Best Available Copy
MEMORANDUM
RM-5778-PR
OCTOBER 1968

THE SIMSCRIPT II
PROGRAMMING LANGUAGE:
REFERENCE MANUAL
P. J. Kiviat and R. Villanueva

This research is supported by the United States Air Force under Project RAND—Con-
tract No. F46200-67-C-0015—monitored by the Directorate of Operational Requirements
and Development Plans, Deputy Chief of Staff, Research and Development, USAF.
Views or conclusions contained in this study should not be interpreted as representing
the official opinion or policy of the United States Air Force.

DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.
SIMSCRIPT II is described completely in P. J. Kiviat, R. Villanueva, and H. M. Markowitz, THE SIMSCRIPT II PROGRAMMING LANGUAGE, The RAND Corporation, R-460-PR, October 1968. This Memorandum, containing only its syntax and semantics, is designed as a reference manual for programmers already familiar with SIMSCRIPT II.
## CONTENTS

PREFACE AND SUMMARY .......................................................... iii

CONTENTS ................................................................................. v

Section

I. NOTATION ............................................................................. 1

II. BASIC CONSTRUCTS ............................................................. 3
    SYMBOLS ........................................................................... 3
    PRIMITIVES ......................................................................... 4
    METAVARIABLES .................................................................. 5

III. STATEMENTS ......................................................................... 8
    NONEXECUTABLE .................................................................. 8
    STORAGE ALLOCATION .................................................... 14
    COMPUTATION ..................................................................... 15
    CONTROL ............................................................................ 17
    INPUT-OUTPUT .................................................................... 19
    SIMULATION ......................................................................... 22

IV. SYSTEM-DEFINED VALUES ..................................................... 23
    CONSTANTS .......................................................................... 23
    VARIABLES ........................................................................... 23

V. SYSTEM-DEFINED ROUTINES .................................................. 24

VI. GENERATED ATTRIBUTES, VARIABLES AND ROUTINES ........ 25

VII. LIBRARY FUNCTIONS ........................................................... 26
I. NOTATION

The notation employed in describing SIMSCRIPT II is a combination of conventions used in several computer programming language descriptions. The authors chose it for its convenience in their work and in describing the language to others. In the following pages:

1) Words in capital letters are statement keywords.

2) Primitives shown in italics are basic language constructs.

3) A metavariable denotes an occurrence of an element of the type represented by the metavariable symbol shown in italics.

4) A statement is a combination of keywords, primitives, and metavariables that follows a certain pattern, called the syntax of the statement. Section III presents the patterns associated with the statements of SIMSCRIPT II and the meanings associated with them, called the semantics of the statements.

5) Brackets [ ] and braces { } denote choices. When brackets appear, a choice may be made from the options indicated. When braces appear, a choice must be made. The items available for selection appear in a vertical list within the brackets or braces. When a choice can be repeated, a symbol (or symbols) that must separate the items in the list of choices is written at the upper right-hand corner of the brackets or braces. For example, if a choice appears as

\[ A^n \]

the sequence A,A,A,...,A might be selected. The choice represented by \[ A^n \] is logically equivalent to \[-[A][A]...[A] \].

6) The null character \( \_ \) is used to indicate that no symbol
need separate the items in a list of choices. An example of \( \{A\}^n \) might be AAAA...A. The choice represented by \( \{A\}^n \) is logically equivalent to \( A[A][A]...[A] \).

(7) A list separate symbol can itself be complex, involving choices and repetitions, as in
\[
\{ A \mid \text{AND} \mid B \mid \text{OR} \mid C \}
\]

an instance of which might be \( A \text{ AND } B \text{ OR } A \text{ OR } B \).

(8) Plural keywords ending in s such as VARIABUS and LINES, can be written in singular form as VARIABLE or LINE when called for by the grammar of a statement.
II. BASIC CONSTRUCTS

SYMBOLS

\( \text{character} = \begin{cases} \text{letter} \\ \text{special character} \\ \text{digit} \\ \text{blank} \\ \text{period} \end{cases} \)


\( \text{special character} = \{*, %, @, \#\} \)

\( \text{digit} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \)

\( \text{blank} = \) empty space

\( \text{period} = \)
### PRIMITIVES

<table>
<thead>
<tr>
<th>name = (letter)* (digit)</th>
<th>A name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(a) must contain at least one</td>
</tr>
<tr>
<td></td>
<td>letter or two periods</td>
</tr>
<tr>
<td></td>
<td>(b) does not terminate with one</td>
</tr>
<tr>
<td></td>
<td>or more periods</td>
</tr>
<tr>
<td>integer i = (digit)*</td>
<td></td>
</tr>
<tr>
<td>number n = (digit)*</td>
<td>A number contains at most one</td>
</tr>
<tr>
<td></td>
<td>period and at least one digit</td>
</tr>
<tr>
<td>string s = &quot;[character]&quot;*</td>
<td>[character]* cannot contain a &quot;</td>
</tr>
<tr>
<td>text t = [character]*</td>
<td>[character]* cannot contain a</td>
</tr>
</tbody>
</table>
METAVARIABLES

word \( w = \) \{ \}

\begin{align*}
\text{routine} & \quad r = \textit{name}' \\
\text{comma} & \quad c = \{ \textit{AND} \} \\
\text{label} & \quad l = \{ \textit{name} \} \\
\text{variable} & \quad v = \textit{name} \left( \left( \left( s \right)^{1} \right)^{2} \right) \\
\text{arithmetic expression} & \quad e = \left[ \left( \left( + \right)^{1} \right)^{2} \right]
\end{align*}

Words must be separated from each other by one or more blanks unless they are special characters.

Periods (.) are ignored between words and at the end of statements.

Comments can be inserted between any two words in a program by enclosing them in quote marks (""") formed by two consecutive apostrophes. The right-hand set of quotes is not necessary if the comment is the last item on a card.
logical expression =

{ ( )
  ( )
}

AND / OR

{ [ ] [ ]
  [ ] [ ]
}

IS / FALSE

logical expression =

{ ( )
  ( )
}

AND / OR

{ [ ] [ ]
  [ ] [ ]
}

IS / FALSE

relational operator n =

{ ( )
  ( )
}

{ [ ] [ ]
  [ ] [ ]
}

{ [ ] [ ]
  [ ] [ ]
}

{ [ ] [ ]
  [ ] [ ]
}

{ [ ] [ ]
  [ ] [ ]
}

{ [ ] [ ]
  [ ] [ ]
}

{ [ ] [ ]
  [ ] [ ]
}

{ [ ] [ ]
  [ ] [ ]
}

{ [ ] [ ]
  [ ] [ ]
}
termination clause $c =$ \begin{cases} \text{WHILE} & \# \[ . \] \\ \text{UNTIL} & \# \[ . \] \end{cases}

selection clause $sc =$ \begin{cases} \text{WITH} & \{ \text{EXCEPT} \} \text{WHEN} & \# \[ . \] \\ \text{UNLESS} & \# \[ . \] \end{cases}

format $f_1 =$ \begin{cases} B \times \{ \} \\ S \times \{ \} \end{cases} \begin{cases} f \times \{ \} \\ C \times \{ \} \\ + \times \{ \} \\ A \times \{ \} \\ D \times \{ \} \\ E \times \{ \} \end{cases}

for phrase for $=$ \begin{cases} \text{NAME} \text{ BACK FROM } \{ \} \text{ TO } \{ \} \text{ BY } \{ \} \\ \text{FOR} \{ \} \end{cases} \begin{cases} \text{EACH } \{ \} \text{ CALLED NAME} \\ \text{EVERY } \{ \} \text{ FROM } \{ \} \text{ IN } \{ \} \text{ IN REVERSE ORDER} \\ \text{ALL } \{ \} \text{ AFTER } \{ \} \text{ AT } \{ \} \end{cases} \begin{cases} \text{EACH } \{ \} \text{ IN THE DICTIONARY} \end{cases}
III. STATEMENTS

NONEXECUTABLE

(1) PREAMBLE

Marks the beginning of the program preamble.

(2) LAST COLUMN | 15 |

Characters beyond column 15 are ignored on subsequent cards.

(3) normally

Establishes background conditions for properties of variables and functions that are effective unless overridden by subsequent define declarations or, in the case of local arrays, first use.

(4) define

Defines properties of global and local variables, and routines.
(5) \[
\{
\text{TEMPORARY ENTITIES}
\}
\]
\[
\{
\text{PERMANENT ENTITIES} \text{ INCLUDE } \text{name}^f
\}
\]

Declares the type of following EVERY statements.

(6) \[
\{
\text{EVERY } \text{name}^f
\}
\]
\[
\{\text{May, Can, Has, Have, Owns, Belongs To, AN, THE, SOME } \text{name}^f \}
\]
\[
\{\text{name}^f \text{ IN ARRAY, IN WORD, FUNCTION DUMMY}
\}
\]

Entity-attribute-set structure declaration. Specifies attribute packing, equivalence, word assignment and function options.

(7) \[
\text{DEFINE } \text{name}^f \text{ AS } \text{[SET]} \text{ SETS } \{\text{HIGH, LOW, NAME}
\}
\]
\[
\text{RANKED } \{\text{HIGH, LOW, NAME}
\}
\]
\[
\text{WITHOUT } \{\text{name}^f \text{ ATTRIBUTES}
\}
\]
\[
\text{WITHOUT } \{\text{VAR, P, S, U, N, M, N, O, T, I, N, E, S}
\}
\]

Defines set ranking, owner and member attributes and generated set processing routines.

(8) \[
\text{EXTERNAL}
\]
\[
\text{EXOGENOUS}
\]
\[
\text{EVENTS ARE } \text{name}^f
\]

Declares the names of events that can be triggered externally.

(9) \[
\text{EXTERNAL}
\]
\[
\text{EXOGENOUS}
\]
\[
\text{EVENT UNITS ARE } \text{name}^f
\]

Names units from which external event data will be read.

(10) \[
\text{BREAK name TIES BY } \text{HIGH, LOW name}
\]

Establishes a priority order within an event class.
Assigns a priority order to different classes of events.

(12)  
\[
\begin{align*}
\text{ACCUMULATE} \quad & \{ \{ \{ \text{\{HISTOGRAM} \} \} \} \} \\
\text{TALLY} \quad & \{ \{ \{ \text{\{HISTOGRAM} \} \} \} \} \\
\end{align*}
\]

Specifies automatic data collection and analysis for named variables.

(13)  
\[
\begin{align*}
\{ \{ \{ \text{\{CREATING \}} \} \} \} \\
\{ \{ \{ \text{\{DESTROYING \}} \} \} \} \\
\{ \{ \{ \text{\{FILLING \}} \} \} \} \\
\{ \{ \{ \text{\{REMOVING \}} \} \} \} \\
\{ \{ \{ \text{\{CAUSING \}} \} \} \} \\
\{ \{ \{ \text{\{CANCELING \}} \} \} \} \\
\end{align*}
\]

Specifies a call to a named routine whenever the indicated statement is executed. Inputs to the routines are:

<table>
<thead>
<tr>
<th>BEFORE</th>
<th>AFTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>Not allowed</td>
</tr>
<tr>
<td>DESTROY</td>
<td>Entity identifier</td>
</tr>
<tr>
<td>CAUSE</td>
<td>Entity identifier, time</td>
</tr>
<tr>
<td>CANCEL</td>
<td>Entity identifier</td>
</tr>
<tr>
<td>FILE</td>
<td>Entity identifier, subscripts</td>
</tr>
<tr>
<td>REMOVE</td>
<td>Entity identifier, subscripts</td>
</tr>
</tbody>
</table>
(14) **DEFINE** \( w \) **TO MEAN** \( w' \)

Instructs the compiler to substitute the words (up to the end of the card on which the statement appears) following the keyword **MEAN** for the indicated word in all subsequent statements, before they are compiled.

(15) **SUBSTITUTE**

\[
\begin{align*}
\text{THIS} \\
\text{THESE} \\
\text{LINES FOR } w
\end{align*}
\]

Similar to (14) but allows more than one card of words to be substituted.

(16) **SUBSTITUTE**

\[
\begin{align*}
\text{RESUME} \\
\text{SUBSTITUTION}
\end{align*}
\]

Used to override currently defined substitutions. These statements must not be placed on program cards with other statements.

(17) **END**

Marks the end of a program preamble, routine, report section, and heading block of a report section.

(18) **MAIN**

Marks the beginning of a program's main routine. Execution commences at the first executable statement after **MAIN**.

(19) **LEFT** **TO** **RIGHT** **ROUTINE** **FOR**

\[
\begin{align*}
\text{THE} \\
\text{GIVING} \\
\text{YIELDING} \\
\text{NAME}
\end{align*}
\]

Subprogram declaration. Routines used as functions only have given arguments. If **LEFT** or **RIGHT** are not stated, **RIGHT** is implied.
Event declaration. Unless saved, an event notice is destroyed before an event routine is executed.

Marks the beginning of a report section.

Marks the beginning of a heading block within a report section.
# A PREAMBLE STATEMENT RECAP

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Statement</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>Normally</td>
<td>Can appear anywhere in preamble.</td>
</tr>
<tr>
<td>1b</td>
<td>Define to mean</td>
<td></td>
</tr>
<tr>
<td>1c</td>
<td>Substitute</td>
<td></td>
</tr>
<tr>
<td>1d</td>
<td>Suppress subst</td>
<td></td>
</tr>
<tr>
<td>1e</td>
<td>Resume subst</td>
<td></td>
</tr>
<tr>
<td>2a</td>
<td>Temporary entities</td>
<td>A preamble may contain many Type 2a, 2b, and 2c statements. Each may be followed by a group of Type 3a, 4, and 5 statements.</td>
</tr>
<tr>
<td>2b</td>
<td>Permanent entities</td>
<td></td>
</tr>
<tr>
<td>2c</td>
<td>Event notices</td>
<td></td>
</tr>
<tr>
<td>3a</td>
<td>Every</td>
<td>Many can follow a Type 2 statement. An entity or event notice name can appear in more than one EVERY statement.</td>
</tr>
<tr>
<td>3b</td>
<td>The system</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Define variable</td>
<td>No precedence relation if it defines a global variable. Must follow all Type 3a statements if it defines an attribute named in them. A variable, attribute, or function name can appear in only one DEFINE statement.</td>
</tr>
<tr>
<td>5</td>
<td>Define set</td>
<td>Must follow Type 4 statements in a Type 2 statement group if it qualifies a set named in them.</td>
</tr>
<tr>
<td>6a</td>
<td>Break ties</td>
<td>One statement allowed for each event notice.</td>
</tr>
<tr>
<td>6b</td>
<td>External events</td>
<td></td>
</tr>
<tr>
<td>6c</td>
<td>External units</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Priority</td>
<td>Must follow all Type 2c and 6b statements.</td>
</tr>
<tr>
<td>8a</td>
<td>Before</td>
<td>Allowed for each temporary entity, set, and event notice.</td>
</tr>
<tr>
<td>8b</td>
<td>After</td>
<td></td>
</tr>
<tr>
<td>9a</td>
<td>Tally</td>
<td>One statement allowed for each global variable or attribute.</td>
</tr>
<tr>
<td>9b</td>
<td>Accumulate</td>
<td></td>
</tr>
</tbody>
</table>

Of these statements, only Types 1 and 4 can be used in routines to declare local background conditions, variables, and substitutions.
STORAGE ALLOCATION

(1) \[ \text{RESERVE} \left\{ \left[ x \right] \text{ AS } \left[ a \right] \text{ BY } \left[ b \right] \right\} \]

Allocates blocks of core of specified size to the pointer variables \( a \). Words assigned are data if no \( \text{as} \) phrase appears, and are pointers otherwise.

(2) \[ \text{RELEASE} \left[ \left[ x \right] \right] \]

Releases blocks of core pointed to by \( x \); \( x \)'s are assumed to be pointer variables.

(3) \[ \begin{align*}
\text{CREATE} & \left\{ \left[ A \right] \text{ AS } \left[ \text{CALLED } x \right] \right\} \\
& \left\{ \text{EACH} \left[ A \right] \text{ AS } \left[ \text{CALLED } x \right] \right\} \\
& \left\{ \text{ALL} \left[ A \right] \text{ AS } \left[ \text{CALLED } x \right] \right\}
\end{align*} \]

Obtains a block of words from the "free-storage" area.

(4) \[ \text{DESTROY} \left\{ \left[ \text{THE } \right] \text{ AS } \left[ \text{CALLED } x \right] \right\} \]

Returns a specified block of words to the "free-storage" area.

(5) \[ \text{ERAS} \left[ \left[ x \right] \right] \]

Removes text variables from the dictionary and returns their space to the "free-storage" area.
COMPUTATION

(1) \textit{LET } t \\ Assigns the value of \( t \) to the variable \( v \).

(2) \textit{ADD } \textit{TO} \( v \\ Adds the value of \( v \) to the value of the variable \( v \).

(3) \textit{SUBTRACT } \textit{FROM} \( v \\ Subtracts the value of \( v \) from the value of the variable \( v \).

(4) 
\[
\text{COMPUTE } \{ \begin{align*}
&\text{AVERAGE} \\
&\text{AVG} \\
&\text{MEAN} \\
&\text{SUM} \\
&\text{NUMBER} \\
&\text{NUM} \\
&\text{VARIANCE} \\
&\text{VAR} \\
&\text{STD. DEV} \\
&\text{STD} \\
&\text{SUM OF SQUARES} \\
&\text{SSQ} \\
&\text{MEAN SQUARE} \\
&\text{MSQ} \\
&\text{MINIMUM}(e) \\
&\text{MIN}(e) \\
&\text{MAXIMUM}(e) \\
&\text{MAX}(e) \\
&\text{MINIMUM} \\
&\text{MIN} \\
&\text{MAXIMUM} \\
&\text{MAX}
\end{align*} \} \text{ OF } \( e \)
\]

Must be controlled by a logical control phrase. Computes the indicated statistics of the expression \( e \) after the \textit{LOOP} statement if the control is over a \textit{DO...LOOP} block.
(5) \[
\text{FIND } \left\{ \text{FIRST CASE} \right\} \left[ \begin{array}{c}
\text{IF} \ \\text{FOUND} \\
\text{NONE}
\end{array} \right]
\]

Must be controlled by a logical control phrase, but cannot be within a do...loop block. The optional IF phrase directs control after the control phrase has been completed, depending upon the "success" of the FIND.

(6) \[
\text{FILE} \left[ \begin{array}{c}
\text{FIRST} \\
\text{LAST}
\end{array} \right] \left[ \begin{array}{c}
\text{IN} \\
\text{THIS}
\end{array} \right]
\]

Files an entity in a set.

(7) \[
\text{REMOVE} \left[ \begin{array}{c}
\text{FIRST} \\
\text{LAST} \\
\text{THIS}
\end{array} \right] \left[ \begin{array}{c}
\text{FROM} \\
\text{THIS}
\end{array} \right]
\]

Removes an entity from a set.

(8) \[
\text{MOVE} \left[ \begin{array}{c}
\text{FROM} \\
\text{TO}
\end{array} \right]
\]

Used within a routine defined for a monitored variable to access or set the value of the variable.

(9) \[
\text{ENTER} \ \text{WITH} \ \text{a}
\]

Used to transfer a "right-hand" value to a left-handed function.

(10) \[
\text{STORE} \ \text{a} \ \text{IN} \ \text{b}
\]

Assigns a value to a variable without mode conversion.

(11) \[
\text{RESET} \ \text{TOTALS OF} \ \text{a}
\]

Initializes an accumulate or tally counters associated with \( a \). If \( \text{TOTALS} \) is not qualified by a word, all counters of \( a \) are initialized.
CONTROL

(1)  
\[ l \{ l \} \]

A statement label identifies a transfer point.

(2)  
\[ \text{GO [TO]} \{ l \{ l \} \} \]

Transfers control to the indicated label.

(3)  
\[ \text{GO [TO]} \{ l \{ l \} \} \]

Transfers control to the nth label in the label list according to the integer value of the transfer expression e.

(4)  
\[ \text{IF} \{ e \{ . \} \] 

If the logical expression e is true, continues execution with the next statement. If e is false, transfers to the following ELSE statement. When nested IF statements appear, the word ELSE can be used to indicate that they have a common ELSE statement.

(5)  
\[ \text{ELSE} \]
\[ \text{ALWAYS} \]
\[ \text{OTHERWISE} \]
\[ \text{REGARDLESS} \]

Synonyms indicating the transfer point of the false condition of a preceding IF statement.

(6)  
\[ \text{ALSO} \{ \text{IF} \{ l \{ l \} \} \text{DO THIS} \text{ THE FOLLOWING} \]

Logical phrases control the execution of statements that follow them. When more than one statement is to be controlled, the
word DO precedes them. Multiple control phrases terminating control on the same LOOP statement are preceded by the word ALSO.

(7) \[
\begin{align*}
\text{LOOP} & \quad \text{REPEAT} \\
& \\
\end{align*}
\]

Used with DO to delimit a group of statements controlled by one or more logical control phrases.

(8) \[
\begin{align*}
\text{PERFORM} & \quad \text{CALL} & \quad \text{EXECUTE} \quad \text{GIVING} \\
\text{NOW} & \quad \text{RETURN} \\
\end{align*}
\]

Calls a routine used as a procedure. Both input GIVEN and output YIELDING argument lists are optional.

(9) \[
\begin{align*}
\text{RETURN} \\
\end{align*}
\]

Used as a procedure, a routine returns control to its calling program with the statement RETURN; used as a function, a routine returns control and a value to its calling program by either of the statements RETURN; or RETURN WITH.

(10) STOP

Halts program execution.
The lines following the `PRINT` statement are format lines containing text and pictorial formats for the display of indicated expression values. The phrases `A GROUP OF` and `SUPPRESSING FROM COLUMN` can only be used within report sections that have column repetition.

A free-form output statement that labels and displays values of expressions, and 1- and 2-dimensional arrays.

Sets the indicated input/output device as the current input or output unit. All subsequent input/output statements that do not specify their own devices in `USING` phrases use these current units. `THE BUFFER` causes reading and writing in an internal file.

Used without an `AS` clause indicates a free-form data input.
WRITE 
\{ \( x \) \} \{ \text{BINARY} \} \{ \text{AS} [ ( x ) ( y ) ] \} \{ \text{USING} [ \text{TAPES} ] \} \{ \text{TAPES} \} \{ \text{UNIT} \} \{ \text{TAPE} \} \{ \text{UNIT} \} \\
AS \{ ( x ) ( y ) \} \{ \text{TAPE} \} \{ \text{UNIT} \} \{ \text{TAPES} \} \{ \text{UNIT} \} \\

Writes formatted output only.

REWIND \{ \text{TAPE} \} \{ \text{UNIT} \} \\
Rewinds an input/output device.

ADVANCE \{ \text{INPUT} \} \{ \text{FILES} \} \{ \text{USING} [ \text{TAPES} ] \} \{ \text{TAPES} \} \{ \text{UNIT} \} \\
BACKSPACE \{ \text{OUTPUT} \} \{ \text{FILES} \} \{ \text{USING} [ \text{TAPES} ] \} \{ \text{TAPES} \} \{ \text{UNIT} \} \\

Performs the indicated operations.

CLOSE \{ \text{TAPES} \} \{ \text{TAPES} \} \{ \text{UNIT} \} \\
Writes an end-of-file mark on an output device.

SKIP \{ \text{FIELDS} \} \{ \text{INPUT} \} \{ \text{CARDS} \} \{ \text{LINES} \} \{ \text{OUTPUT} \} \{ \text{RECORDS} \} \\

Applies to the current input or current output unit. SKIP * FIELDS applies to the current input unit only when it is used for free-form data input. CARDS, LINES, and RECORDS are synonyms. If neither INPUT nor OUTPUT is specified, INPUT is implied.
Applies to the current input or current output unit. LINE, CARD, and RECORD are synonyms. If neither INPUT nor OUTPUT is specified, INPUT is implied.

Reads data as successive characters in a TEXT variable until the character contained in MARK.V is encountered.

Writes TEXT variables, starting at the current output column.

Produces a backtrack of current subprogram calls. When the SIMSCRIPT II operating system uses TRACE the standard output device (printer) is used.

Used either with program overlay or dynamic program relocation to load an indicated routine from a system load unit.

Used only with dynamic program relocation. Saves an indicated routine on a system save unit.
(1) START SIMULATION

Starts simulation by removing the first event from the events set and executing it.

(2) 

\[
\text{SCHEDULE} \quad \left( \text{[\text{AN}] [\text{THE [ABOVE]}]} \right) \quad = \quad \left( \text{[CALLED \text{*}]} \right) \quad \left( \text{[GIVEN \text{\{\text{\text{*}}}]} \right) \quad \left( \text{[AT \text{\{\text{\text{*}}}]} \right) \quad \left( \text{[IN \text{\{\text{\text{*}}}]} \right) \quad \left( \text{[AFTER \text{\{\text{\text{*}}}]} \right) \quad \left( \text{[DAYS \text{\{\text{\text{*}}}]} \right) \quad \left( \text{[HOURS \text{\{\text{\text{*}}}]} \right) \quad \left( \text{[MINUTES \text{\{\text{\text{*}}}]} \right)
\]

Fills an event notice in the events set according to its time.

(3) 

\[
\text{CANCEL} \quad \left( \text{[\text{THE [ABOVE]}]} \right) \quad = \quad \left( \text{[CALLED \text{*}]} \right)
\]

Removes a scheduled event notice from the event set.
### IV. SYSTEM-DEFINED VALUES

#### CONSTANTS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP.C</td>
<td>Largest INTEGER value that the computer can store</td>
</tr>
<tr>
<td>IM.C</td>
<td>π</td>
</tr>
<tr>
<td>P1.C</td>
<td>57.29577 degrees/radian</td>
</tr>
<tr>
<td>R1M.C</td>
<td>Largest REAL value that the computer can store</td>
</tr>
</tbody>
</table>

#### VARIABLES

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BETWEEN.V</td>
<td>SUBPROGRAM variable called before events are executed</td>
<td>0</td>
</tr>
<tr>
<td>BUFFER.V</td>
<td>Length of the BUFFER</td>
<td>Implementation</td>
</tr>
<tr>
<td>EOF.V</td>
<td>End-of-file action code</td>
<td>0</td>
</tr>
<tr>
<td>EVENT.V</td>
<td>Code of the current event</td>
<td>0</td>
</tr>
<tr>
<td>EVENTS.V</td>
<td>Number of event classes</td>
<td>0</td>
</tr>
<tr>
<td>F.I.S.</td>
<td>Array containing first-in-set pointers for set EV.S</td>
<td>0</td>
</tr>
<tr>
<td>HOURS.V</td>
<td>Number of hours per simulated day</td>
<td>24</td>
</tr>
<tr>
<td>LINE.V</td>
<td>Number of current output line</td>
<td>0</td>
</tr>
<tr>
<td>LINES.V</td>
<td>Number of lines allowed per page</td>
<td>55</td>
</tr>
<tr>
<td>MARK.V</td>
<td>TEXT, external event data and RANDOM variable read termination character</td>
<td>** ***</td>
</tr>
<tr>
<td>MINUTES.V</td>
<td>Number of minutes per simulated hour</td>
<td>60</td>
</tr>
<tr>
<td>PAGE.V</td>
<td>Number of current page</td>
<td>1</td>
</tr>
<tr>
<td>RCOLUMN.V</td>
<td>Location in buffer of current read pointer</td>
<td>0</td>
</tr>
<tr>
<td>READ.V</td>
<td>Number of current read unit</td>
<td>Implementation</td>
</tr>
<tr>
<td>SEED.V</td>
<td>Array containing initial random numbers</td>
<td>Implementation</td>
</tr>
<tr>
<td>TIME.V</td>
<td>Current simulation time</td>
<td>0</td>
</tr>
<tr>
<td>WCOLUMN.V</td>
<td>Location in buffer of current write pointer</td>
<td>0</td>
</tr>
<tr>
<td>WRITE.V</td>
<td>Number of current write unit</td>
<td>Implementation</td>
</tr>
</tbody>
</table>
V. SYSTEM-DEFINED ROUTINES

ORIGIN (s, e, r) Establishes a simulation-time origin

s, e, r, INTEGER
## VI, GENERATED ATTRIBUTES, VARIABLES AND ROUTINES

<table>
<thead>
<tr>
<th>SETS</th>
<th>P.ave</th>
<th>S.ave</th>
<th>M.ave</th>
<th>F.ave</th>
<th>L.ave</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generated attributes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generated routines</td>
<td>A.ave</td>
<td>File first or ranked</td>
<td>B.ave</td>
<td>File last</td>
<td>C.ave</td>
<td>File before</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ENTITIES</th>
<th>N.entity</th>
<th>M.entity</th>
<th>L.entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generated variables</td>
<td>N.entity</td>
<td>M.entity</td>
<td>L.entity</td>
</tr>
<tr>
<td>Generated routine</td>
<td>C.entity</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENT NOTICES</th>
<th>I.entity</th>
<th>L.entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generated variables</td>
<td>I.entity</td>
<td>L.entity</td>
</tr>
<tr>
<td>Generated routines</td>
<td>C.entity</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STANDARD ENTITIES</th>
<th>RANDOM I</th>
<th>EVENT NOTICE</th>
<th>TEXT VARIABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generated attributes</td>
<td>PROB A</td>
<td>TIME A</td>
<td>LENGTH A</td>
</tr>
<tr>
<td></td>
<td>VALUE A</td>
<td>ENTITY A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RVALUE A</td>
<td>M. EV S</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>S. EV S</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACCUMULATE VARIABLES</th>
<th>Generated routine</th>
<th>R. variable</th>
</tr>
</thead>
</table>
### VII. LIBRARY FUNCTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Operation Description</th>
<th>Mode of e</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS.F</td>
<td>e</td>
<td>Computes the absolute value of e</td>
<td>REAL</td>
<td>- SIGF and REAL</td>
</tr>
<tr>
<td>ARCCOS.F</td>
<td>e</td>
<td>Arccosine of e</td>
<td>REAL</td>
<td>- SIGF and REAL</td>
</tr>
<tr>
<td>ARCSIN.F</td>
<td>e</td>
<td>Arcsine of e</td>
<td>REAL</td>
<td>(0, 1) ≤ e and REAL</td>
</tr>
<tr>
<td>ARCTAN.F</td>
<td>e, e</td>
<td>Arctangent of e and e/e</td>
<td>REAL</td>
<td>(0, 1) ≤ e and REAL</td>
</tr>
<tr>
<td>ATOT.F</td>
<td>v</td>
<td>Converts the characters of the ALPHA variable v to TEXT</td>
<td>TEXT</td>
<td>v is a variable and ALPHA</td>
</tr>
<tr>
<td>BINOINTIAL.F</td>
<td>e, e</td>
<td>Random sample from a binomial distribution with number of trials = e, probability of success = e, using random number stream e</td>
<td>INTEGER</td>
<td>e, e, INTEGER, e, REAL</td>
</tr>
<tr>
<td>CONCAT.F</td>
<td>v, v</td>
<td>Concatenation of v and v</td>
<td>TEXT</td>
<td>v, v, TEXT</td>
</tr>
<tr>
<td>COS.F</td>
<td>e</td>
<td>Cosine of e</td>
<td>REAL</td>
<td>e in radian and REAL</td>
</tr>
<tr>
<td>DATE.F</td>
<td>e, e, e</td>
<td>Converts month, day, year to cumulative time</td>
<td>REAL</td>
<td>e, e, e, e, INTEGER</td>
</tr>
<tr>
<td>DAY.F</td>
<td>e</td>
<td>“Day part” of time expression e</td>
<td>INTEGER</td>
<td>e is REAL</td>
</tr>
<tr>
<td>DIM.F</td>
<td>e</td>
<td>Number of elements pointed to by e</td>
<td>INTEGER</td>
<td>v is a pointer variable</td>
</tr>
<tr>
<td>DIV.F</td>
<td>e, e</td>
<td>TRUNC.(e/e)</td>
<td>INTEGER</td>
<td>e, e, INTEGER, e, e, e, 0</td>
</tr>
<tr>
<td>FIELD.F</td>
<td></td>
<td>Ending column of next field to be read in free-form data input</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>ERLANG.F</td>
<td>e, e</td>
<td>Random sample from Erlang distribution with mean e, k = e, using random number stream e</td>
<td>REAL</td>
<td>e, REAL, e, k, INTEGER</td>
</tr>
<tr>
<td>EXP.F</td>
<td>e</td>
<td>Exponential e</td>
<td>REAL</td>
<td>e is REAL</td>
</tr>
<tr>
<td>EXPONENTIAL.F</td>
<td>e, e</td>
<td>Random sample from exponential distribution with mean = 1/e, using random number stream e</td>
<td>REAL</td>
<td>e, e, REAL, e, INTEGER</td>
</tr>
<tr>
<td>FRAC.F</td>
<td>e</td>
<td>- TRUNC.F(e/e)</td>
<td>REAL</td>
<td>e is REAL</td>
</tr>
<tr>
<td>GAMMA.F</td>
<td>e, e, e</td>
<td>Random sample from a Gamma distribution with mean e, k = e, using random number stream e</td>
<td>REAL</td>
<td>e, e, e, REAL, e, INTEGER</td>
</tr>
<tr>
<td>HOUR.F</td>
<td>e</td>
<td>“Hour part” of time expression e</td>
<td>INTEGER</td>
<td>e is REAL</td>
</tr>
<tr>
<td>INT.F</td>
<td>e</td>
<td>e rounded to an integer</td>
<td>INTEGER</td>
<td>e is REAL</td>
</tr>
<tr>
<td>ISTEP.F</td>
<td>e, e</td>
<td>Random sample from a look-up table pointed to by e using random number stream e</td>
<td>INTEGER</td>
<td>e is a pointer variable, e is INTEGER</td>
</tr>
<tr>
<td>ITOA.F</td>
<td>e</td>
<td>Converts the integer expression e to ALPHA</td>
<td>ALPHA</td>
<td>e is INTEGER</td>
</tr>
<tr>
<td>LIN.F</td>
<td>e, e</td>
<td>Random sample from a look-up table pointed to by e using random number stream e applying linear interpolation</td>
<td>REAL</td>
<td>e is a pointer variable, e is INTEGER</td>
</tr>
<tr>
<td>LOG.E F</td>
<td>e</td>
<td>Computes the logarithm of e</td>
<td>REAL</td>
<td>e ≥ 0 and REAL</td>
</tr>
<tr>
<td>LOG.NORMAL.F</td>
<td>e, e, e</td>
<td>Random sample from a lognormal distribution with mean = e, standard deviation = e, using random number stream e</td>
<td>REAL</td>
<td>e, e, REAL, e, INTEGER</td>
</tr>
<tr>
<td>LOG.10.F</td>
<td>e</td>
<td>Computes the base 10 logarithm of e</td>
<td>REAL</td>
<td>e ≥ 0 and REAL</td>
</tr>
</tbody>
</table>
### VII. LIBRARY FUNCTIONS (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Operation Description</th>
<th>Type</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX</td>
<td>$a_1, \ldots, a_n$</td>
<td>Value of largest $a_i$</td>
<td>INTEGER of all $a_i$, INTEGER, otherwise REAL</td>
<td>None</td>
</tr>
<tr>
<td>MIN</td>
<td>$a_1, \ldots, a_n$</td>
<td>Value of smallest $a_i$</td>
<td>INTEGER of all $a_i$, INTEGER, otherwise REAL</td>
<td>None</td>
</tr>
<tr>
<td>FMIN</td>
<td>$a_1, \ldots, a_n$</td>
<td>minute part of time expression $a$</td>
<td>INTEGER $a_1, a_2, \ldots, a_n$</td>
<td>REAL</td>
</tr>
<tr>
<td>MOD</td>
<td>$a_1, a_2$</td>
<td>$a \mod a_2$</td>
<td>INTEGER of $a$, and $a_2$, INTEGER, otherwise REAL</td>
<td>$a_2 \neq 0$</td>
</tr>
<tr>
<td>MONTH</td>
<td>$a_1$</td>
<td>month part of time expression $a$</td>
<td>INTEGER $a_1$</td>
<td>REAL</td>
</tr>
<tr>
<td>NORMAL</td>
<td>$a_1, a_2, a_3$</td>
<td>Random sample from a normal distribution with mean $a_1$, standard deviation $a_2$, using random number stream $a_3$</td>
<td>REAL $a_1$, REAL $a_2$, INTEGER $a_3$</td>
<td>None</td>
</tr>
<tr>
<td>OUT</td>
<td>$a_1$</td>
<td>ALPHA value of $a_1$ character in the current output buffer</td>
<td>ALPHA $a_1$</td>
<td>$a_1$ and INTEGER</td>
</tr>
<tr>
<td>POISSON</td>
<td>$a_1, a_2$</td>
<td>Random sample from a Poisson distribution with mean $a_1$, using random number stream $a_2$</td>
<td>INTEGER $a_1$, REAL $a_2$, INTEGER $a_3$</td>
<td>None</td>
</tr>
<tr>
<td>RANDOM</td>
<td>$a_1$</td>
<td>Random number on interval $(0,1)$ using random number stream $a_1$</td>
<td>REAL $a_1$, INTEGER $a_2$</td>
<td>None</td>
</tr>
<tr>
<td>REAL</td>
<td>$a_1$</td>
<td>Expressed as a REAL number</td>
<td>REAL $a_1$</td>
<td>INTEGER $a_2$</td>
</tr>
<tr>
<td>RSTEP</td>
<td>$a_1, a_2$</td>
<td>Random sample from look-up table pointed to by $a_1$, using random number stream $a_2$</td>
<td>REAL $a_1$, INTEGER $a_2$</td>
<td>None</td>
</tr>
<tr>
<td>SFIELD</td>
<td>$a_1$</td>
<td>Starting column of next field to be read in free-form data input</td>
<td>INTEGER $a_1$</td>
<td>INTEGER $a_2$</td>
</tr>
<tr>
<td>SIGN</td>
<td>$a_1$</td>
<td>$1$ if $a_1 &gt; 0$, $0$ if $a_1 = 0$, $-1$ if $a_1 &lt; 0$</td>
<td>INTEGER $a_1$</td>
<td>REAL $a_2$</td>
</tr>
<tr>
<td>SIN</td>
<td>$a_1$</td>
<td>sin $a_1$</td>
<td>REAL $a_1$</td>
<td>in radians and REAL $a_2$</td>
</tr>
<tr>
<td>COS</td>
<td>$a_1$</td>
<td>cos $a_1$</td>
<td>REAL $a_1$</td>
<td>in radians and REAL $a_2$</td>
</tr>
<tr>
<td>TAN</td>
<td>$a_1$</td>
<td>tan $a_1$</td>
<td>REAL $a_1$</td>
<td>in radians and REAL $a_2$</td>
</tr>
<tr>
<td>TRUNC</td>
<td>$a_1$</td>
<td>$\lfloor a_1 \rfloor$</td>
<td>INTEGER $a_1$</td>
<td>REAL $a_2$</td>
</tr>
<tr>
<td>TTOA</td>
<td>$a_1$</td>
<td>Converts the initial characters of the TEXT string $a_1$ to ALPHA</td>
<td>ALPHA $a_1$</td>
<td>INTEGER $a_2$</td>
</tr>
<tr>
<td>UNIFORM</td>
<td>$a_1, a_2, a_3$</td>
<td>Random sample from a uniform distribution over interval $(a_1, a_2)$ using random number stream $a_3$</td>
<td>REAL $a_1$, REAL $a_2$, INTEGER $a_3$</td>
<td>None</td>
</tr>
<tr>
<td>WEEKDAY</td>
<td>$a_1$</td>
<td>day of the week of time expression $a_1$</td>
<td>INTEGER $a_1$</td>
<td>REAL $a_2$</td>
</tr>
<tr>
<td>WEIBULL</td>
<td>$a_1, a_2, a_3$</td>
<td>Random sample from a Weibull distribution with scale parameter $a_1$, and shape parameter $a_2$, using random number stream $a_3$</td>
<td>REAL $a_1$, REAL $a_2$, INTEGER $a_3$</td>
<td>None</td>
</tr>
<tr>
<td>YEAR</td>
<td>$a_1$</td>
<td>year part of time expression $a_1$</td>
<td>INTEGER $a_1$</td>
<td>REAL $a_2$</td>
</tr>
</tbody>
</table>
A compact reference listing of the syntax and semantics of SIMSCRIPT II, designed for professional programmers already familiar with the language. (SIMSCRIPT II is fully described in R-460-PR, and its IBM 360 implementation in RM-5777-PR.) The notation employed was chosen for convenience and descriptive power from conventions previously used in computer programming language descriptions. The study describes notation; basic constructs (symbols, primitives, metavariables); statements (non-executable, storage allocation, computation, control, input-output, simulation); system-defined values (constants, variables); a system-defined routine (the ORIGIN routine for simulation time); generated attributes, variables, and routines; and library functions.