items of control information: (1) the sign bit is negative (-) if the vector contains pointers and is positive (+) if the vector contains data; (2) the magnitude indicates the number of actual computer words contained in the vector, excluding the control word. Computer words are allocated to array vectors in 8-byte multiples. Arrays are reserved and released by executing GETMAIN and FREEMAIN instructions, respectively.

VIII. RANDOM NUMBER GENERATOR AND STATISTICAL FUNCTIONS

The pseudorandom number generator is described in an article in the *Communications of the Association for Computing Machinery*, February 1969. The starting values used for SEED.V(i), i = 1, 2, ..., 10 are:

```
SEED.V(1)  = 2116429302
SEED.V(2)  =  683743814
SEED.V(3)  =  964393174
SEED.V(4)  = 1217426631
SEED.V(5)  =  618433579
SEED.V(6)  = 1157240309
SEED.V(7)  =   15726055
SEED.V(8)  =   48108509
SEED.V(9)  = 1797920909
SEED.V(10) =  477424540
```

Each sequence generates 100,000 numbers. Starting with an initial seed of 524287, a sequence of random numbers is generated. The 100,001 number from this sequence is the initial value of SEED.V(2), the 100,001 number generated from the second sequence is the initial value of SEED.V(3), etc. SEED.V(1) is the 100,001 number in the sequence beginning with SEED.V(10).

To replace the system random number generator one need only write his own routine named RANDOM.F, i.e., RRANDOM$ (see Sec. VI).

The routines that generate samples from different statistical distributions are best described by their respective SIMSCRIPT II programs.† To replace them with other algorithms or to add required error checks, one need only write his own routines with the same names.

SIMSCRIPT II Library Name	External Name in Program
BETA.F	RBETA$F
BINOMIAL.F	RBINOMIA
ERLANG.F	RERLANG$
EXPONENTIAL.F	REXPONEN
GAMMA.F	RGAMMA$F
GAMMAJ.F	RGAMMAJ$
LOG.NORMAL.F	RLOG$NOR
NORMAL.F	RNORMAL$
POISSON.F	RPOISSON
RANDI.F	RRANDI$F
RANDOM.F	RRANDOM$
UNIFORM.F	RUNIFORM
WEIBULL.F	RWEIBULL

†Further discussion of the algorithms can be found in Chapter 4 of Naylor, T. H., Balintfy, J. L., Burdick, D. S., and Chu, K., *Computer Simulation Techniques*, John Wiley and Sons, New York, 1968.

When replacing a system function, such as GAMMA.F,SIN.F or SQRT.F, the routine name must not be declared in the preamble as a function. The routine is automatically declared by the system and an additional DEFINE statement is taken as an error.

The name ERR.F that appears in all the statistical functions is a left-handed system function that terminates a program after printing the error code given it by the right-hand side expression, e.g., LET ERR.F = 46 terminates after printing error code 46.

```
ROUTINE BETA.F(K1, K2, STREAM)
ROUTINE BETA.F(K1, K2, STREAM)
DEFINE K1 AND K2 AS REAL VARIABLES
DEFINE X AS A REAL VARIABLE
DEFINE STREAM AS AN INTEGER VARIABLE
IF K1<=0, LET ERR.F=147  ELSE
IF K2<=0, LET ERR.F=148  ELSE
LET X=GAMMA.F(K1, K1, STREAM)
RETURN WITH X / (X + GAMMA.F (K2, K2, STREAM))
END
```

```
ROUTINE BINOMIAL.F(N,P,STREAM)
DEFINE N,I,SUM AND STREAM AS INTEGER VARIABLES
DEFINE P AS A REAL VARIABLE
IF N<=0, LET ERR.F=141  ELSE
IF P<=0, LET ERR.F=142  ELSE
''  IF N IS A LARGE INTEGER OR INCORRECTLY INITIALIZED TO A REAL VALUE
''      COMPUTATION TIME FOR THIS FUNCTION MAY BE EXCESSIVE
FOR I=1 TO N, WHEN RANDOM.F(STREAM)<=P, ADD 1 TO SUM
RETURN WITH SUM
END
```

```
ROUTINE ERLANG.F(MU, K, STREAM)
DEFINE MU AND E AS REAL VARIABLES
DEFINE I, K AND STREAM AS INTEGER VARIABLES
IF MU<=0, LET ERR.F=133  ELSE
IF K<=0, LET ERR.F=134  ELSE
LET E=1
''  IF K IS A LARGE INTEGER OR INCORRECTLY INITIALIZED TO A REAL VALUE
''      COMPUTATION TIME FOR THIS FUNCTION MAY BE EXCESSIVE
FOR I=1 TO K, LET E=E*RANDOM.F(STREAM)
RETURN WITH -LOG.E.F(E) * MU / K
END
```

```
ROUTINE EXPONENTIAL.F(MU, STREAM)
DEFINE MU AS A REAL VARIABLE
DEFINE STREAM AS AN INTEGER VARIABLE
IF MU<=0, LET ERR.F=132  ELSE
RETURN WITH -MU*LOG.E.F(RANDOM.F(STREAM))
END



ROUTINE GAMMA.F(MEAN, K, STREAM)
''NOTE:  FOR NON-INTEGRAL VALUED K<5, THIS ALGORITHM APPROXIMATES
''  THE GAMMA DISTRIBUTION RELATIVELY POORLY COMPARED TO JOHNK'S METHOD
''  (GAMMAJ.F).
DEFINE I,MEAN,K,KK AND E AS REAL VARIABLES
DEFINE STREAM AS AN INTEGER VARIABLE
IF K<1 RETURN WITH GAMMAJ.F(MEAN,K,STREAM) ELSE
IF MEAN<=0, LET ERR.F=145  ELSE
IF K<=0, LET ERR.F=146  ELSE
LET E=1
LET KK=TRUNC.F(K)
IF RANDOM.F(STREAM) > K-KK, LET K=KK  GO TO A  ELSE LET K=KK+1
'A'  FOR I=1 TO K, LET E=E*RANDOM.F(STREAM)
RETURN WITH (MEAN/K)*(-LOG.E.F(E))
END



ROUTINE GAMMAJ.F(MEAN,K,STREAM)
''CALCULATION OF GAMMA DISTRIBUTED VARIATES BY  JOHNK'S METHOD.
''THIS ALGORITHM MUST BE USED FOR 0<K<1 INSTEAD OF GAMMA.F, AND SHOULD
''BE USED FOR NON-INTEGRAL VALUES OF K<5, ALTHOUGH IT IS 2.5 TO 3
''TIMES SLOWER THAN GAMMA.F.  FOR FURTHER DISCUSSIOR SEE "GENERATING
''GAMMA DISTRIBUTED VARIATES FOR COMPUTER SIMULATION MODELS",
''M. B. BERMAN,THE RAND CORPORATION, R-641-PR, FEBRUARY 1971.
DEFINE MEAN,K,KK,I,Z,A,B,D,E,X,Y, AND W AS REAL VARIABLES
DEFINE STREAM AS AN INTEGER VARIABLE
IF MEAN<=0, LET ERR.F=145 ELSE
IF K<=0,  LET ERR.F=146 ELSE
LET Z=0
LET KK=TRUNC.F(K)
LET D=K-KK
IF KK=0, GO TO BETA ELSE
LET E=1
FOR I=1 TO KK, LET E=E*RANDOM.F(STREAM)
LET Z=-LOG.E.F(E)
IF D=0,  RETURN WITH Z*(MEAN/K) ELSE
'BETA'
LET A=1/D     LET B=1/(1-D)
'NEXT'
LET X=RANDOM.F(STREAM)**A
LET Y=RANDOM.F(STREAM)**B+X
IF Y<=1, GO OUT ELSE GO TO NEXT
'OUT'
LET W=X/Y
LET Y=-LOG.E.F(RANDOM.F(STREAM))
RETURN WITH (Z+W*Y)*(MEAN/K)
END
```

```
ROUTINE LOG.NORMAL.F(MU, SIGMA, STREAM)
DEFINE MU AND SIGMA AS REAL VARIABLES
DEFINE S AND U AS REAL VARIABLES
DEFINE STREAM AS A INTEGER VARIABLE
IF MU<=0, LET ERR.F=135  ELSE
IF SIGMA <=0, LET ERR.F=136  ELSE
LET S=LOG.E.F((SIGMA*SIGMA)/(MU*MU)+1)
LET U=LOG.E.F(MU)-0.5*S
RETURN WITH EXP.F(NORMAL.F(U, SQRT.F(S), STREAM))
END



ROUTINE NORMAL.F(MU, SIGMA, STREAM)
DEFINE MU AND SIGMA AS REAL VARIABLES
DEFINE STREAM AS AN INTEGER VARIABLE
NORMALLY, MODE IS REAL
IF SIGMA<=0, LET ERR.F=137  ELSE
'A' LET X=RANDOM.F(STREAM)  LET Y=2*RANDOM.F(STREAM)-1
LET XX=X*X  LET YY=Y*Y   LET S=XX+YY
IF S > 1, GO TO A  ELSE
LET R=SQRT.F(-2*LOG.E.F(RANDOM.F(STREAM)))/S
RETURN WITH MU+(XX-YY)*R*SIGMA
END



ROUTINE POISSON.F(MU, STREAM)
DEFINE MU, Y AND X AS REAL VARIABLES
DEFINE STREAM AND N AS INTEGER VARIABLES
IF MU<=0, LET ERR.F=138  ELSE
IF MU > 6,
  RETURN WITH ABS.F(NORMAL.F(MU,SQRT.F(MU),STREAM))
  OTHERWISE
LET Y=EXP.F(-MU)   LET X=1
'A' LET X=X*RANDOM.F(STREAM)
IF X < Y, RETURN WITH N  ELSE ADD 1 TO N  GO TO A
END



ROUTINE RANDI.F(I,J,STREAM)
DEFINE I,J AND STREAM AS INTEGER VARIABLES
IF J<I, LET ERR.F=139  ELSE
RETURN WITH TRUNC.F(RANDOM.F(STREAM)*(J-I+1))+I
END



ROUTINE UNIFORM.F(A, B, STREAM)
DEFINE A AND B AS REAL VARIABLES
DEFINE STREAM AS AN INTEGER VARIABLE
IF B<A, LET ERR.F=140 ELSE
RETURN WITH A+RANDOM.F(STREAM)*(B-A)
END
```

```
ROUTINE WEIBULL.F(SHAPE, SCALE, STREAM)
DEFINE SHAPE AND SCALE AS REAL VARIABLES
DEFINE STREAM AS AN INTEGER VARIABLE
IF SHAPE<=0. LET ERR.F=143  ELSE
IF SCALE<=0, LET ERR.F=144  ELSE
RETURN WITH SCALE/((-LOG.E.F(RANDOM.F(STREAM)))**SHAPE)
END
```

IX. INSTALLING THE COMPILER

SYSTEM REQUIREMENTS

SIMSCRIPT II is designed to run under OS/360. It is currently running under RELEASE 19 of MVT on a model 65. There are no known OS release dependencies.

The compiler requires a minimum core region of 150K. However, since it dynamically expands depending on the nature of input source statements, more core may be needed. Generally, compiled programs will require at least a 52K region (see Sec. II).

SIMSCRIPT II accepts no parameters either at compile time or execution time, but standard options (placed on the PARM field of the EXEC card) are available to the assembly and link-edit phases.†

DISTRIBUTED SYSTEM TAPE

The SIMSCRIPT II system is distributed on a nine-track, 800 bpi labeled tape, and contains the following data sets:

1. SYS1.SIM2LIB--an unloaded partition data set used as both an object and execution time library.
2. SYS1.SIM2--an unloaded partition data set containing the SIMSCRIPT II compiler and an OS Assembler F interface routine.
3. SYS1.SIM2GEN--JCL to generate the compiler from tape to disk.
4. SYS1.SIMPROC--JCL to be added to the installation's procedure library.
5. SYS1.SIMSAMP--a sample SIMSCRIPT II program.
6. SIM2.COMPILER--compiler source code.
7. SIM2.LIBRARY--SIMSCRIPT library routine source code.
8. SIM2.SUPPORT--support programs for the compiler.

GENERATING THE COMPILER

The following is an example of JCL that can be used to punch the SYS1.SIM2GEN file. To punch other files, change the LABEL= and DSN= to appropriate label number and label name.

---

†For the assembler, see form C26-3756 and for the linkage editor, see form C28-6538, both IBM publications.

```
//  EXEC  PGM=IEBPTPCH
//SYSPRINT  DD  SYSOUT=A
//SYSUT1    DD  DSN=SYS1.SIM2GEN,UNIT=TAPE9,VOL=SER=003381,LABEL=(3,SL)
//SYSUT2    DD  SYSOUT=B
//SYSIN     DD  *,DCB=BLKSIZE=80
 PUNCH
/*
```

The following is the JCL (contained in SYS1.SIM2GEN) that is used to load the SIMSCRIPT II library, compiler and assembler interface.

```
    SIMSCRIPT II TAPE TO DISK GENERATION PROCEDURES

THIS JOB ALLOCATES SPACE TO THE DATA SETS Y6700.SIM2LIB AND Y6700.SIM2
LOCATED ON DISK RAND02.

THIS JOB THEN MOVES THE FIRST TWO FILES OF TAPE 003381 TO THE PREALLOCATED DATA
SETS ON RAND02.  THE FILES CONTAIN THE SIMSCRIPT II LIBRARY, COMPILER, AND
ASSEMBLER INTERFACE ROUTINE.

//Y6700#SS JOB (8317,100,77),'TAPE TO DISK',CLASS=A
//    EXEC PGM=IEFBR14
//DD1 DD DSN=Y6700.SIM2LIB,DISP=(NEW,CATLG),UNIT=2314,VOL=SER=RAND02,
//    SPACE=(1024,(100,50,50),,CONTIG)
```



```
//*  SIMSCRIPT II COMPILE, ASSEMBLE, LINK EDIT, EXECUTE
//SIM  EXEC  PGM=SIM2,REGION=150K
//STEPLIB  DD  DSNAME=Y6700.SIM2,DISP=SHR
//LOADLIB  DD  DSNAME=Y6700.SIM2LIB,DISP=SHR
//SIMU03  DD  SYSOUT=A,DCB=(RECFM=FBA,BLKSIZE=1330,LRECL=133,BUFNO=1)
//SIMU06  DD  DSNAME=&TEMP,DISP=(,PASS),UNIT=(SYSDA,SEP=(LOADLIB)),     CONTINUE
//             SPACE=(80,(5000,1000)),                                  CONTINUE
//  DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)
//SIMU05  DD  DDNAME=SYSIN
//SIM.SYSIN  DD  *
''SAMPLE SIMSCRIPT II PROGRAM
NORMALLY MODE IS INTEGER
RESERVE D AS 132   LET D(132)=1
WHILE CARRY IS ZERO, DO THE FOLLOWING.....
   FOR K=1 TO 132 WITH D(K) NOT ZERO, FIND THE FIRST CASE
   WRITE AS /, B K
   FOR KK=K TO 132, WRITE D(KK) AS I 1
ALSO FOR K BACK FROM 132 TO 1, DO
   LET SUM=2*D(K)
   LET D(K)=MOD.F(SUM,10)+CARRY
   LET CARRY=DIV.F(SUM,10)
LOOP
END
/*
```

Y O U M U S T P A Y A T T E N T I O N TO:

RAND05: DISK CONTAINING THE WORK AREA FOR THE UTILITY PROGRAMS
RAND02: DISK CONTAINING THE COMPUTER DATA SETS
003381: SERIAL NUMBER OF THE INPUT TAPE
SYS1.SIM2LIB: THE NAME OF THE DATA SET IN THE FIRST TAPE FILE
SYS1.SIM2: THE NAME OF THE DATA SET IN THE SECOND TAPE FILE
Y6700.SIM2LIB: THE NAME WE HAVE GIVEN TO THE DATA SET WE ARE MOVING THE FIRST FILE ON THE TAPE INTO. THIS NAME IS OPTIONAL FOR YOU AND MUST BE REFLECTED IN YOUR PROCEDURES FOR USING THE COMPILER
Y6700.SIM2: SAME AS ABOVE BUT FOR THE OTHER TAPE FILE AND ITS DATA SET
TAPE9: TAPE UNIT USED FOR INPUT TAPE

SOME NOTES ON PROCEDURES

The distributed procedures are designed to be used with an MVT system; they are listed on the following pages. Each installation should review the JCL and make whatever changes it feel necessary before applying the procedures to its system. You may or may not wish to catalogue SIM2, SIM2CA, and SIM2LG. The sample program (SYS1.SIMSAMP) is shown imbedded in the JCL to indicate where a program deck goes if you do not catalogue the procedures.

The following is a capsule summary of data sets of the compile, assembly, and execution steps:

A. The compile step (SIM)

SIMU05--the source input data set
SIMU03--print urit
SIMU06--output to Assembler F

B. The assembly step (ASM)

SYSUT1-SYSUT3, SYSPRINT } same as the standard Assembler F data sets
SYSIN--input to the interface routine
INPUT--a temporary data set in which input is passed to the assembler
SYSGO--the assembler object module data set
SYSGOX--a data set in which the interface routine copies the object module and passes it on to the linkage editor

C. The execution step (GO)

SIMU03--the standard print unit
SIMU02--the standard punch unit
SIMU05--the standard read unit
LOADLIB--an execution time data set from which error messages are dynamically loaded

```
//*  SIMSCRIPT II COMPILE, ASSEMBLE, LINK EDIT, EXECUTE
//SIM  EXEC  PGM=SIM2,REGION=150K
//STEPLIB  DD  DSNAME=Y6700.SIM2,DISP=SHR
//LOADLIB  DD  DSNAME=Y6700.SIM2LIB,DISP=SHR
//SIMU03  DD  SYSOUT=A,DCB=(RECFM=FBA,BLKSIZE=1330,LRECL=133,BUFNO=1)
//SIMU06  DD  DSNAME=&TEMP,DISP=(,PASS),UNIT=(SYSDA,SEP=(LOADLIB)),     CONTINUE
//             SPACE=(80,(5000,1000)),                                  CONTINUE
//  DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)
//SIMU05  DD  DDNAME=SYSIN
//SIM.SYSIN  DD  *
''SAMPLE SIMSCRIPT II PROGRAM
NORMALLY MODE IS INTEGER
RESERVE D AS 132   LET D(132)=1
WHILE CARRY IS ZERO, DO THE FOLLOWING.....
   FOR K=1 TO 132 WITH D(K) NOT ZERO, FIND THE FIRST CASE
   WRITE AS /, B K
   FOR KK=K TO 132, WRITE D(KK) AS I 1
ALSO FOR K BACK FROM 132 TO 1, DO
   LET SUM=2*D(K)
   LET D(K)=MOD.F(SUM,10)+CARRY
   LET CARRY=DIV.F(SUM,10)
LOOP
END
/*
//ASM  EXEC  PGM=MULTASM,COND=(16,EQ,SIM),PARM='DECK,LOAD,NOXREF',
//       REGION=104K
//STEPLIB  DD  DSNAME=Y6700.SIM2,DISP=SHR
//SYSLIB  DD  DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1  DD  UNIT=SYSDA,SPACE=(1700,(400,50),,,ROUND)
//SYSUT2  DD  UNIT=(SYSDA,SEP=(SYSUT1)),SPACE=(1700,(400,50),,,ROUND)
//SYSUT3  DD  UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),                        CONTINUE
//             SPACE=(1700,(400,50),,,ROUND)
//SYSPUNCH  DD  SYSOUT=B
//SYSPRINT  DD  SYSOUT=A
//SYSGO  DD  DSNAME=&LOAD,DISP=(NEW,DELETE),SPACE=(80,(200,50)),       CONTINUE
// UNIT=SYSDA
//SYSGOX  DD  DSNAME=&LOADSET,UNIT=SYSDA,DISP=(MOD,PASS),              CONTINUE
// SPACE=(80,(1000,500),RLSE),DCB=(RECFM=FB,BLKSIZE=80)
//INPUT  DD  DSNAME=&TEMPRY,UNIT=SYSDA,DISP=(NEW,DELETE),              C
//             SPACE=(TRK,(40,5)),DCB=(,RECFM=FB,BLKSIZE=3600,BUFNO=1)
//SYSIN  DD  DSNAME=&TEMP,UNIT=SYSDA,DISP=(OLD,DELETE)
//LKED  EXEC  PGM=IEWL,PARM='LIST,MAP',COND=(16,EQ,SIM),REGION=104K
//SYSLIB  DD  DSNAME=Y6700.SIM2LIB,DISP=SHR
//SYSLIN  DD  DSNAME=&LOADSET,DISP=(OLD,DELETE)
//  DD  DDNAME=SYSIN,DCB=(BLKSIZE=80)
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=605
//SYSUT1  DD  UNIT=SYSDA,SPACE=(1024,(50,20),,,ROUND)
//SYSLMOD  DD  DSNAME=&GOSET(GO),UNIT=(SYSDA,SEP=(SYSUT1)),            CONTINUE
//             SPACE=(1024,(50,20,1)),DISP=(,PASS)
//GO  EXEC  PGM=*.LKED.SYSLMOD,COND=(16,EQ),REGION=52K
//LOADLIB  DD  DSNAME=Y6700.SIM2LIB,DISP=SHR
//SIMU02  DD  SYSOUT=B,DCB=(RECFM=FBA,LRECL=81,BLKSIZE=81)
//SIMU03  DD  SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=133)
//SIMU05  DD  DDNAME=SYSIN
//GO.SYSIN  DD  *
/*
```

005

```
//*  SIMSCRIPT II COMPILE AND ASSEMBLE
//SIM  EXEC  PGM=SIM2,REGION=150K
//STEPLIB  DD  DSNAME=Y6700.SIM2,DISP=SHR                                    00002000
//LOADLIB  DD  DSNAME=Y6700.SIM2LIB,DISP=SHR
//SIMU03  DD  SYSOUT=A,DCB=(RECFM=FBA,BLKSIZE=1330,LRECL=133,BUFNO=1)        C00004000
//SIMU06  DD  DSNAME=&TEMP,DISP=(,PASS),UNIT=SYSDA,                          C00005000
//              SPACE=(80,(5000,1000)),
//  DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)                                      00008000
//SIMU05  DD  DDNAME=SYSIN
//SIM.SYSIN  DD  *
/*
//ASM EXEC PGM=MULTASM,COND=(16,EQ,SIM),PARM='DECK,NOLOAD,NOXREF',
//       REGION=104K
//STEPLIB  DD  DSNAME=Y6700.SIM2,DISP=SHR
//SYSLIB  DD  DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1  DD  UNIT=SYSDA,SPACE=(1700,(400,50),,,ROUND)                       00011000
//SYSUT2  DD  UNIT=(SYSDA,SEP=(SYSUT1)),SPACE=(1700,(400,50),,,ROUND)        00012000
//SYSUT3  DD  UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),                              C00013000
//              SPACE=(1700,(400,50),,,ROUND)                                00014000
//SYSPUNCH  DD  SYSOUT=B                                                     00015000
//SYSPRINT  DD  SYSOUT=A                                                     00016000
//INPUT  DD  DSNAME=&TEMPRY,UNIT=SYSDA,DISP=(NEW,DELETE),                    C
//   SPACE=(TRK,(20,5)),DCB=(RECFM=FB,BLKSIZE=3600,BUFNO=1)                  00018000
//SYSGOX  DD  DUMMY,DCB=BLKSIZE=80
//SYSIN  DD  DSNAME=&TEMP,UNIT=SYSDA,DISP=(OLD,DELETE)



//*  SIMSCRIPT II LINKEDIT AND EXECUTE
//LKED  EXEC  PGM=IEWL,PARM='LIST,MAP',REGION=104K                           00001000
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=605                                       00003000
//SYSUT1  DD  UNIT=SYSDA,SPACE=(TRK,(10,10))                                 00004000
//SYSLIB  DD  DSNAME=Y6700.SIM2LIB,DISP=SHR                                  C00005000
//SYSLMOD  DD  DSNAME=&GOSET(GO),DISP=(,PASS),UNIT=SYSDA,                    00006000
//  SPACE=(3025,(50,20,1))
//SYSLIN  DD  DDNAME=SYSIN,DCB=(BLKSIZE=80)
//LKED.SYSIN  DD  *
/*
//GO  EXEC  PGM=*.LKED.SYSLMOD,COND=(5,LT,LKED),REGION=52K                   00009000
//LOADLIB  DD  DSNAME=Y6700.SIM2LIB,DISP=SHR
//SIMU02  DD  SYSOUT=B,DCB=(RECFM=FBA,LRECL=81,BLKSIZE=81)                   00011000
//SIMU03  DD  SYSOUT=A,DCB=(RECFM=FBA,BLKSIZE=133,LRECL=133)                 00012000
//SIMU05  DD  DDNAME=SYSIN
//GO.SYSIN  DD  *
/*                                                                                00
```

DEPARTMENT OF THE AIR FORCE
FEDERAL COMPUTER PERFORMANCE EVALUATION AND SIMULATION CENTER (AFDAA)
WASHINGTON, D.C. 20330

18 APR 1977

SHARE Program Library Agency
Triangle Universities Computation Center
Post Office Box 12076
Research Triangle Park, NC 27709

ATTN: Librarian

Reference Program Catalog Number: 360D-03.2.014 - SIMSCRIPT II

The attached user report was forwarded to me as the listed author of the referenced SHARE Program. The user report makes the suggestion that the library software be modified to reflect several identified problems. This letter is to inform you that the SIMSCRIPT II program now resident in the SHARE library has not been modified since 1969; there are no activities in progress that I am aware of that might make such modifications. My organization, for example, has dealt with a commercial version of the SIMSCRIPT II system for the past four years.

Accordingly, I recommend that this user report be distributed along with copies of the library program when it is requested, as no modifications to the library program will be made in the future.

Sincerely yours,

PHILIP J. KIVIAT
Technical Director

SHARE LIBRARY USER REPORT

PROGRAM CATALOG NUMBER 360D-03.2.014

COMPUTER MODEL 370/168

OPERATING SYSTEM MVS REL 3.7

1. Were you able to implement this program using the instructions supplied?

YES, AFTER AN INITIAL PROBLEM WITH FILE NAMES.

2. Did the program run as documented?

YES, BUT ONLY AFTER SEVERAL ERRORS WERE CORRECTED.

3. Was the documentation adequate? NOT QUITE. THERE WAS INSUFFICIENT DOCUMENTATION FOR THE MULTIPLE ASSEMBLER INTERFACE, AND CORRECT TAPE FILE NAMES WERE HARD TO FIND.

4. Did you find any errors in the program?

YES. SEE ATTACHED NOTES.

5. Do you have any suggestions to improve the distributed material?

YES. SEE ATTACHED LIST.

6. Comments:

NAME R.J. WHATMOUGH DATE 13/77 INSTALLATION CODE (if any) WA

ADDRESS COMPUTING SERVICES GROUP, WEAPONS RESEARCH ESTABLISHMENT, BOX 2151, SALISBURY, 5108, SOUTH AUSTRALIA.

PLEASE RETURN TO:

SHARE Program Library Agency
Triangle Universities Computation Center
Post Office Box 12076
Research Triangle Park, NC 27709

ERRORS IN SIMSCRIPT II COMPILER AND SAMPLE PROBLEM

The following problems were encountered and solutions applied to make the sample problem run correctly:

1. The compiler ABENDED with error 204 (protection) while processing the first few statements, after generating a 'PRMB CSECT' assembler statement with nulls instead of blanks.

   The error was traced to the library routine RRES$R (reserve storage for user arrays). After the GETMAIN at source statement XRES0870, register 0 is assumed to hold the length of the gotten area, but the OS/VS2 GETMAIN overwrites it.

   The source code was extracted from the distribution tape and modified by inserting after line XRES0870 the statement "L R0,ALPHA" to restore the register, then re-assembled. The library load module, and the RRES$R CSECT in the compiler, were replaced using the linkage editor.

2. The multiple assembler produced no SYSPRINT output and an unusable object module.

   The error was found in the alternative DDNAME list, passed by module MULTASM when it dynamically invokes the assembler for each source CSECT in the compiler output. The byte count for the replacement DDNAMEs incorrectly included its own two bytes and our assembler treated the two bytes as another eight.

   No source listing of MULTASM is available, so AMASPZAP was used to change the count thus:

```
NAME MULTASM MASM
VER  038E    002A
REP  038E    0028
```

3. The assembler rejected the 'LOAD' option specified in the JCL for the sample problem. Under OS/VS2, an object module is produced by the 'OBJ' or 'OBJECT' option. The sample JCL was corrected.

4. After correction 3, no combined object module was produced.

   This happened because multiple assembler checks for a parameter 'LOAD' or 'NOLOAD' to decide whether to copy object module output to a combined data set (DDNAME SYSGOX).

./2

AMASPZAP was used to modify MULTASM to check for a parameter beginning 'OBJ' or 'NOOBJ', without checking its length:

NAME	MULTASM	MASM
VER	03B8	D3D6C1C4
REP	03B8	D6C2D1C5
VER	03C0	D5D6D3D6C1C4
REP	03C0	D5D6D6C2D1C5
VER	40	47707048D503
REP	40	47000000D502
VER	52	47707066D505
REP	52	47000000D504

This solution is imperfect and re-assembly would be better.

5. The sample program reached execution and ABENDED with code 0C1.

The linkage editor did not allow MAIN as a default entry point.

An ENTRY MAIN control statement was supplied and the sample problem executed correctly. The problem can be avoided in future by a permanent or user-supplied JCL change, but a compiler change would be more to the point.

SUGGESTED IMPROVEMENTS

1. Correct the bugs in RRES$R and the multiple assembler interface MULTASM.

2. Modify the compiler routine END to produce an "END MAIN" statement in the assembler language output for the main routine.

3. Supply a source listing for the multiple assembler interface.

4. Supply the correct distribution tape file names in the write-up. The name changes may have been essential but are buried in a map supplied with the tape.

5. Allow sequencing of Simscript source statements in bytes 73-80. This will make it easier to create and maintain models using TSO EDIT.

SHARE PROGRAM LIBRARY AGENCY

TRIANGLE UNIVERSITIES COMPUTATION CENTER

P.O. Box 12076 • Research Triangle Park, N.C. 27709
Telephone (code 919) 549-8291
919-549-0671

October 19, 1978

TO: Users of The SIMSCRIPT II Programming Language Program

FROM: Pam Durham, SHARE Program Library Agency

The attached user report contains some valuable comments about the use of Program No. 360D-03.2.014, "The SIMSCRIPT II Programming Language."

SHARE PROGRAM LIBRARY AGENCY

USER REPORT

PROGRAM CATALOG NUMBER 360D-03.2.014

COMPUTER MODEL IBM 370/3168

OPERATING SYSTEM MVS 3.7

1. Were you able to implement this program using the instructions supplied? yes

2. Did the program run as documented? Yes, but only after installing fixes suggested by Computing Services Group of Australia. Plus one additional fix.

3. Was the documentation adequate? Yes, especially the fixes suggested by the Australian group.

4. Did you find any errors in the program? We experienced an ØC4 error - see below

5. Do you have any suggestions to improve the distributed material? Yes. When copying the SYSI.SIMPROC data set, the "//GO.SYSIN DD *" card should be included in the procedure as the last card.

6. Comments: See attached.

NAME Bruce C. Graves DATE 8/29/78 INSTALLATION CODE (if any)

ADDRESS Operations Research Dept.
IBM Corporation
D. 812/X585
P.O. Box 12195
Research Triangle Park, NC 27709

PLEASE RETURN TO:

SHARE Program Library Agency
Triangle Universities Computation Center
Post Office Box 12076
Research Triangle Park, NC 27709

The tape did not contain the '//GO.SYSIN DD *' card in the procedure file (file #4), although it is shown on page 46 of the documentation. If the '//GO.SYSIN DD *' card is omitted, the GO step will abend. The //SIMUØ5 DD card refers to DDNAME=SYSIN. If SYSIN is missing, the scheduler will make //SIMUØ5 a dummy. SIMSCRIPT always opens and does a read ahead on unit Ø5, the card reader. If it is dummy without DCB parameters (LRECL, RECFM, BLKSIZE), QSAM will program check. SIMSCRIPT will intercept and close. SIMSCRIPT will program check during its close housekeeping.

Specific errors are: A SIMSCRIPT error 204 (protection exception) followed by an ØC4 in CSECT 'RUSE$R' in routine 'GETO' called by CSECT XCLS, who was closing. The JCL listing has a message flagging the SIMUØ5 DD card.

Bruce C. Graves
Operations Research Department
IBM Corporation
D. 812/X585
P.O. Box 12195
Research Triangle Park, NC 27709