4 Implementation defined parts of the SIMULA language.

The references in each sections refer to (12).

4.1 Use of system prefixes.

Refer section 2.2.1.

The system prefixes SIMSET and SIMULATION must not be used at more than one block level at a time in a program (including external procedures and classes). When a SIMSET or SIMULATION block is left through the final end, another one can be declared at any block level. Several incarnations of a SIMSET or SIMULATION block can exist at the same time, as long as they have the same block level.

SIMSET and SIMULATION can be used for block or class prefixing.

LINK, HEAD and PROCESS can be used for class prefixing only.

FILE subclasses must not be used as prefixes.

4.2 Accepted virtual specifiers and rules for virtual matches.

Refer section 2.2.3.

The accepted virtual specifiers are the same as the legal procedure parameter specifiers. For all virtual quantities any match must have the same type as the specification, except for the case of a virtual REF procedure, where a subordinate type is accepted (cf. (12), 3.2.5), and a <notype> procedure, which can be matched by a <type> procedure. In the latter case the type is accessible only at access levels deeper than or equal to that of the match, and any match in a subclass must have the same type, or a type subordinate to that of the latest match.

4.3 Collating sequence.

Refer section 3.2.2.1.

The collating sequence is defined by the 8-bit internal EBCDIC character representation, which also defines the integer-character correspondence established by the standard procedures RANK and CHAR (4).

4.4 Initialisation of character variables.

Refer section 3.2.4.

Character variables are initialised to internal zero bytes. These have no printable equivalent, but usually show up as blanks.

4.5 Evaluation of Boolean expressions.

Refer to section 4.2.2.1.

In SIMULA for 360/370 all function designators occurring in a Boolean expression will be evaluated from left to right. Some other systems, however, do not follow this rule. They operate such that evaluation proceeds from left to right only until the resulting value of the Boolean expression can be calculated. For this reason you are advised not to use side-effects of function designators occurring in Boolean expressions, unless you have checked that execution is independent of the evaluation method. Otherwise this would destroy the portability of the program.

4.6 For statements.

Refer section 6.2.

The controlled variable of a for statement must not be a remote or subscripted variable, a formal parameter called by name, or a procedure identifier. Note that the value of a controlled variable is welldefined when the for statement is left.

4.7 Parameter type correspondence.

Refer to section 8.2.

The type correspondence rules for parameters are given in the table 4.1.

*. mode / *. formal parameter *.	name	reference	value
<value type>	C	-	C
<reference type>	C	C	C
<value type> ARRAY	I	I	I
<ref type> ARRAY	C	I	-
<type> PROCEDURE	S	S	-

Table 4.1: Actual/formal type correspondence

C: types must be compatible.

I: types must coincide.

S: actual type must be subordinate to formal type.

4.8 Separator

Refer to section 10.8.2.

On output (PUTFRAC, OUTFRAC) a <separator> is a blank character.

On input (GETFRAC, INFRAC) a blank or a comma is accepted.

A separator is a character separating the digit groups in a <GROUP>.

4.9 Power-of-ten symbol.

The basic symbol is used by the standard procedures PUTREAL, OUTREAL, GETREAL and INREAL. Initially is represented by E (as in Fortran), but it can be altered dynamically by means of the standard procedure LOWTEN, which has one character parameter. After a call on LOWTEN, the character passed as parameter will act as the power-of-ten symbol in place of E. This character should not be a digit, a sign, a blank or a period.

4.10 Edit overflow.

Refer section 10.10

If an edit overflow occurs, the text into which the number could not be edited is filled with asterisks.

In addition a warning message is printed at the end of program execution and 4 is added to the return code.

4.11 Formats of editing procedures.

Refer section 10.10.

PUTINT: The number is edited with initial zero suppresion. If it is negative, a minus sign is edited immediately before the first significant digit. Zero is edited as the single digit 0, right adjusted in the text.

PUTFRAC: The number is edited in three-digit groups, starting outwards from the decimal point. If the number is zero, the digits after the decimal point are zero. For a negative number, a minus sign is edited immediately before the decimal point or the first significant digit, whichever is first.

No more than 12 digits may appear after the decimal point.

PUTREAL: The number is edited according to the picture:

sd.dddEzdd or sEzdd

where s is blank or -
d is a digit
z is + or -

The number of digits after the decimal point depends on the precision requested.

Zero is edited as a single 0, right adjusted in the text.

4.12 Mathematical subroutines.

The following elementary functions are available as standard functions:

name	function
ARCCOS	(cos X)**-1
ARCSIN	(sin X)**-1
ARCTAN	(tan X)**-1
COS	cos X
COSH	coshyp X (1/2(e**x + e**(-x)))
EXP	e**x
LN	ln X
SIN	sin X
SINH	sinhyp X (1/2(e**x - e**(-x)))
SQRT	SQRT(X)
TAN	tan X
TANH	tanhyp X ((e**x - e**(-x))/(e**x + e**(-x)))

The approximation methods used to compute these standard functions are described in (see (14)).

The function value is computed in double precision if the argument is of type LONG REAL.

The algol-defined functions ABS, ENTIER and SIGN are also available.

4.13 Array subscript checking.

The subscripts of an array are not checked individually, but a check is made to see that the address of a subscripted variable lies within the array.

4.14 External procedures.

Any REF type parameter of an external procedure must be qualified by a subclass of FILE.

An external <type> procedure may have any type except REF.

The name of an external procedure may not start with the prefix ZYQ, and seven characters are significant.

When a program includes a declaration of an external procedure, an entry in the external symbol dictionary is generated (ESD). When the procedure is used in the program, this ESD entry will be used as the basis for the information passed on to the loader. If the procedure is not referenced, the information will not be passed to the loader, i.e. only those procedures that are actually used will be included by the loader.

A user can indicate that an assembly procedure does not require the standard interface in order to work correctly. In such case the routine is linked in directly which results in an increased efficiency. In particular note that:

- One must use the word FAST after external e.g.

  ```
  external fast long real procedure cputime;
  ```

- Such procedure cannot have parameters (so far) but, if the procedure name happens to be CODE (Cf. External Procedure Library, NCC Publication S56) then the compiler will plant its literal parameters as in-line code and no call will be generated at all.

- A given procedure can only be linked in one way in a program.

4.15 External classes.

Any REF type parameter of an external class must be qualified by a subclass of FILE or by the external class itself.

The name of an external class may not start with the prefix ZYQ, and seven characters are significant.

To compile a class separately, use the procedure SIMC as follows:

```
//XEC  EXEC  SIMC,PARM='EXTERN=C'
//SYSPUNCH DD DSN=library-name(Mname),DISP=OLD,DCB=BUFNO=1
//SYSIN    DD *
 <class source code>
/*
```

Notes:

a. The class code submitted on SYSIN must start with the word class or a prefix identifier or an external class declaration, i.e. it must not start with begin.

b. If DISP=OLD is coded on the SYSPUNCH statement and there was already a member called Mname in the library, it will be overridden by the currently compiled class. DISP=NEW disables the overriding.

c. Mname may conveniently be identical to the class identifier but this is not a rule.u

External class usage in the main program can be achieved by use of procedure SIMCLG:

```
//USEC  EXEC  SIMCLG
//SYSLIB DD   DSN=library-name,DISP=SHR
SYSIN    DD   *
 <SIMULA program (or class to be separately compiled)
  which contains the external class declaration>
/*
```

Note:

a. External class declaration has one of the following forms:

```
external class classname;
external class classname=Mname;
```

This is placed among the declarations, and the line itself must not contain anything else.

The second alternative is used in case that classname is different from Mname.

Example:

The following is an example showing how to use an external class MYSIMSET as a program prefix:

```
//USEC1  EXEC SIMCLG
//SYSLIB DD   DSN=library-name,DISP=SHR
//SYSIN  DD   *
external class MYSIMSET;
MYSIMSET begin
         <program declarations and statements>
         end of program;
/*
```

4.16 Assembly and Fortran procedures.

Any parameter to an EXTERNAL ASSEMBLY PROCEDURE must be a declared variable, a parameter called by value, a declared array, an array parameter not called by name or a label local to the block of the call. An EXTERNAL <type> ASSEMBLY PROCEDURE may have any type.

The same rules apply to an EXTERNAL FORTRAN PROCEDURE and an EXTERNAL <type> FORTRAN PROCEDURE, but types REF, TEXT, LABEL and formal arrays are not permitted.

4.17 Random drawing procedures.

The procedures for probability distributions use Lehmers multiplicative congruence method for random number generation:

$$U(i) = lambda * U(i-1) \pmod P$$

$$X(i) = U(i)/P$$

with

$$P = 2^{32} = 4294957296$$
$$lambda = 5^{13} = 1220703125$$

Each X(i) is computed with 24 bits significance. Since the U(i) are represented with 32 value bits and no sign bit, the antithetic drawings are obtained by reversing the sign of the initial value.

Since it may be important to know the algorithms used for obtaining the various distributions, a formal definition is given in SIMULA. The utility procedures XI(U) and UI(U) return X(i) and U(i), respectively, and they also replace U(i-1) in U by U(i). The checks for parameter validity are not shown, but if the algorithmic description implies a reference to a non-existent array element, then this will be detected.

```
BOOLEAN PROCEDURE DRAW(A,U); NAME U; REAL A;
        INTEGER U;
DRAW := (XI(U)<A);

INTEGER PROCEDURE RANDINT(A,B,U); NAME U;
        REAL A, B; INTEGER U;
RANDINT := (UI(U)//2*(B-A+1)//2**31+A;
COMMENT THE PRECEDING EXPRESSION CANNOT BE EVALUATED
        IN SIMULA FOR 360 (FIXED OVERFLOW);
REAL PROCEDURE UNIFORM(A, B, U); NAME U;
        REAL A, B; INTEGER U;
UNIFORM := XI(U)*(B-A)+A;

REAL PROCEDURE NEGEXP(A,U); NAME U;
        REAL A; INTEGER U;
NEGEXP := -LN(XI(U))/A;

INTEGER PROCEDURE POISSON(A,U); NAME U;
        REAL A; INTEGER U;
        BEGIN INTEGER T; REAL R, R1;
              R1 := EXP(-A); R := 1;
        L:    R := R*XI(U); IF R>=R1 THEN
              BEGIN T := T+1; GOTO L END;
              POISSON := T;
        END   POISSON;
```

```
REAL PROCEDURE ERLANG(A, B, U); NAME U;
     REAL A, B; INTEGER U;
     BEGIN REAL AB, P;
           AB := A*B; P := 1;
           FOR B := B-1 WHILE B>=0 DO
           P := P*XI(U);
           P := LN(P);
           IF B <>0 THEN
           P := P-B*LN(XI(U));
           ERLANG *= -P/AB
     END   ERLANG;

COMMENT LSB AND USB BELOW ARE NOT USER-ACCESSIBLE;
INTEGER PROCEDURE LSB(A); ARRAY A;
        COMMENT RETURNS LOWER SUBSCRIPT BOUND
                FOR A ONE-DIM ARRAY; ...;

INTEGER PROCEDURE USB(A); ARRAY A;
        COMMENT RETURNS UPPER BOUND; ...;

INTEGER PROCEDURE DISCRETE(A, U); NAME U;
        ARRAY A; INTEGER U;
        BEGIN INTEGER I, J; REAL X;
              X := XI(U); I := LSB(A); J := USB(A)-I;
              FOR J := J//2 WHILE J <>0 DO
              IF A(I+J)<=X THEN I := I+J;
              FOR I := I+1 WHILE I<= USB(A) DO
              IF A(I)>X THEN GOTO L;
        L:    DISCRETE := I;
        END   DISCRETE;

REAL PROCEDURE LINEAR(A, B, U); NAME U;
     ARRAY A, B; INTEGER U;
     BEGIN INTEGER I, J; REAL X;
           X := XI(U); I := LSB(A); J:= USB(A)-I;
           FOR J := J//2 WHILE J <>0 DO
           IF A(I+J)<= X THEN I := I+J;
           FOR I := I+1 WHILE A(I)<=X DO

           LINEAR := B(I-1)+(X-A(I-1))*(B(I)-B(I-1))
                     /(A(I)-A(I-1))

     END OF LINEAR;

INTEGER PROCEDURE HISTD(A,U); NAME U; ARRAY A;
        INTEGER U;
        BEGIN REAL S; INTEGER T;
              FOR T := LSB(A) STEP 1 UNTIL USB(A) DO
              S := S+A(T);
              S := S*XI(U); T := LSB(A);
        L:    S := S-A(T); IF S>=0 THEN
              BEGIN T := T+1; GOTO L END;
              HISTD := T;
        END OF HISTD;
```

```
REAL PROCEDURE NORMAL(A,B,U); NAME U;
     REAL A,B; INTEGER U;
     BEGIN REAL X,L,S;
     A1:   X := 2*XI(U)-1;
           S := XI(U)**2+X*X;
           IF S>1 THEN GOTO A1;
           L := SQRT(-2*LN(X1(U))/S);
           NORMAL := X*L*B+A;
     END NORMAL;
```

4.18 Attribute protection.

The definition of attribute protection is not fully implemented.

Programs containing hidden and protection specifications will be checked syntactically, but the specifications will be treated as comments.

A warning is given indicationg the lack of semantical implementation of the feature.