Norsk Regnesentral
Norwegian Computing Center
January, 1983

SIMULA
Programmer's Guide
IBM System 360/370

Contents

0 Introduction

1 System requirements
1.1 Computer model
1.2 Operating system
1.3 Secondary storage

2 System elements
2.1 Job control language
2.1.1 Job statement
2.1.2 Execute statement
2.1.3 Data definition statement
2.1.4 Catalogued procedures
2.2 SIMULA system elements
2.2.1 Compiler
2.2.1.1 Compiler parameters
2.2.2 Linkage Editor
2.2.3 Object program
2.2.3.1 Object program parameters
2.2.3.2 Return code
2.2.4 Storage management under RTS
2.3 External procedures and classes
2.4 Use of catalogued procedures
2.4.1 Using the SIMULA system with the JCL procedure SIM.

3 Hardware and implementation defined restrictions and capacity limitations
3.1 Permitted ranges of arithmetic quantities, precision of real number arithmetic
3.2 Maximum sizes of block instances, arrays and texts
3.3 Program capacity limitations
3.4 Object program storage requirements

4 Implementation defined parts of the SIMULA language
4.1 Use of system prefixes
4.2 Accepted virtual specifiers and rules for virtual matches
4.3 Collating sequence
4.4 Initialisation of character variables
4.5 Evaluation of Boolean expressions
4.6 For statements
4.7 Parameter type correspondence
4.8 Separator
4.9 Power-of-ten symbol
4.10 Edit overflow
4.11 Formats of editing procedures
4.12 Mathematical subroutines
4.13 Array subscript checking
4.14 External procedures
4.15 External classes
4.16 Assembly and Fortran procedures
4.17 Random drawing procedures
4.18 Attribute protection

5 Object program input/output
5.1 OS 360/370 data management terminology

5.2 Principles of OS 360/370 data management
5.2.1 Device independent restrictions
5.2.2 Run-time specification of I/O processing parameters
5.3 Data definition statement (DD-statement)
5.3.1 Binding a file to a data set
5.3.2 Creating a data set on magnetic tape, disk or drum
5.3.2.1 Naming a data set
5.3.2.2 Allocating a data set
5.3.2.3 Direct-access storage
5.3.2.4 Magnetic tape
5.3.3 System output
5.3.4 Retrieving a data set
5.3.5 System input
5.3.6 Disposition of a data set
5.3.7 Sequential data sets: characteristics and processing options (outfile, infile)
5.3.7.1 Fixed record length
5.3.7.2 Blocked records
5.3.7.3 Variable and undefined record formats
5.3.7.4 Carriage control character
5.3.7.5 Additional DCB subparameters
5.3.8 Printfile data sets
5.3.9 Direct data sets
5.4 End-of-file text
5.5 Sysin and sysout

6 Debugging aids
6.1 Diagnostics
6.2 Tracing
6.2.1 Program control flow tracing
6.3 The symbolic dump facility
6.3.1 Design principles
6.3.2 Dump format
6.3.2.1 Blocks
6.3.2.2 Local quantities
6.3.2.3 Arrays
6.3.2.4 Text object
6.3.3 System overhead
6.3.4 System control
6.3.4.1 Compiler
6.3.4.2 RTS
6.3.4.3 Additional control remarks
6.3.5 Example
6.3.6 External utilities related to the SYMBDUMP facility
6.4 Dump

7 Dynamic profile of program execution
7.1 Objectives and design principles
7.2 Scope of the facility
7.2.1 Assignment counter
7.2.2 Frequency monitor
7.2.3 Data structure register
7.3 Output format of the respective components
7.3.1 Assignment count
7.3.2 Assignment frequencies
7.3.3 Generated data structures
7.4 Control
7.5 Example

8 Bibliography

Appendices

A HARDWARE REPRESENTATION OF THE SOURCE LANGUAGE

B COMPILER DIAGNOSTICS

C RUN-TIME DIAGNOSTICS

D SIZES OF RUN-TIME LIBRARY ELEMENTS

E HOW TO DESIGN AN OVERLAY STRUCTURE

F INTERNAL REPRESENTATION OF DATA STRUCTURES

G HOW TO WRITE AN EXTERNAL ASSEMBLY OR FORTRAN PROCEDURE

H SPECIFYING USER EXITS

I SAMPLE JOB LISTING

J CATALOGUED PROCEDURES

K REPORT PROCEDURE

0 Introduction.

This SIMULA Programmers Guide is intended to cover all parts of SIMULA for 360/370 not covered by the defining document of SIMULA (see (12)).

This manual covers the essential features of System 360/370 OS Job Management, but is not meant to duplicate the IBM documentation of these items (see section 8). To take full advantage of the operating system facilities and installation dependent features, consult the IBM documentation and the system programmers at your own installation. This should not, however, be necessary except for unusually advanced applications, and when you have very high efficiency requirements.

If familiar with System 360/370 OS, you may skip sections 1, 2, 3.1.1 and 5, except for 2.2.1, 2.2.3, 2.3, 5.2.1, 5.3.1, 5.3.7, 5.4 and 5.5.

1 System requirements.

1.1 Computer model.

The 360/370 IBM SIMULA can be used on a 360 or 370, with full instruction set.

The core storage should be less than 16,340 K bytes, and greater than or equal to 128 K.

1.2 Operating system.

The system operates under any of the following operating systems: OS (MFT, MVT, MVT with TSO) and their VS (VS1/VS2) counterparts, CMS, VM, MTS, GUTS, MVS and TSS, and other operating systems that are compatible with one of these. The system cannot operate under DOS.

1.3 Secondary storage.

The compiler uses secondary storage for intermediate tables and partially translated code. It will not need more direct access secondary storage than three times the size of the source program.

2 System elements.

2.1 Job control language.

The environment of 360/370 SIMULA is the 360/370 Operatin System. Actions to be performed under this system are specified by a job control language (JCL), the statements of which are called job control statements. The job control language is considered to be more complicated and more difficult to use than other similar control languages, but it is also very flexible and useful once it is mastered.

The general format of a JCL statement is:

```
//name  operation   operands
```

The two slashes are in columns 1 and 2 of a line and identify it as a JCL statement.

name	is an identifier of not more than 8 consecutive alphameric characters, the first of which is a letter. The name starts in position 3.
operation	is a word separated from the name and from the operands by at least one blank. The words considered here are JOB, EXEC and DD, identifying a job statement, an execute statement and a data definition statement, respectively.
operand	is a list of operands separated by commas, with no intervening blanks. The operands are either positional (the meaning of the operand is determined by its relative position in the list) or keyword (the meaning is determined by a keyword followed by an equal sign preceding the operand). An operand list can contain only positional operands, only keyword operands, or both, but all positional operands must precede all keyword operands in the operand list. The operand list can be continued on the next line if it is interrupted after a comma. The following line is then marked with two slashes in position 1 and 2, and the operand list is continued starting anywhere between the 4th and 16th position. No part of an operand list may extend past position 71.

The use of each particular JCL statement is described in the succeeding sections. For reasons of readability and simplicity several operands are omitted and the rules are sometimes given in a stricter form than is necessary when this does not decrease usability. The actual rules are given in (6).

2.1.1 Job statement.

The largest unit of work recognized by the system is a job, defined by a job statement and the following JCL statements up to the next job statement in the input stream.

The name field of the job statement is the job name, used to identify the printed output from the job. The operands of a job statement supply account inginformation and programmer name by positional operands. Keyword operands can be used to specify whether the JCL statements of the job will appear on the message listing, and to specify a minimum size of core in which the job can be run.

Example:

```
//A  JOB  0,SMITH,MSGLEVEL=1,REGION=90K
```

A	is the job name
JOB	identifies the this as a job statement
0	is the accouting information. In general you must replace the 0 by your account number and possibly some other information depending on the installation.
SMITH	is the programmer name. Special rules apply if the name contains blanks or other nonalphameric characters (see (6)).
MSGLEVEL=1	requests the system to let the JCL statements of the job appear on the message listing. If omitted, no JCL statement will appear.
REGION=90K	is a request for a problem program partition of not less than 90 K bytes. The parameter is an unsigned integer followed by a K indication multiplication by 1024. In a system with LCS support (release 17 or later) storage of both hierarchies can be requested (see (6)).

2.1.2 Execute statement.

A job consist of one or more job steps. A job step is defined by an execute statement and the data definition statements (DD-statements) following it. The job steps of a job will be executed sequentially in the order in which they appear within the job. The execute statement defines a program which will operate on data defined by DD-statements of the job step. The name field of the EXEC statement is the job step name. The operands of the execute statement define the program to be executed and, optionally, a parameter string to the program and a test for bypassing the step.

Example:

```
//S1  EXEC  PGM=progname,PARM='parm string',COND=conditions
```

S1 — This is the step name (name of the execute statement) of the job step to be executed. This is used for referencing the step for instance in JCL messages.

PGM=progname — The PGM operand defines the program to be executed. Progname os the name of a load module (2.2.2). in the system link library, the job library or the step library.

The system link library (SYSL.LINKLIB) is always accessible. A job library can be defined by a DD-statement with the ddname JOBLIB, immediately following the job statement. A step library is defined by a DD-statement with the ddname STEPLIB, placed among the DD-statements of the step.

PARM='parm string'

A parameter in the form of a character string is passed to the program by means of the PARM operand. The interpretation of the parameter string is dependent on the program. The parameters allowed for the SIMULA compiler and the SIMULA object program are discussed in sections 2.2.1 and 2.2.3, respectively.

COND=conditions — conditions is a condition or a list of conditions separated by commas and enclosed in parentheses. If any one of the conditions is satisfied, the step will not be executed. A condition has the form (unsigned integer, relop, stepname)

relop is any of EQ, GE, GT, LE, LT or NE,
meaning=, >=, >, <=, < or ,
respectively.
stepname is the name of a preceding job step in the job.

When a program terminatesnormally, a return code is passed to the operating system. It the stepname of the conditions is replaced by the return code of that step you get an arithmetic relation, and the condition is satisfied if this relation is true. The return codes passed by application programs are discussed in subsections of 2.2.

A condition can also be any of the words EVEN and ONLY.

EVEN is never satisfied, which implies that the step will be executed even if an earlier step terminated abnormally.

ONLY is satisfied only if no earlier step has terminated abnormally.

If neither EVEN or ONLY occurs among the conditions, the job step will be bypassed if an earlier job step terminated abnormally.

Special rules apply if EVEN of ONLY is mixed with relational conditions (see (6)).

2.1.3 Data definition statement.

A data definition statement (DD-statement) defines either data on which the program of a job step will operate, or a program library from which the program is to be loaded. The name field of the DD-statement is the ddname, which connects the data to a logical file of the program. A detailed description on the use of DD-statement is found in section 5.

2.1.4 Catalogued procedures.

In order to reduce the number of control statements for frequently used job step combinations, the system provides a cataloguing facility for JCL statements. A set of catalogued JCL statements, constituting one or more job steps (procedure steps) is called a catalogued procedure. The operands of the JCL statements can be modified when the procedure is used by means of symbolic parameters. The operands that can be changed in this way are defined when the procedure is catalogued. Operands not given symbolic parameters can be overridden when the procedure is used. The use of the catalogued procedures supplied as part of the 360/370 SIMULA is described in section 2.4.

Invoking a catalogued procedure.

A catalogued procedure is invoked with an EXEC statement in which the keyword PGM operand is replaced by a positional operand which is the name of the procedure. The keyword operands of this statement are the symbolic parameters of the procedure.

Modifying execute statements of a procedure.

Operands of the execute statements of the procedure can be redefined in the EXEC statement invoking the procedure by means of operands of the form

```
keyword.procstepname=newoperand
```

where keyword is the keyword of the operand which is to be added or modified, procstepname is the name of the execute statement in the procedure which is to be modified, and newoperand is the new operand value.

If operands of more than one procedure step are redefined, the operands must occur in the same order as the procedure steps to which they apply.

Modifying and adding DD-statements in a procedure step.

Operands of a DD-statement of a procedure step can be modified by means of an overriding DD-statement:

```
//procstepname.ddname DD newoperands
```

DD-statements can be added to a procedure step with an ordinary DD-statement where the ddname is preceded by the procedure step name and a dot.

Overriding and added DD-statements must be sorted so that DD-statements belonging to a later step follow those belonging to an earlier step, and within each step overriding DD-statements must occur in the same sequence as the overriding DD-statements in the procedure.

Overriding DD-statements of a procedure step must precede added DD-statements of that step.

Some examples on modifying and added DD-statements are found in 2.4.

2.2 SIMULA System Elements.

The software elements needed to use 360/370 SIMULA are the SIMULA compiler, named SIMULA, the IBM-supplied Linkage Editor, named IEWL, and the run-time system load module library, named SIMLIB.

Furthermore, the JCL procedures of Appendix J should be catalogued, i.e. put in the catalogued procedure library SYS1.PROCLIB.

2.2.1 Compiler.

The SIMULA compiler will read a SIMULA source program (Appendix A), and it will produce one or more of the following items:

i) a source program listing,
ii) a cross-reference listing,
iii) an object module,
iv) a copy of the object module and
v) a diagnostic message listing.

The parameter string passed to the compiler from the EXEC statement determines which items are produced (2.2.1.1).

The object module is a machine-language, non-executable version of the source program (2.2.2).

The source program and cross-reference listings are useful for debugging and documentation.

When an error or a possible error in the source program is detected, an error or a warning message (Appendix B) is written. Production of the object module is suppressed if an error with severity code greater than 3 has been detected by the compiler.

The files used by the compiler are given in Table 2.1.

The return code issued by the compiler is the highest diagnostic severity code encountered (Appendix B), rounded upwards to a multiple of 4.

ddname	mode	use
SYSIN	input	contains the source program
SYSGO	output	will contain the object module produced by the compiler.
SYSLIB	input, partitioned	contains external class definitions used in the source program.
SYSPRINT	output	will contain the source listing, the cross-reference table and diagnostic messages.
SYSPUNCH	output	will contain the object deck (identical to the object module).
SYSUT1 SYSUT2 SYSUT3 SYSUT4	input, ouput	temporary data sets used to hold compiler created tables and partially translated code if these do not fit into the available memory.

Table 2.1: Compiler files

2.2.1.1 Compiler parameters.

Compiler control is achieved by means of the parameters coded in the PARM field of the compiler EXEC statement. The compiler parameters are a number of words, separated by commas and with no intervening blanks.

The parameters are separated into two groups, those that are only given by a keyword (positional keyword), and those that are given by a keyword and a value (value keyword).

Positional keywords are used to invoke various processing options of the compiler, and any of these can be preceded by NO which has a suppressing effect.

Word:	Meaning:
LIST	an assembly-like version of the compiled program is listed on SYSPRINT.
LOAD	an object module is produced on SYSGO.
DECK	a copy of the object module is written to SYSPUNCH.
WARN	warning messages will appear on the diagnostic message listing.
SUBCHK	instructions to check array indexing are compiled.
EXTERN	the source program on SYSIN is an external procedure definition.
XREF	a cross-reference listing is produced on SYSPRINT.
SOURCE	the source program is listed on SYSPRINT.
RESWD	reserved words will be underlined on the source listing. This is accompanied by by double printing and on most systems two lines will be charged for each line which contains a reserved word.
TERM	output abbreviated error messages on SYSTERM.
LONGREAL	All arithmetic computations involving real quantities should be performed in long real precision.

The following parameters are specified with a value following the parameters and an equalsign.

LINECNT=n — n is the number of lines per page written on SYSPRINT.

MAXERROR=n — n is the maximum number of diagnostic messages (errors and warnings) to be printed. Further diagnostics will be counted, however.

MAXLINES=n — n is the maximum number of lines to be listed in the source listing.

MAXPAGES=n — n is the maximum number of pages in the source listing. If the value is exceeded, the processing is terminated. n=0 is interpreted as "no limit".

INDENT=n — n is the number of positions that each block level is to be indented (i.e. shifted right) in the source listing to show the block structure of the program.

TIME=n — specifies that the compilation should be stopped after n/1oo cpu seconds. n=0 is interpreted as "no limit".

SYMBDUMP=n — specifies that provision of compiler generated data structures for runtime debugging is to be made, see section 6.3.

SIZE=(s1h1,s2h2,s3h3,s4h4,s5,s6)

The SIZE parameter controls the internal use of core by the compiler. The default size will be appropriate for most purposes, otherwise see section 2.2.1.2.

RESWD=n — specifies special marking of reserved words in the source listing. The value n has the following meaningful values:

0	no marking (identical to NORESWD)
1	underline reserved words (identical to RESWD)
2	reserved words in triple printing
3	reserved words in lower case, identifiers in uppercase and standard names in lower case with capital letter.
4	reserved words in upper case, identifiers in lower case and standard names in lower case with capital letter.

The underlining of the reserved words in the compilation listing is only possible if the applied printer allows printing of more than one image at one physical line and the printer chain contains the underscore (_) character.

EXTERN=v — Specifies the kind of compilation:
O: main program,
C: external class,
P: external procedure.
'EXTERN' is equivalent to EXTERN=P, and
'NOEXTERN' is equivalent to EXTERN=O.

The default parameters and values are:

```
'NOLIST,NODECK,LOAD,WARN,SUBCHK,NOEXTERN,NOXREF,SOURCE,
 NORESWD,LINECNT=60,MAXERROR=50,INDENT=0'.
```

SIZE parameter

The SIZE parameter is composed of 6 subparameters, written in the general format:

SIZE=(s1h1,s2h2,s3h3,s4h4,s5,s6)

Each si (i=1,2,...,6), stands for a value of the form ddd,,d (a decimal integer) or ddd...dK, where K means multiplication by 1024, or it may be omitted, leaving the default value unchanged.

Each hi (i=1,2,3,4) is either "H1" or omitted.

"H1" means that the corresponding area or areas should be placed in LCS (large core storage, hierarchy 1). Otherwise the area(s) are placed in HSS (high speed storage, hierarchy 0).

Trailing commas need not be written.

The subparameters have the following significance (refer to "DEFAULT"):

s1 - DSYMBUFL	length of buffers for SYSUT1 and SYSUT2 (intermediate language symbols). Preferably a multiple of 16.
s2 - DIDBUFLE	length of buffers for SYSUT3 (idenfifier list). Should be a multiple of 16.
s3 - SYSFREE	work area size for system use. Must accommodate one or two buffers for each of SYSGO and SYSPUNCH (if used), in addition to some space for transient system modules.
s4 - COREMIN	this is the minimum work storage needed by the compiler. If DIDBUFLE and COREMIN have the same storage hierarchy, and COREMIN <= LIMBLKSZ, then the actual min. work area size is (2*DIDBUFLE+COREMIN) except in passes 1,2 and 9, because SYSUT3 buffers are released when not in use in that case.
s5 - COREMAX	maximal amount of core requested in any hierarchy. Must be larger than the sum of component areas in the respective hierarchies. The effective components are: 1*SYSFREE, 1*COREMIN, 2*DIDBUFLE, 4*DSYMBUFL, 8*MAXERROR.
s6 - LIMBLKSZ	largest block size allowed to be written on SYSUT1-3. A larger value of DIDBUFLE or DSYMBUFL forces the corresponding file to be kept entirely in-core.

Default values of S1,...,S6 are installation dependent, and, for the sake of efficiency, it is important that any change in buffer size etc. should be reflected in the corresponding DD statement.

2.2.2 Linkage Editor.

The Linkage Editor is an IBM-supplied program which reads the object module and produces an executable object program.

An executable program is always a member of a program library on direct-access secondary storage. In IBM terminology it is called a load module, since it has a different organization from an object module and it can be loaded into core by the control program.

The object module (primary input) is read from a file with ddname SYSLIN. Additional input is taken from a loaded module library identified by the ddname SYSLIB to resolve external references (automatic library call). The load module produced is put in the library identitied by ddname SYSLMOD, under a name given in the DD-statement.

Information and diagnostic messages, as well as an optional map and cross- reference listing, useful for debugging on the machine-code level, are printed on a data set with ddname SYSPRINT.

The Linkage Editor will optionally perform a large number of functions requested by control statements in the primary input. One of these functions, overlay editing, is described in Appendix E, while the remaining functions are described in (10).

2.2.3 Object program.

The object program output from the Linkage Editor is an executable program corresponding to the source program. When this program is executed by means of an execute statement, the machine equivalents of the SIMULA statements will be executed. The PARM operand of the execute statement can be used to control the storage setup and the amount of debugging information produced (2.2.3.1).

The return code to the operating system can be used in COND operands of following job steps (2.2.3.2).

If a run-time error occurs during the execution of the object program it is terminated at once and a diagnostic message is printed together with programmer controlled debugging information on a file identified by the ddname SYSOUT (Section 6).

The run-time system is assigned the three letter prefix ZYQ, occurring in all diagnostic messages and control section names.

2.2.3.1 Object program parameters.

The parameters to the object program are given by the PARM operand of the execute statement in the form of a character string enclosed in quotes. The character string consists of keyword operands separated by commas with no intervening blanks. If a permitted keyword is omitted, the default value of that operand is used. The permitted keywords and operand formats are shown below.

DUMP=digit — The DUMP parameter controls the post-mortem dump (see section 6.2). The digit is 0,1,2,3,4,5,6 or 7. Default value is 1, giving a diagnostic message and a register dump.

HIARCHY=0 or 1 — In a system with LCS support this operand determines the hierarchy of the SIMULA working storage. The default value is 0 (fast core storage).

LINECNT=unsigned integer

The unsigned integer is the initial value of the variable LINES PER PAGE of a printfile object.

60 is the default value.

MAXPAGES=unsigned integer

The amount of user program output on SYSOUT is controlled together with all other printfiles. If the value is exceeded, the processing is terminated. A value of zero means "no limit".

SIZE=(q1,q2,q3) This operand controls the partitioning of available storage into SIMULA working storage and system free storage. The size of the SIMULA working storage is fixed during execution, and is used for declared quantities, texts, arrays and control blocks for program sequencing and the sequencing set, see section 2.2.4 for further details.

q1, q2 and q3 are unsigned integers, optionally followed by the letter K indicating multiplication by 1024.

Only those of the sizes which are required may be specified, however, only trailing commas may be omitted.

All sizes are measured in bytes.

q1 is the maximum expected sum of the sizes of all blocks, arrays, texts and temporary results in the SIMULA program. If this size is exceeded, the object program is terminated.

If q1 is omitted it is set equal to q3.

q2 is the minimum size of the system free storage. It should be large enough to accommodate access routines, I/O buffers, and all GETMAIN, LINK and LOAD requests from non-SIMULA external procedures.

q2 is set to 10K if omitted.

q3 is the desired size of the SIMULA working pool. If q3 is omitted all storage except that defined by q2 is allocated.

If q2 and q3, are both specified, q2 takes precedence if the sum of q2 and q3 is greater than the available storage. If the sum is less, q3 takes precedence.

TRACE=unsigned integer

This operand determines the size of the tracing buffer for control and dataflow tracing (See section 6.3).

TEST
NOTEST print header lines prior to the first line of the program output indicating the compilation date, release identification and the options in force and traling lines indicating the total exectuion time, return code and the time spent in garbage collection (if any).

TIME=unsigned integer

set cpu-time limit for execution in hundredths of seconds - the default value = no limit. Appropriate diagnostics and optional dump are obtained in the case of limit overflow rather than abnormal termination, thus facilitating the location of unintended infinite loops.

SYMBDUMP=digit specifies the level of information to be given in a post-mortem symbolic dump (see section 6.4). The digit is 0, 1, 2, 3, 4, 5 or 6. Default value is 3, giving the heading of blocks on the operating chain and a symbolic dump of the local quantities in the involved blocks.

The default values indicated are common standard. They can, however, be altered to any other default values at system installation using the SIMULA system macro SIMRDF which is a standard part of every system delivery. The same macro also allows alteration of the standard ddnames of the RTS files.

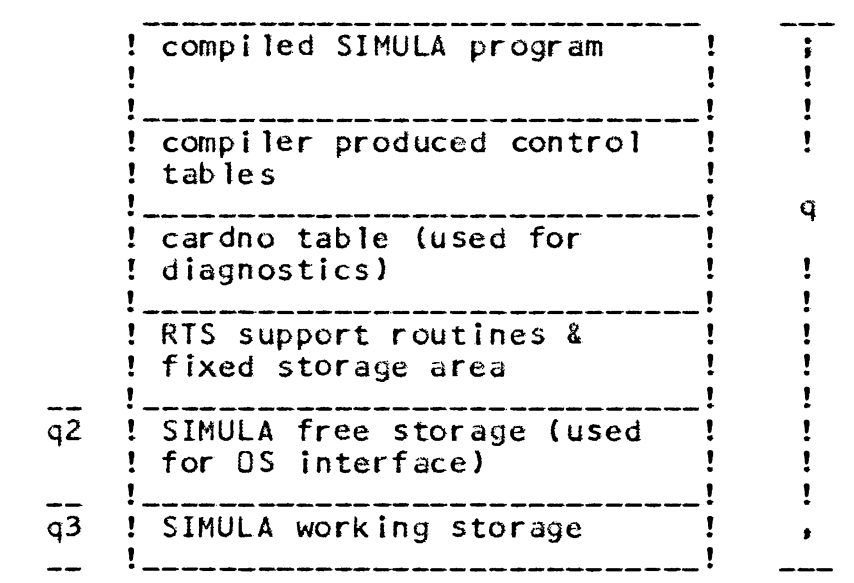
2.2.3.2 Return code.

The object program passes a return code to the operating system indicating the way the program was completed.

Code:	Meaning:
0	Successful completion.
4	One or more edit overflows occurred.
8	The program was terminated because of a run-time error.
12	One or more edit overflows occurred and the program was terminated because of a run-time error.

2.2.4 Storage management under RTS.

The following is a simplified scheme of the storage allocation during a SIMULA program execution:



This figure illustrates use of storage in the case where a SIMULA program is executed in a separate JCL step (it has to be modified accordingly for situations where the execution is invoked using the OS loader or NCC's utility SIMCNT). In this case the value of q is equal to the size of the region allocated to this task by the operating system.

The size of the SIMULA free storage is 10K by default. If more core than q2 bytes is needed for OS interface routines (e.g. when performing I/O using extraordinarily large buffers), the execution terminates abnormally (usually with the System Completion Code 804).

The SIMULA working storage is by default the largest contiguous area of core left within the region after all other core requests are fulfilled, and its size is printed in the RTS header.

The internal organisation of the SIMULA working storage is as follows:

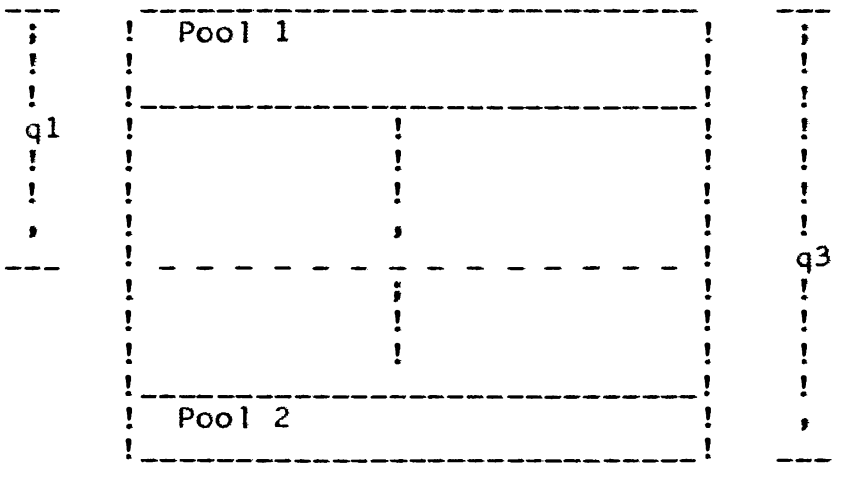


fig. 2.

Pool 1 is used for block instances, array and text objects and temporary results; pool 2 is used for control blocks used by the RTS working storage management system. The governing idea of this system is that pool 1 and pool 2 are extended in the course of program execution in the respective directions indicated by arrows in fig. 2 until they use up all the working storage area. The garbage collection is then automatically performed which deletes all unreferenceable data structures so that further extensions of the pools are possible. If garbage collection does not result in a sufficient gain of core, the RT error ZYQ0017 (STORAGE EXHAUSTED) is forced.

Further, it is the user's option to set up a limit for pool 1, namely the value of q1 in fig. 2. If this is done and pool 1 reaches this limit in the course of the program execution without garbage collection being able to compress pool 1 beyond this limit, the RT error ZYQ0018 (DATA LIMIT) results.

The value of q1 is zero by default, which the RTS interprets as "no pool 1 limit" and allows pool 1 to expand until it reaches pool 2.

A user may exercise control over the RTS storage management via the parameter SIZE, where the respective values of q1, q2 and q3 may be specified in the form of subparameters, see section 2.2.3.1.

A user should bear in mind that:

- the range of the values of the respective subparameters is dependent on the program size, the size of the requested RTS support and the region size.

- if both q2 and q3 are specified, q2 takes precedence if the sum of q2+q3 is greater than the available storage, otherwise, q3 takes precedence.

- the main storage request issued by the RTS via GETMAIN on behalf of the SIZE subparameter q3, is conditional and no extra action is taken if the requested amount of the main storage is not available.M

2.3 External procedures and classes.

External procedures and classes make it possible to compile procedure and class declarations separately.

An external procedure can be saved as an object module (2,4. ex. 7) or as a load module (2,4 ex. 8).

An external class is saved in an image library.

External procedures can be replaced in a program without recompilation of the main program, but external classes cannot.

External procedures can be of three different types: SIMULA, FORTRAN or ASSEMBLY. How to write an external procedure of type FORTRAN or ASSEMBLY is described in Appendix G.

2.4 Use of catalogued procedures.

The following sample jobs illustrate the use of the catalogued procedures listed in Appencix I.

The catalogued procedures are:

SIMC compile source program, create object module. The procedure consists of one procedure step named SIM, which is an execution of the SIMULA compiler. The procedure has no symbolic parameters.

SIMCL Compile source and linkedit to executable PROGRAM. Procedure step names are SIM and LKED. Symbolic parameters:

PROG=progname
progname will be the name of the created program.

LIB='libname'
libname is the name of the catalogued library in which the object program is placed.

EXLIB='libname1'
libname1 is the name of the catalogued library from which external procedures are taken.

LDISP=OLD or MOD
OLD: the object program is to replace an existing, identically named program in the library.
MOD: The object program is to be added to the library.

SIMCLG Compile, linkedit and execute object program. Procedure step names are SIM, LKED and GO. Symbolic parameter:

EXLIB='libname'
Name of library from which external procedures are to be loaded.

SIMG Execute a SIMULA object program from a load module library. The only procedure step of SIMG has the name GO. Symbolic parameters:

PROG=progname
Name of program to be executed.

LIB='libname'
libname is the name of the load module library.

SIMCG Compile cource and execute the LOADER program (which combines the steps LKED and GO in SIMCLG). Step names are SIM and GO. Symbolic parameter same as in SIMCLG.

SIM A one-step JCL-procedure which makes use of a special program called SIMCNT. This program is a standard part of every SIMULA system, starting with release 3.0. For details on this procedure, see section 2.4.1.

Note: The mandatory job statement has been omitted in all the following examples.

```
//A  EXEC   SIMC,PARM.SIM='NOLOAD,XREF'     1
//SIM.SYSIN   DD  *                         2
   <source program deck>
/*                                           3
```

Ex. 1: Compile program to obtain program listing, cross-reference listing and diagnostic messages.

Explanation (digits refer to line numbers of the example).

1: This line invokes the catalogued procedure SIMC. The PARM operand of its only step, SIM, is replaced by 'NOLOAD,XREF', which will suppress object module generation (NOLOAD) and cause the cross-reference listing to be printed (XREF).

2: This dd-statement defines the source program data set. The asterisk indicates that the data set follows this line in the job stream.

3: The delimiter statement signals the end of the source program.

```
//A  EXEC   SIMC,PARM.SIM='DECK,NOLOAD'          1
//SIM.SYSPUNCH DD DSN=MYSAVE,UNIT=3330-1,---     2
//SIM.SYSIN DD   *                               3
     <source program
/*                                               4
```

Ex. 2: Compile and save program.

When the program has been found to be free from errors, the object module can be saved. This will mean that when the program is to be executed it is not necessary to recompile it each time.

1: Since we want to produce an object module but not load it, the parameter to the compiler is 'DECK,NOLOAD'.

2: This DD-statement has been added to the procedure step. SYSPUNCH is the ddname on which the module is stored.

3 and 4:
Same as 2 and 3 in the preceding example.

```
//A  EXEC  SIMCLG          1
//SIM.SYSIN DD *           2
  <source program>
/*                         3
//GO.SYSIN  DD *           4
  <test data>              5
/*                         6
```

Ex. 3: Compile, linkedit and execute.

This is the typical test run in a debugging cycle. The object program is not saved.

1: This statement invokes the SIMCLG catalogued procedure.

2 and 3:
Same as in the previous examples.

5: Test data to be read by sysin.

6: Signals end og file for sysin.

When you want to use the LOADER program replace SIMCLG by SIMCG in line 1.

```
//A  EXEC  PGM=IEFBR14                          1
//LIB  DD DSN=A.LLIB,DISP=(NEW,CATLG),         C2
//  UNIT=2314,VOL=SER=XXXXXX,                  C3
//  SPACE=(TRK,(10,5,20))                       4
//B EXEC  SIMCL,PROG=PROGA,LIB='A.LLIB'         5
//SIM.SYSIN  DD *                               6
    <source program>                            7
/*                                              8
```

Ex. 4: Create a program library in which an object program is saved.

The library is created in a dummy step preceding the compilation.

1: This statement requests an execution of the dummy program IEFBR14.

2: The first line of the DD-statement requests the system to catalogue a new data set with data set name A.LLIB. A must be the name of an index in the catalogue.

3: The first continuation line defines the disk pack on which the data set will be put: 2314 disk pack with serial number XXXXXX.

4: The second continuation line defines the space allocated to the data set (section 5.3.2.2). The number 20 indicates that the data set will be a library with 20 directory blocks, which will permit approximately 5 * 20 = 100 different programs in the library.

5: This statement invokes the SIMCL catalogued procedure. The symbolic parameters request the linkage editor to add the object program to the library A.LLIB under the name PROGA.

```
//A  EXEC    SIMCL,PROG=PROGA,LIB='A.LLIB',LDISP=OLD    1
//SIM.SYSIN    DD *                                      2
   <source program>
/*                                                      3
```

Ex. 5: Compile and replace a program in an existing program library.

The first statement of this job requests the linkage editor to replace the program PROGA in A.LLIB with the object program resulting from this compilation.

```
//A  EXEC   SIMG,PROG=PROGA,LIB='A.LLIB'                1
//GO.SYSIN   DD *                                       2
    <execution data>
/*
```

Ex. 6: Execute an object program from an old program library.

When an object program has previously been saved, the above example shows how it will be executed.

```
//A  EXEC  SIMC,PARM.SIM=EXTERN                         1
//SIM.SYSIN  DD *                                       2
  <external procedure source>                           3
/*                                                      4
//B  EXEC  SIMCLG                                       5
//SIM.SYSIN  DD *                                       6
  <main program source>                                 7
/*                                                      8
//GO.SYSIN   DD *                                       9
  <test data>                                          10
/*                                                     11
```

Ex. 7: Compile and run external procedure and main program.

Statement 1 - 4 can be repeated if there is more than one external procedure.

```
//A  EXEC SIMCL,PROG=PROC1,LIB='A.LLIB',                1
//  PARM.SIM=EXTERN,PARM.LKED=NCAL                      2
//SIM.SYSIN  DD *                                       3
  <external procedure source>                           4
/*                                                      5
```

Ex. 8: Save an external procedure in a program library.

1: Add the operand LDISP=OLD if an old procedure is to be replaced.

2: The linkage-editor parameter NCAL prevents addition of run-time system elements. This will save space in the library. The operands must occur in this order since the SIM procstep precedes LKED in the catalogued procedure.

```
//A  EXEC SIMCLG,EXLIB='A.LLIB'          1
//SIM.SYSIN  DD *                        2
  <source program>
/*
//GO.SYSIN   DD *                        3
  <test data>
/*
```

Ex. 9: Compile and execute program using external procedures in a load module library.

The external procedures used must have been put in A.LLIB using the method of Example 8.

The name specified in the PROG operand of Example 8 must coincide with the identifier of the procedure and with the <external identifier> of the external procedure declaration in the main program.

2.4.1 Using the SIMULA system with the JCL procedure SIM

SIM is a one-step JCL procedure which makes use of a special program called SIMCNT. The basic action of SIMCNT is to retrieve input programs and process them, one after another.

There are four modes in which SIMCNT can operate:

```
source mode = input program is in source code
object mode = input program is an object module
load mode   = input program is a load module
update mode = input program is a source code which is to be updated
              prior to further processing.
```

The operating mode may be freely changed for any program.

Program processing in the source mode consists of compilation (using the SIMULA compiler), loading (using the OS LOADER) and execution. Once compiled and loaded, a program can be executed an arbitrary number of times. The above mentioned pattern of program processing is cut short if either compiler or loader actions were not successfull or if only compilation was requested; processing continues with the next program in such a case. In object mode, program processing consists solely of loading and subsequent execution (possibly multiple executions). In load mode, the program is loaded from secondary storage into core (using the LOAD macro) and then executed the specified number of times. In update mode, the source program is updated (using the IEBUPDTE utility) and the new master, which is a temporary, sequentially organised data set, is then treated as an ordinary input program in source mode unless updating was not successfull.

The function of SIMCNT and mode switching are controlled by a trivial command language, sentences of which are passed to SIMCNT via the SIM symbolic parameter P, which is positioned at the beginning of the EXEC PARM field. A sentence of the SIMCNT language is a string of arbitrary length consisting of the letters C, O, L, U, E and unsigned integers. An empty string is also legal and it is interpreted as digit 1. In addition, program names (terminated by commas, if necessary) may also be part of the control string.

The meanings of the respective symbols are as follows:

C ... switch to source mode:
retrieve and compile a source code program.

O ... switch to object mode:
retrieve, load and execute an object program.

L ... switch to load mode:
retrieve the program whose name follows the L (and is delimited by a comma if not terminating the string), and execute it.

U ... switch to update mode:
retrieve and update a source program, then compile the updated version.

E ... if in source or update mode:
(load if necessary and) execute the program which was compiled last, otherwise execute the last loaded program.

unsigned integer

... if followed by any of the above control letters, indicates repetition of the associated action (e.g. 2C means CC, 3E means EEE, etc.); if terminating the control string: perform CE the requested number of times (i.e. compile/load/execute the specified number of programs, e.g. digit 3 at the end of the control string stands for CECECE).

The physical location of programaams to be processed by SIMCNT depends on their processing mode: the program to be processed in load mode must be a member of a load library specified as or concatenated with STEPLIB; the program to be processed in update mode must be a (member of a partitioned) data set specified as (or concatenated with) SYSLIB; any other program must be a sequential data set with ddname SYSINn, where n is empty for the very first program to be processed or i-1 for the i-th program (regardless of mode witching). In update mode, SYSINn is used as a ddname for the IEBUPDTE control statement data set. Execution data (if any) must be submitted on the following ddnames:

DATA for the first execution of the first program (i.e. on SYSIN)

DATAj for the (j+1)-th execution of the first program (i.e. on SYSIN)

DATAij for the J-th execution of the i-th program (i.e. on SYSINi)

Notes:

- Execution data set ddname conflicts must be resolved by appropriate sequencing of the programs (e.g. DATA11 may be data for the 12th execution of the first program as well as for the first execution of the second).

- Syntax checking of the control string is minimal and the effect of illegal strings is unpredictable.

- The global condition code returned by SIMCNT is:
(max update/compilation cc)*100 + (max loading/execution cc) or 1111 in the case of the command language syntax error.

- Control of the processors involved (compiler, loader, RTS) may be exercised in the usual way, i.e. by specifying requested options in the EXEC PARM field. But remember that this affects all processed programs equally. (The EXEC PARM field to be used with SIMCNT may either be empty or of the following format:

  PARM='<SIMCNT control>/<SIMULA comp. parm.>/<loader parms.>/<RTS parms.>/<user parms.>'

  Any angle bracketed part may be empty, but only trailing slashes may be dropped.)

The complete set of control statements for procedure SIM as suggested in the SIMJCL data set of the release tape are given in appendix I.

Note that the basic set of DD statements used by SIMCNT is identical to that of a standard SIMULA compiler. Providing that SYSPRINT and SYSGO DD statements are not altered, SIMCNT operates equally well with any compiler version regardless of its default ddnames.

Additional data sets may also be used.

Setting of the RTS parameter TIME is a recommended safety precaution which will inhibit waste of CPU time if an infinite loop occurs in some executed program (the time unit used is 1/100 second).

Examples of SIMCNT activation using SIM:

```
//A      EXEC SIM
//SYSIN  DD   *
  <SIMULA program>
//DATA   DD   *
  <execution data>
/*
```

Ex. 10: Compile, load and execute one SIMULA program

```
//B      EXEC SIM,P=2,CP=XREF
//SYSIN  DD   *
  <1st program>
//DATA   DD   *
  <data for 1st program>
//SYSIN1 DD   *
  <2nd program>
//DATA11 DD   *
  <data for 2nd program>
/*
```

Ex. 11: Compile two source programs, taking cross-reference listing and execute them.

```
//C      EXEC SIM,P=COE
//SYSIN  DD   *
  <1st program (source)>
//SYSIN1 DD   *
  <object module of the 2nd program>
//DATA11 DD   *
  <data for the 1st exec of 1nd prg>
//DATA12 DD   *
  <data for the 2nd exec of 2nd prg>
/*
```

Ex. 12: Compile one program, then load and execute twice another program which is supplied in object module form

```
//D      EXEC SIM,P='LPROGRAMX,2EC'
//DATA   DD   *
  <data for 1st exec of PROGRAMX>
//DATA1  DD   *
  <data for 2nd exec of PROGRAMX>
//DATA2  DD   *
  <data for 3rd exec of PROGRAMX>
//SYSIN1 DD   *
  <source program>
/*
```

Ex. 13: Execute three times PROGRAMX which has previously been compiled and linked-edited (into SIMLIB), then compile a source program

```
//E      EXEC SIM,P=U
//SYSLIB DD DSN=OLDTEST,DISP=SHR
//SYSIN  DD *
./  CHANGE NAME=TEST,NEW=PS
 -- update data statements --
./  ENDUP
/*
```

Ex. 14: Update and compile a source program which is held under member name TEST in pds OLDTEST.

3 Hardware and implementation defined restrictions and capacity limitations.

3.1 Permitted ranges of arithmetic quantities, precision of real number arithmetic.

The ranges of arithmetic quantities are listed in table 3.1.

quantity	maximum	minimum
INTEGER	2**31 - 1	-2**31
SHORT INTEGER	2**15 - 1	-2**15
REAL *)	((1-16)**-6)*16**63	16**-65
LONG REAL *)	((1-16)**-14)*16**63	16**-65

Table 3.1: Ranges of arithmetic quantities.

*) The range of the magnitudes of normalized numbers are given. A true zero is also representable, and the range of the magnitudes of negative numbers is the same as for positive numbers.

Integer arithmetic is performed exactly.

REAL quantities have a precision of 6 hexadecimal digits (around 7 decimal digits), LONG REAL quantities have 14 hexadecimals (around 16 decimal digits). There may be an additional loss of precision due to the rules for truncation of floating point operation results (see (4)).

3.2 Maximum sizes of block instances, arrays and texts.

The maximum size of a block instance is 4096 bytes. The size of a block instance depends on its parameters and declared quantities and can be computed using the information in 3.4.

There is no maximum size of an array, but if the last subscript is ignored, the array must not have more than 2**15 - 1 elements.

A text must not have length greater than 2**15 - 20.

3.3 Program capacity limitations.

Besides the capacity limitations given in the table below there are restrictions on the complexity of an expression, for which no comprehensive estimate can be given. See also 3.2.

Item		max value
Block level	1)	15
Prefix level of class or prefixed block	2)	62
Number of parameters to a procedure or a class (including parameters of prefixes)		127
Number of parameters to Fortran or assembly procedure.		18
For statement nesting level		255
Designational expressions in switch declaration		127
Number of virtuals specified in class (including prefixes)		255
Number of subscripts of array		127
Identifiers declared in one list		127
Number of differently spelled identifiers, compiled in minimum core area		1000
Number of differently spelled identifiers, independent of core area	3)	3072
Redeclaration level capacity	4)	30
Number of fixups	5)	Dep. on part.size
Number of external references	6)	200

1) The block level for a program point is the number of begin-end pairs that enclose it, excluding those of compound statements, but including connection blocks.

2) A class with no prefix has prefix level zero, and a class with prefix has prefix level one higher than its prefix.

3) This is normally 1500 user identifiers as default. In order to obtain the maximum, the compiler CSECT DEFAULT must be modified and the compiler regenerated to increase tha table size.

4) Block level plus the number of enclosing compound statements that have local labels and are also controlled statements or connection blocks.

5) The number of fixups generated by program constructions are:

Item	Fixups
Class declaration, prefixed block	4
Other block	3
then	1
else	1
label	1
inspect	3
when	1
for statement	2
switch	1

3.4 Object program storage requirements.

The SIMULA object program needs storage of three categories:

i) Storage for compiled code (load module).
ii) SIMULA working storage.
iii) System working storage.

The amount needed for i) is fixed during program execution and can be determined from the linkage-editor listing when the object program is created.

The amount of system working storage needed can be determined from (2).

The amount of SIMULA working storage needed is the sum of the storage needed for all referable block incarnations in the program, and a general overhead of 368 bytes (which includes the images of sysin and sysout).

Except for space for declared variables a block takes:

24 bytes if it is not terminated, or if it is an object of a class with local class attributes

24 bytes if it is a scheduled process

24 bytes if it is a procedure made visible through connection or remote referencing.

The block instance itself has an overhead of 8 bytes and the storage needed for parameters and declared quantities can be determined from table 3.2, which gives the number of bytes and the alignment factor for any quantity. Quantities are allocated in the order in which they are declared, then 4 bytes are added for each nesting level of for statements.

*. mode *. quant *.	declared or by value		parm.by reference		parm. by name	
	S	A	S	A	S	A
PROCEDURE LABEL,SWITCH	0	0	8	4	8	4
REAL	4	4	-	-	8	4
LONG REAL	8	8	-	-	8	4
INTEGER	4	4	-	-	8	4
SHORT INTEGER	2	2	-	-	8	4
BOOLEAN CHARACTER	1	1	-	-	8	4
REF	4	4	4	4	12	4
TEXT	12	4	12	4	8	4
REF ARRAY	4	4	4	4	12	4
array:s	4	4	4	4	8	4

Table 3.2 Space for quant:s in block instance .

S : size (bytes) for quantity.
A : alignment of quantity.

For a <type> procedure block instance, the result takes the space of a declared quantity of the same type.

Arrays and text blocks are allocated outside the block instance. An array needs the space required for all its elements according to table 3.2, and an overhead of 28+2*n bytes, where n is the number of subscripts of the array. A text block needs 12+n bytes, where n is the length of the main text.

The sizes of block instances, arrays and texts are always rounded upwards to a multiple of 8.

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:

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

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

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 HISTO(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;
              HISTO := T;
        END OF HISTO;
```

```
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.

5 Object program input/output.

Input/output (I/O) is the part of SIMULA used for communication. The mode of communication is:

i) user to program
ii) program to user
iii) program to program

An example of i) is the reading of the source program from the user.

ii) is exemplified by printing on a line printer.

Examples of iii) are the writing of a tape to be read by another program or the same program, and the writing and reading of blocks selected at random on a direct-access device.

In the following subsections the basic principles of Data Management with respect to SIMULA I/O are described. Subsections marked with an asterisk cover special features requiring some background knowledge not given in this manual. The references in the heading of these sections refer to the IBM 360/370 documentation that should be understood before reading the section.

The following features are not described at all since they are not related to the fact that the data sets are processed by SIMULA programs:

Operations on data sets by Utility Programs	(see (9))
Concatenation of data sets	(see (6))
Multivolume data sets	(see (6))
Using members of partitioned data sets as sequential data sets	(see (6))
Details of SPACE allocation	(see (6))
Password data sets	(see (6))
Generation data sets	(see (6), (9))

5.1 OS 360/370 data management terminology.

The pieces of equipment which do the physical reading or writing on the external data carrier are called devices or units (peripherals with non-IBM terminology). The word device usually refers to the kind of equipment (e.g. "a 2311 device") whereas unit refers to a particular piece ("this tape unit"). The distinction is, however, not always maintained and is rather unimportant. A detachable data carrier is called a volume (tape reel, a disc pack or even a drum). Data on volumes are read and written in blocks, which are physically recognized by the units. A block will always correspond to an integral number of bytes.

Devices with a fixed block length are called unit record devices (card reader/punch, line printer).

An important function of data management is to maintain a logical structuring of data, which may be different from the physical structuring in volumes, blocks and bytes.

The largest logical data unit is a data set. Several data sets may reside on one volume, but a data set may also occupy several volumes. A data set is uniquely defined by its data set name and the volume on which it starts. For retrieval and checking purposes, the system requires that all volumes have volume labels and that all data sets have data set labels (except for magnetic tapes, which need not have labels, in which case the system assumes that the operator has mounted the correct volume).

The data set label contains a description of the data set characteristics, such as data set organization, logical record format, logical record length, etc. Simula programs process data sets with sequential, direct or partitioned organization. In a data set with sequential organization the records are retrieved in sequence while in a direct data set the records can be retrieved in a random order. A partitioned data set can be regarded as a collection (library) of similar sequential data sets (members).

5.2 Principles of OS 360/370 data management.

Since the SIMULA object program runs under the operating system OS 360/370, it must follow the rules of Data Management of this system, and it also has access to the facilities provided by the system. The most important facilities are device independence of the source program and run-time specification of I/O processing parameters. The most important rule is that each file must be defined by a data definition statement (DD-statement), which logically connects the file to a data set at run-time.

5.2.1 Device independence restrictions.

Device independence means that the source program does not refer to actual devices, so it can be different devices without recompilation, depending on availability. The device characteristics must, however, be compatible with the functions requested by the program. The allowed devices for the different file classes are given in table 5.1. Infiles, outfiles and printfiles process sequential or partitioned data sets, whereas directfiles process direct data sets.

*. file / device *.	infile	outfile	printfile	directfile
card reader	X	I	I	I
line printer	I	(X)	X	I
disk	X	X	X	X
drum	X	X	X	X
tape unit	X	X	X	I
card punch	I	X	(X)	I

Table 5.1 Device-file class correspondence.

I: illegal combination.

X: legal combination.

(X): legal but not recommended.

5.2.2 Run-time specification of I/O processing parameters.

A SIMULA source program will, in general, not contain detailed information on the processing mode for a file. OS 360/370 allows a large number of alternatives for such things as record format, record length, block size, number of buffers, buffer length, recording density (tape), etc. to be specified at run-time by means of the DD-statement. Some of these parameters are permanent characteristics of the file (data set), and they need only be specified in the job step that creates it. When the data set is used later, the characteristics will be obtained from the data set label by the control program (this does not apply to magnetic tapes without standard labels).

5.3 Data definition statement (DD-statement).

The DD-statement serves the following purposes:

i) It defines the correspondance between the program unique filename and the system unique file identification.

ii) It supplies additional information needed to find an existing data set, when this is necessary.

iii) It tells the system how to handle the data set when the job step is finished.

iv) It defines whether write operations on a sequential data set are going to start at the beginning or at the end of the data set, that is whether the program will add to the data set or replace it (outfile).

v) It indicates where a new data set is to be created and how much space it will occupy.

vi) It defines data set characteristics for a new data set.

vii) It can be used to specify special processing options for a new or old data set.

The items above are defined by a number of keyword parameters. The keyword parameter DCB= has a large number of subparameters used for purposes ii), vi), and vii).

The keywords relevant to purposes i) - vi) above are:

i) : DSNAME=

ii) : UNIT=, VOLUME=, LABEL=
DCB subparameters DEN=, TRTCH=

iii) : DISP=

iv) : DISP=

v) : UNIT=, VOLUME=, LABEL=, SPACE=, SYSOUT=

vi) : LABEL=, DCB subparameters DEN=, DSORG=, RECFM=, LRECL=, TRTCH=, BLKSIZE=

vii) : DCB subparameters BUFNO=, EROPT=, HIARCHY=, OPTCD=, MODE=, STACK=

5.3.1 Binding a file to a data set.

Each file object of a SIMULA program will, at execution time, be connected to a data set. The binding is effected by a DD-statement, which defines a data set and a DD-name, and which is supplied to the execution job step. When the file object is created it is given a DD-name which is the FILENAME parameter of the object. The DD-name should be a valid SIMULA identifier, possibly with trailing blanks. 8 characters are significant. When the file is opened it is logically connected to the data set defined by the DD-statement with a matching DD-name. If no such DD-statement exists, the SIMULA program is terminated with a diagnostic. Subsequent calls of OUTIMAGE/ INIMAGE will cause records to be written/read on this data set. Several files may refer to the same data set, i.e. an infile may read data written by an outfile earlier in the program. This can be achieved by either giving the same DD-name to the files or defining the same data set in several DD-statements. The files should not, however, be open simultaneously.

This also goes for different members of the same partitioned. Use of the same dataset in different DD-names in parallell will give unpredictable results at run-time.

> Note: If the catalogued procedures described in 2.4 are used to execute an object program, the DD-name should be preceded by 'GO', indicating that DD-statement is added to the GO step of the procedure.

5.3.2 Creating a data set on magnetic tape, disk or drum.

5.3.2.1 Naming a data set.

A data set which will be used in more than one job step must be given a name when it is created. The name is supplied by the DSNAME= parameter. The name may be simple (e.g. LLIB) or qualified (A.B.LLIB). If the data set will be referred to in other jobs, it is preferably catalogued (5.3.6). In order to catalogue a data set with a qualified name 'qual.sequence.name', the qual.sequence must be the name of an index in the catalogue. Indexes are built with the IEHPROGM utility program (see (9)).

If the data set is only used in later steps of the job, it can be assigned a temporary name of the form &&name, where name is a simple name. Such data sets should be passed (5.3.6) to the later job steps of the job.

5.3.2.2 Allocating a data set.

When a data set is created, a volume or volume class on which it is to be allocated must be indicated. There are two ways to specify the volume: by specific or nonspecific volume request.

Specific request

A permanent data set should be allocated by a specific request identifying the volume on which the data set is to be placed. A specific request is effected by giving either

i) the device number and the volume serial number, e.g.

UNIT=3330,VOLUME=SER=111111,

when you want to put the data set on the 3330 disc pack with several number 111111, or

ii) the name of a data set catalogued on the same volume, e.g.

VOLUME=REF=MYLIB,

where MYLIB is the data set name of a data set catalogued on the volume.

Nonspecific request

For temporary data sets it is usually enough to make a nonspecific request, in which case the system chooses a suitable volume depending on general specifications you supply by means of the UNIT=parameter. In each installation a number of unit group names can be used to specify the type of volume. Common names are SYSSQ for tape or disk, SYSDA for disk or drum, and DRUM for drum.

In applications with several files, a considerable increase of performance can be achieved by carefully considering the times at which each of the data sets are processed, and requesting separate access mechanisms for files operated at the same time.

As an example, consider a program that operates in two passes. The first pass reads a catalogued data set and produces intermediate results on a temporary sequential file. The second pass reads the intermediate results and produces a new data set. Since the input and output data sets are not used in the same pass, they can use the same access mechanism, but the temporary data set should be separated from both the input data set and the output data set:

```
//STEP1   EXEC  PGM=TWOPASS
//SYSOUT  DD  SYSOUT=A
//SYSIN   DD  DSNAME=INPUT,DISP=OLD
//OUTSET  DD  UNIT=AFF=SYSIN,DISP=(NEW,PASS),
//            SPACE=(...),DSNAME=&&OUTSET
//TEMPSET DD  UNIT=(SYSDA,SEP=(SYSIN,OUTSET)),
//            SPACE=(...)
```

The DD-names of the three files are SYSIN, OUTSET and TEMPSET.

The DD-statement with DD-name SYSOUT is mandatory, and this data set will contain possible diagnostics. The UNIT= parameter of OUTSET specifies affinity with SYSIN, i.e. this data set will be allocated on the same volum as INPUT. The UNIT= parameter of TEMPSET indicates separation from OUTSET and SYSIN and direct-access storage (SYSDA). The separation request will be ignored if it cannot be satisfied.

The SIMULA program may look like the following, with the detailed processing of data removed:

```
BEGIN           REF(OUTFILE) TEMPOUT,OUTPUT;
                REF(INFILE) TEMPIN;
                ...

PASS1:          TEMPOUT :- NEW OUTFILE ("TEMPSET");
                TEMPOUT.OPEN(BLANKS(80));
                ...

PASS1LOOP:      INIMAGE; ... TEMPOUT.OUTIMAGE;
                GOTO PASS1LOOP;

PASS2:          TEMPOUT.CLOSE; TEMPOUT :- NONE;
                SYSIN.CLOSE;
                OUTPUT :- NEW OUTFILE("OUTSET");
                TEMPIN :- NEW INFILE("TEMPSET");
                TEMPIN.OPEN(BLANKS(80));
                OUTPUT.OPEN(BLANKS(80));

PASS2LOOP:      TEMPIN.INIMAGE; ... OUTPUT.OUTIMAGE;
                GOTO PASS2LOOP;

ENDPROG:        TEMPIN.CLOSE; OUTPUT.CLOSE;
                OUTTEXT("TWOPASS FINISHED");
END TWOPASS;
```

Note: The performance can be increased significantly by specifying proper blocking of OUTSET and TEMPSET (5.3.7). The SPACE requests will be discussed in the next section. Unit affinity can only be requested for removable volumes.

5.3.2.3 Direct-access storage.

Except for the volume or volume type, one must specify the space a data set will occupy. A data set on a direct access device may occupy several disjoint areas, called extents.

The first extents are allocated when the data set is created, and they constitute the prime area of the data set. Additional extents are allocated when all previous extents are filled by a process called secondary allocation. Secondary allocation will occur up to 15 times, but after that the data set must be restructured.

The format of the space parameter is

```
SPACE=(units,quantity)
SPACE=(units,(quantity,increment))
SPACE=(units,(quantity,increment),,,ROUND)
SPACE=(units,quantity,,,ROUND)
```

unit is the unit in which the space request is given. It could be:

```
  i)  average block length in bytes
 ii)  TRK:  tracks
iii)  CYL:  cylinders
```

quantity is the number of units to be allocated as prime area.

increment is the number of units allocated to each additional extent.

ROUND this indicates that the control program will round the size of each extent upwards to an integral number of cylinders. Extents are also allocated on cylinder boundaries.

Notes: Specify units with average block length if the volume request is non-specific, since it may be satisfied by devices with different track capacity.
ROUND can increase performance (see (9), p. 49). There are several alternate space allocation methods, which are given in (see (9), p. 47).
The track and cylinder capacities are listed in (see (7), p. 158).

If the data set is going to have direct organization (directfile), you must also specify DCB=DSORG=DA in the DD-statement. No other DCB subparameters may be specified. A direct data set will never use secondary allocation, so the increment is superfluous.

5.3.2.4 Magnetic tape.

On a magnetic tape any SPACE parameter in the DD-statement is ignored. Instead, the data set serial number in the LABEL= parameter must be specified. The format of this parameter is

```
LABEL=(n,SL),
```

where n is the ordinal number of the data set on the reel, and SL indicates that the data set has standard labels.

A tape volum which is going to have standard labels must be initialized with the IEHINITT utility program (see (9)).

Tapes with no labels should be used only if they are used or produced outside the System 360/370 OS environment.

Note: When a data set with serial number n is written, all data sets on the volume with serial number >n are lost.

5.3.3 System output.

When a data set is created on a line printer or a card punch, it will logically leave the system, since the system does not recognize labels on the corresponding data carriers (printer listings and punched card files). Therefore, the DD-statement will not contain a DSNAME= parameter or a volume request, but the kind of output is specified with the SYSOUT= parameter. The parameter is a letter or a digit, usually A for printed output and B for punched output. In some systems (MFT or MVT) the SPACE= parameter can be used to limit the number of lines if the program is looping. An example of the use of SYSOUT=A is found in section 5.3.2.2.

5.3.4 Retrieving a data set.

A data set is retrieved by the system if it is given the data set name and the volume on which it resides. The volume information can be given explicitly in the DD-statement in the form of a specific volume request, or is implicit if the data set is catalogued or passed from a preceding job step of the job. The data set name is given in the DSNAME= parameter. In addition, the DISP= parameter must indicate that the data set is to be retrieved (5.3.6).

For a magnetic tape you must give the density, which is needed for reading the label, e.g. DCB=DEN=1 and the data set serial number in the LABEL operand.

5.3.5 System input.

When an input data set is small, it is conveniently put in the job input stream among the control statements. The DD-statement for the data set has the following format:

```
//ddname  DD  *
```

and indicates that the data set follows this line in the input stream.

In a PCP system, this must be the last DD-statement of the job step, and only one such data set can be processed in any single job step.

The line images may not have // or /* in the first two columns.

If you need to input datasets containing e.g. // in the first two positions the DD-statements to be used is

```
//ddname  DD  DATA
```

The end of the data set is signalled with a line containing /* in the first two columns.

5.3.6 Disposition of a data set.

The DISP= parameter serves several purposes:

i) it indicates whether a data set is to be created or retrieved

ii) it indicates whether it is to be catalogued, passed, kept or deleted

iii) it defines positioning of sequential data sets.

The format of the DISP= parameter is:

```
DISP=(parm1,parm2), or DISP=parm1,
```

where parm1 is OLD, SHR, MOD or NEW, and parm2 is CATLG, KEEP, PASS, DELETE or is absent.

If the DISP= parameter is missing, DISP=(NEW,DELETE) is assumed, and DSNAME is not necessary.

parm1:	meaning:
OLD	An existing data set is to be retrieved. The data set will be locked, so that no other tasks in a multiprogramming environment can access it during the job step. If the data set has sequential organization, it will be positioned to the beginning, so that read operations will read the entire data set and write operations will replace the existing records.

SHR — This parameter has the same meaning as OLD, except that other tasks in the system may access the data set concurrently. Do not specify SHR if the job step writes on the data set, unless the write operations are synchronized by ENQ/DEQ macro instructions (these can be issued in an external assembly procedure of a SIMULA program, (see (7))).

MOD — The system will check if a data set with the same name has been passed to the step. If this is not the case, or if the system cannot find the volume information for the dataset, a new data set will be created (and the DD-statement must contain enough information to create it). Otherwise the existing data set is used, and it is positioned to the last record, so that write operations will add records to the data set.

NEW — A new output data set is to be created. Write operations will add records, starting from the beginning of the data set.

parm2: meaning

CATLG — The data set will be catalogued at the end of the job step. It can later be retrieved by name alone.

KEEP — The data set will be kept after the job step. It must later be retrieved with a specific volume request.

A sequential data set (infile, outfile) will be rewound after each close, and a later opening of the same data set (not necessarily the same file) will process it from the beginning.

PASS — The data set is passed to the next job step that uses it. The next job step will retrieve it by name, if MOD or OLD is specified in the parm1 field. A sequential data set will not be rewound when closed, and if it is reopened, records can be added to it.

DELETE — The data set is deleted at the end of the job step. It is rewound when closed.

A third subparameter can also be used. This will specify what is to be done with the dataset if the job step obnormally terminates (See (6)).

5.3.7 Sequential data sets: characteristics and processing options (outfile, infile).

A sequential data set will have several characteristics which are determined from the DD-statement when the data set is created.

These are record format, logical record length, block size, and, for data sets on magnetic tape, recording density and tape recording technique. They are determined from DCB subparameters RECFM=, LRECL=, BLKSIZE=, DEN= and TRTCH=, respectively.

5.3.7.1 Fixed record length.

If characteristics are not specified the data set will have fixed length and unblocked records, i.e. each time OUTIMAGE or INIMAGE is called, a block of fixed size is written or read. The length of all blocks and records in the data set will be the same as the length of the image passed as a parameter to OPEN. When the data set is connected to an outfile, the image length of the latter must always be less than or equal to the record length. When it is connected to an infile, its image length must be greater than or equal to the record length. In both cases some CPU time and core storage is saved by having image length equal to the record length of the data set.

5.3.7.2 Blocked records.

A data set with fixed length and unblocked records provides the fastest processing of blocks with a given length. It will, however, often be the case that the logical units of information (customers, employees, projects, loads, charges, measurement points, etc.) require much less space than the optimum block size for the device. In these cases one can maintain logical clarity of the source program and still use optimal size of blocking the records (images) together. A blocked data set is obtained by specifying RECFM=FB,LRECL=image length, BLKSIZE=block size in the DCB parameter of the DD-statement. The block size must be a multiple of the image length. It must in no case exceed the track capasity of the device of 32760.

Example: 80 byte images blocked 20/block:

```
//BLOCKSET  DD  DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600),...
```

5.3.7.3 Variable and undefined record formats. (see (7), p. 58).

If there are considerable variations in the amount of information per image, space can be conserved on secondary storage by writing variable (V or VB) or undefined (U) format records. The length of image determines the length of each record when written.

On input, the image must contain the record if variable format is used, whereas for U format data sets the image length determines the number of bytes read. The LRECL and BLKSIZE parameters are filled in as shown in Table 5.2

RECFM=	LRECL=	BLKSIZE=
V	max+4	-
VB	max+4	> max + 8
U	-	max
F	max	-
FB	max	n*max

Table 5.2: LRECL and BLKSIZE parameters

max is the maximum length of image.

BLISIZE= may never exceed the track capasity or 32760, whichever is smallest.

5.3.7.4 Carriage control character. (see (7), p.60)

A carriage control or stacker bin selector character may be the first character of each record of a sequential data set. This is indicated by appending A (ASA control character) or M (device dependent control character) to the RECFM= code. This character must be supplied in the first position of image. It will be written together with the rest of the image on secondary storage (disc, drum, tape). It will not appear on a printer listing or on punched cards.

5.3.7.5 Additional DCB subparameters. (see (6) and (7))

The following additional subparameters can be used in special applications (none is compulsory):

BUFNO= Number of buffers. If omitted, two buffers are allocated, which is the most suitable in most cases. If there is lack of storage BUFNO=1 is specified. If chained scheduling is used it is advantageous to have many buffers (always less than 255).

DEN= Tape recording density (0, 1, 2, 3 or 4) 2 (800 bpi) is assumed if omitted.

ERDPT= Action taken when an erroneous block is read (I/O error):

ACC : accept block. The image should be investigated character by character to avoid termination in number de-editing procedures.

SKP : Skip the erroneous block.

not defined : the program is terminated.

In any case a message is printed on sysout.

HIARCHY= Hierarchy of buffer storage in systems with Large Capacity Storage support (0 or 1). 0 (high speed core) is assumed if omitted.

MODE= Mode of cared reader or card punch (e.g. E indicating EBCDIC).

OPTCD= Special services requested

C: Chained scheduling. Several I/O requests are grouped together and performed in one channel operation. This is of advantage if the data set has been given a small blocksize because of the core requirements of one program and is also processed by a program with small core requirements. Specify channel separation and many buffers for better performance.

Q: Conversion EBCDIC/ASCII. The conversion is from EBCDIC to ASCII on output, and the opposite on input. Note that the maximum block size which can be used is 1600. Note also that only non-labeled tapes can be handled.

STACK= Stacker bin selection on a card punch (1 or 2).

TRTCH= Magnetic tape recording technique (C,E,T). Note that the parameters TRTCH, STACK and MODE are mutually exclusive.

(See (6) for further details)

5.3.8 Printfile data sets.

All printfile data sets have variable length and blocked records with ASA control character.

The control character is inserted automatically and is controlled by standard procedures (lines per page, spacing, eject).

Performance can be increased by giving

LRECL = (maximum image length + 5) and

BLKSIZE = (>=maximum image length + 9)

The other DCB parameters (except RECFM) of section 5.3.7 can be specified, but most of them will usually not be applicable.

5.3.9 Direct data sets.

A direct data set will consist of a number of fixed length blocks. The block length will be equal to the image length when the file is first opened. The first time the data set is opened, it is initialized by consecutively writing the open text parameter until the prime area is filled. No DCB subparameters except DSORG=DA may be specified, and this must be specified when the data set is created. The image length must at all times be equal to the block size of the data set.

The access method is BDAM. Addressing is by relative block number.

When INIMAGE or OUTIMAGE is called and LOC is out of range, the end of file text is supplied if INIMAGE was called, but image is unaltered if OUTIMAGE was called. The Boolean procedure ENDFILE indicates whether the last i/o call on the file was successful (FALSE) or not (TRUE). ENDFILE is reset after a successfull i/o operation. Two unsuccessful calls may not occur in sequence.

Keyword	infile	outfile	printfile	directfile
BLKSIZE=	1)	2)	max.blocksize	
BUFNO=	3)	3)	3)	
DEN=	0,1,2,3,4(tape)	0,1,2,3,4,(tape)		
DSORG=				
EROPT=	ABE,SKP,ACC 5)			
HIARCHY=	0, 1 9)	0, 1 9)	0, 1 9)	0, 1 9)
LRECL=	1)	6)	7)	
MODE=				
OPTCD=				
RECFM=	1)			
STACK=	1 or 2 8)	1 or 2 8)		
TRTCH=	C,E,T (tape)	C,E,T (tape)		

Table 5.3: DCB subparameters.

1) must be specified for tape without standard labels if the default values were overriden when the data set was created.

2) for V and U formats, a maximum blocksize can be specified when the data set is created.

3) The system default values (two buffers) may be overriden.

4) must be specified.

5) Defines system action on bad blocks.

   ACC: accept the bad record

   SKP: skip the bad records

   ABE: terminate processing

6) U format: do not specify LRECL
   V format: maximum length of image + 4
   F format: length of each record, can be defined when data set is created.

7) maximum length of image + 5.

8) card punch.

9) 0 is default.

5.4 End-of-file text.

When all records of a sequential input data set have been read, an end of file condition exists. On the next call of inimage, the Boolean variable endfile (sensed by the Boolean procedure ENDFILE) is set and the end of file text is stored in image. The end of file text consists of /* in positions 1 and 2 and of blanks in the other position. If the length of image is greater than 260 bytes, the character after the 260th are not blanked.

Be careful with making programs depend on the EOF-record. The EOF-record definition in SIMULA Common Base is the text value "!25!". Programs making use of this value will not be compatible with other SIMULA systems.

5.5 Sysin and sysout.

The standard files sysin and sysout are defined with the ddnames 'SYSIN' and 'SYSOUT', respectively. DD-statements for these files must always be supplied. If a program does not use sequential input, sysin can be defined as a dummy data set by the statement

```
//SYSIN  DD  DUMMY
```

and core storage can be released by closing it at once.

The standard output file sysout will be used to produce any run-time error or warning diagnostics. If sysout is closed, these diagnostics will appear on the console, but if SYSOUT, OUTIMAGE is called, the program is terminated.

If sysout is closed, a dump or trace should not be requested since the console typewriter is fairly slow and is not supposed to be used for such purposes.

6 Debugging aids.

The following utility functions are available for program debugging at compile- or run-time:

- diagnostic messages from the compiler,
- diagnostic messages at execution time,
- identifier cross-reference table from the compiler, see 2.2.1.1.
- manipulation of the source listing by improving layout, see 2.2.1.1.
- program control and data flow tracing,
- symbolic dump of the environment at a run-time error,
- formatted storage dumps,
- assembly listing of the object code.

The two latter are normally only used in connection with system maintenance, but users who connect to assembly procedures may think of these functions as useful.

6.1 Diagnostics.

Messages may be issued by all system components involved in the use of SIMULA. These messages will appear on the printed output from the job, or on the console listing. Each message is identified by a three-letter prefix identifying the system component and a three-digit message number within the component in the first six positions of the printed line. An explanatory text will follow the message identification.

Messages are of three kinds:

i) Information messages - requiring no response from the user.

ii) Warning messages - should be considered, but no response necessary.

iii) Error messages - requiring correction by user.

The most common messages are issued by the Job Scheduler (IEF), the SIMULA Compiler (SIM), the Linkage Editor (IEW), the SIMULA object program (ZYQ) and the Fortran diagnostic package (IHC, in external Fortran procedures).

If a program is aborted by the control program, a system completion code and an optional abnormal termination dump will be printed.

Completion codes and messages issued by IBM-supplied system components are listed in (see (11)).

Compiler and object program messages are explained in Appendices B and C.

6.2 Tracing.

6.2.1 Program control flow tracing.

The program control flow tracing has been available in the IBM SIMULA System from release 07.00, while the data flow has been available from release 08.00.

General description

- The system reports all events causing program control to change its sequential flow; the control switching is indicated in the form:

  aaaa*bbbb: mm...m

  where aaaa is the line number at which the sequential flow was interrupted, bbbb is the line number at which control continues and mm...m is the message giving the explanation.

- There are altogether 22 different events that can be reported. In order to distinguish tracing messages in the listing from program output they are embedded in dots.

- Either implicit or explicit tracing can be required (or both). The implicit tracing will cause the specified number of tracing messages to be output in case of a program error describing the corresponding number of events occurring before the error occurred. The explicit tracing can be achieved during program execution via utility procedures TRACE, TRACEON etc.

Commencing with release 08.00 of the IBM SIMULA there exists a possibility for tracing the flow of data during the program execution. It is the tracing of events affecting the contents of the user defined data structures that is covered by this facility (i.e. alterations of data structures predefined in the system are not traced).

All kinds of data transmissions are monitored whether they are expressed in the program explicitly as assignment statements or not (e.g. parameter transmissions, text attributes for editing of numerals etc.).

Means of control

- The control of tracing resides mostly with the RTS. The following are the only traces whose presence is dependent on compiler cooperation (SYMBDUMP must be greater than zero):

  - jump from one prefix level to another.
  - jump to first statement within a block.
  - jump to a local label.

- The number of tracing messages required in case of error is to be specified using TRACE option of the runtime system. (See 2.2.3.1.) This activates the tracing facility which then remains in action behind the scene during the whole program execution. Be aware of the fact that this degrades program efficiency. For production programs the normal efficiency is regained simply by specifying TRACE=0 as the RT option which is normally a default anyway (check your installation).

  A positive value n, if specified, has as a consequence the allocation of a buffer with n entries i.e. up to n tracing messages will be output describing events immediately preceding a RT-error. The disadvantage of this mechanism is that maintenance of the buffer alone (i.e. disregarding overhead due to its final interpretation) deteriorates program execution considerably (30% - 50% cpu-time can be used for this activity).

  Therefore it is much more economical to initiate the tracing activity dynamically at a suitable instant prior to the RT-error. To this end one can specify a negative integer as a TRACE parameter value. If value -n is specified the tracing activity is started first after n assignments were carried out in the course of the program execution. Consequently one must know how many assignments were executed at the instant when RT-error occurred. This information is given from now on at the RTS trailing line. Obviously, using this method program must be rerurn in case of a RT-error.

  Finally, one can also use the TRACE option to specify a fill character which will be used for embedding the tracing message to distinguish them from normal program output. (The default value is a dot character). This is convenient in case that e.g. a default character is not sufficiently represented on the pointer chain, or to improve the outlook of tracing messages (underscore may be quite handy) or simply to shorten the messages to fit on one line on a screen (use blank as the fill character). All that is required for changing the default fill character is to indicate the suitable character right behind the TRACE parameter numeric value or next to the equal sign if no numeric value is specified, e.g.

  TRACE=100_ or TRACE=!

- It is recommended that only selected events are traced (e.g. simulation events etc.); for this as well as the total list of traced events see documentation of the tracing utilities.

The necessary conditions for obtaining data flow tracing message are:

- compilation of the program (module) with the option SYMBDUMP=4.

- activating the tracer at execution either through specifying a non-zero value for the RTS TRACE option (see Updates for tracing control in release 08.00) or explicitly using the tracing utility TRACE.

Since assignments are often the most frequently executed statements in a program it is arranged so that a particular assignment is traced only limited number of times (5 times in default) and further monitoring is suppressed automatically. It is possible however to obtain further traces by lifting this limit through use of the tracing utility TRACECNT.

Finally note that the data flow tracing can be switched on and off using the tracing utilities TRACEON and TRACEOFF through the message number 0 used as the parameter value (it is on in default when tracing commences unless explicitly suppressed).

Format of messages

All data transactions are illustrated in a form of a synthetized assignment statement. The left hand side is a unique identification (throughout the whole program) of the location being altered while the right hand side is the actual value being assigned. The left hand side usually takes the form of a (subscripted) variable, its declaration block being identified at the right from the actual assignment image in a separate column. Exception is a text value assignment where the left hand side identifies the modified text object directly through its counter. (Note that TEXT(1) stands for the standard SYSOUT image and TEXT(2) for the standard SYSIN image.)

Finally one should note that each message of course identifies the line number in the source text where the monitored event is expressed and to which of its successive executions the message is related (the very first number on the line followed by a star).

Control procedures for the tracing facility.

The following utility procedures are available for control of the tracing facility on the IBM 360/370 SIMULA. The routines have effect both for the program control-flow and data-flow tracing.

1) TRACE

Motivation: It should be possible to start output of tracing messages from any point in the program.

Description: The procedure can optionaly have one parameter. Depending on whether it is used or not and whether it is positive or negative, we have the following cases:

- No parameter:
  Tracing is wanted from now on.

- Positive parameter:
  Tracing is wanted from now on, but only given the number of messages are wanted. If a RT-parameter TRACE was specified, accumulation of messages in the buffer continues afterwards. Otherwise the effect is that of NOTRACE after the specified number of messages is output.

- Negative parameter:
  The corresponding number of tracing messages present in the tracing buffer is wanted. This possibility demands that the RT-parameter for trace was specified. The buffer is emptied after dumping.

2) NOTRACE

Motivation: It should be possible to stop listing of tracing messages at any point in the program. The execution of the following part of the program should not be affected by the fact that we have been tracing a previous part of the program.

Description: The procedure takes no parameters. If the RT-parameter for tracing was specified, the accumulation of trace messages continues in the buffer of course. Otherwise all links to the tracing routines are removed, and the program efficiency will hereafter be as if the program had not been traced at all.

3) TRACEOFF

Motivation: The number of different trace messages is relatively high. For a specific application it might be desirable to use only some of the messages. Therefore the messages can be turned off individually.

Description: The procedure can take up to 18 integer/short integer parameters. These will refer to a message number. The specified tracing message will then no longer be listed, nor collected in the tracing buffer if the RT-parameter was set. If the value 0 is used as parameter, the dataflow tracing will be suppressed. If the procedure is called without any parameters, then all tracing messages are suppressed.

4) TRACEON

Motivation: For a limited part of the program it might be desirable to have a more complete trace of the control flow, without getting an endless tracing listing. Therefore it should be possible to turn the messages on and off individually.

Description: The parameters are specified in the same form as under TRACEOFF. If the value 0 is used, the dataflow tracing will be activated.

5) TRACETXT

Motivation: For some applications the tracing facility may be more than a debugging tool. It is also a good general guide to the description of control flow in SIMULA programs. For educational purposes it might be desired to alter the text in some of the messages. Some may also wish to change the text to get a better overview of the messages from the tracing for their special problem.

Description: The procedure takes two parameters. The first one is an integer or short integer referring to a message number. The second one is a text, specifying the contents of the new message. In some of the messages there is incorporated variable information gathered under the flow of program execution. It should be specified where in the new text this information should occur. The three characters &, % and $ are used to specify where object 1, object 2 and time is to be placed, respectively. If any of these are missing in a message which needs specification, it is assumed that the information is not required. However, once dropped this information cannot be anymore obtained in laber tracing messages.

A list of the default messages with placement of object 1, object 2 and time is appended.

In any of the procedure is used in conflict with its specification, it will have no effect.

TRACECNT

function: TRACE facility control

declaration: external asssembly integer procedure TRACECNT

parameters: two optional parameters which must be simple variables or literals of type (short) integer.

The first parameter (if present) specifies how many times (from now on) any assignment statement is to be traced (default=5). The second parameter (if any) specifies at which column of the tracing line the declaration block/ current process is to be monitored (default=100).

Note that the skipping of the first parameter can be achieved by specifying a negative value at its place, also that the permitted range of the second parameter values is <0,109>, the value zero disabling the output of the declaration block/current process monitor altogether.

result: the integer value yielding the number of the so far recorded assignment statements.

note: the returned value may be conveniently used for initiating/terminating dynamically various events e.g. debugging printouts. Note though that the assignment counting is carried out only if the program was compiled with SYMBDUMP>3.

Tracing messages:

Nr.:	Text:
1	CALLING &
2	GENERATING &
3	EXIT FROM &
4	DETACHING &
5	RESUMING &
6	ATTACHING &
7	EXPLICIT GOTO STATEMENT
8	NAME PARAMETER OR SWITCH THUNK
9	THUNK EVALUATION COMPLETED
10	LEAVING PREFIX &
11	STARTING BLOCK PREFIX
12	JUMP TO FIRST STATEMENT
13	& TERMINATED, % BECOMES CURRENT
14	& PASSIVATED, % BECOMES CURRENT
15	& CANCELLED
16	& SUSPENDED DUE TO ACTIVATION OF %
17	& SCHEDULED FOR TIME $, % CONTINUES
18	& SCHEDULED FOR TIME %, $ CONTINUES *
19	& SCHEDULED BEFORE %, $ CONTINUES *
20	SIMULATION CLOCK IS ADVANCED
21	& REMOVED FORM A SET
22	& PUT AFTER %

* For message nr. 18 and 19, three objects should be specified. Here the character $ is borrowed to indicate the third object.

OBS: Message no 0 (zero) is used for the data flow tracing.

6.3 The symbolic dump facility.

Commencing with release 06.00 the 360/370 SIMULA System has a provision for taking symbolic dump of the program under execution either at specified points or when an execution error occurs. This facility is provided in addition to the earlier formatted hexadecimal dumps.

6.3.1 Design principles.

Although the full implementation of this facility requires an extensive cooperation of many compiler and RT modules, the bulk of the work is done in the RT-routine ZYQDEBUG. By this separation it was possible to design the debugging system so that

- the incurred overhead in the execution time is negligible unless the dump is really produced, in which case the increase in the execution time is reasonably proportional to the amount of the dump.

- the extra core required by the debugging system is at all times at user control and may be altogether eliminated for production runs where this system is not used.

This, plus the amount of the dump received and its format is controlled by SYMBDUMP options introduced both in the compiler and the runtime system. (Similar control can be exercised through parameters to the assembly procedure SYMBDUMP which can be used at will for program debugging).

6.3.2 Dump format.

As regards the format of the dump in general, the following remarks are of relevance:

- the detail of the information provided can be graded in approximately similar levels as with the hexadecimal program dump, although in this case all information is given in the source language terms.

- as regards the line number identifications occurring in the dump they apply to the main program listing unless followed by IN <external module name>.

6.3.2.1 Blocks.

The currently existing block instance of the program under execution are identified by

- procedure/class identifiers and/or line number of their beginning in the source program. Prefixed blocks are displayed with their prefix identifier.

- current status of the block instance is always shown where relevant (e.g. 'detached', 'inspected', etc.)

- object instances are counted on generation, separately for each class and the corresponding counts then identify the respective objects throughout their life span (given in parentheses following the class name).

- addresses (in hexadecimal notation) identify only arrays and text objects in principle. However in the case that no object counters are provided, the system must resort to the use of addresses - in addition they can be provided also on request. Note though that due to garbage collecting the addresses may vary from dump to dump.

6.3.2.2 Local quantities.

The respective quantities declared/specified in a block instance occur in the dump after the block heading in the order of their definition. Note particularly that:

- in case that its current value is identical to the initial value the quantity does not appear at all.

- an attempt is made to identify actual parameters to formal parameters called by name. However, in case that this would entail an evaluation of an expression only an indication of a thunk presence is given.

- matches are shown for virtual specifications.

- in default one and only one quant is output on a line. However, the dump can be compressed by putting as many quants per line as possible when required (e.g. when SYSOUT is connected to a display screen).

6.3.2.3 Arrays.

Arrays form a specific kind of RT data structure which is also reflected in their dump:

- arrays are always mentioned in the dump together with their bounds. However, only non-default valued elements appear.
- as many as possible array elements are output on a line.
- respective array elements are identified by the true subscript (in parenthesis) in case of one dimensional arrays or by ordinal numbers (commencing with 1) enclosed in a single appostrophe in case of multidimensional arrays. The mapping between the ordinal numbers and the full subscripts is obtained by counting the array elements varying the first subscript most frequently, then the next, etc.

6.3.2.4 Text objects.

Similarly text objects, mostly due to their reference/ value properties, require a special treatment:

- the reference part of text objects signifies whether the object in question is a subtext, and in all cases the current position indicator value and length are shown.
- if the text value in its entirety can be placed on the current line it appears within a pair of double quotes. Otherwise only the stripped value is shown, possibly continued - rightly aligned after its first character - on one or more lines in which case the closing double quote appears only in case the last non-blank character comes on the same line of dump as the very last character of this text does.

6.3.3. System overhead.

The execution time overhead caused by this facility is hardly of relevance, however one should realise that the following overheads are inflicted in space requirements:

- the prototype section of the compiled program is expanded when dump of individual quants is required.
- all objects are expanded by a minimum of one fullword when object counts are required.
- the size of the loaded program is increased by ZYQDEBUG length when its services are potentially required.

6.3.4. System control.

In the sequel there is a detailed description of the compiler and RT option SYMBDUMP. Note that unless locally changed at system installation (using SIMCDF and SIMRDF macros), their default values are 0 (=NONSYMBDUMP) in case of compiler and 1 in case of RTS. Also note that the following overrides take place automatically:

- compiler SYMBDUMP is forced to 3 for separate compilation of classes and procedures.
- RTS SUMBDUMP is forced to 1 in case of an error or time limit overflow in garbage collection.
- RTS SYMBDUMP is suitably reduced when the program was compiled with SYMBDUMP<3 and dump of local block quantities is required through the initial RTS SYMBDUMP setting.

6.3.4.1 Compiler.

SYMBDUMP=3 causes extension of compiler generated prototypes by local quantity lists in addition to all below.

SYMBDUMP=2 causes extension of object lengths to encompass object counts which identify individual instances in the dump (hexadecimal addresses used otherwise).

SYMBDUMP=1 (equivalent to SYMBDUMP) causes the RTS ZYQDEBUG routine, producing the dump to be automatically linked in.

SYMBDUMP=0 (equivalent to NOSYMBDUMP) has the effect of avoiding the linkage of the above routine to the program, thus limiting the dump possibilities to the formatted hexadecimal dump (convenient for production runs).

6.3.4.2 RTS

SYMBDUMP<=1	no dump produced in case of error.
SYMBDUMP=2	headings of blocks on the operation chain appear.
SYMBDUMP=3	above plus the symbolic dump of the local quantities of the involved blocks.
SYMBDUMP=4	above plus SQS contents (an implicit garbage collection is forced).
SYMBDUMP=5	above plus headings of all referencable blocks.
SYMBDUMP=6	above plus full dump of all referencable blocks.

6.3.4.3 Additicnal control remarks.

- Hexadecimal addresses of block instanced will appear, in additional to eventual counters, with SYMBDUMP=7.

- Any of the above SYMBDUMP values may be multiplied by 16 to affect compression of quant dump.

- the values recommended for program testing are:

  compiler: SYMBDUMP=3

  RTS: SYMBDUMP=3 and DUMP=1 (default).

- the effective length of the lines produced by ZYQDEBUG is directly controlled by LRECL subparameter of SYSOUT DCB (maximum). The default value is customarily 132.

6.3.5. Example

The following example was solely designed to illustrate the symbolic dump facility at work. In order to cut its length short, the system output concerned with the snapshot of the compilation and the run time system options in use was omitted. Also omitted is the (useless) output made on the inspected printfile SIMPRINT, but on the other hand the SYSOUT output shown here is complete. Note that the SYMBDUMP settings were identical to those recommended in 4.3:

SIMULA 67 (VERS 86.88) *** SYMBDUMP DEMONSTRATION ***

```
01  Simulation begin real X; integer U, COUNT;                          B1
02      ref(Head) WAITQUEUE, OUTQUEUE;
03
04      Process class ENTITY; virtual: procedure SNAPSHOT;
05      begin real TIMEUSED;                                             B2
06            procedure SNAPSHOT(TITLE); name TITLE; text TITLE;
07            begin external assembly procedure SYMBDUMP;
08                  TITLE:="DEBUGGING VERSION";
09                  SYMBDUMP(4,"SITUATION BEFORE START");
10            end OF SNAPSHOT;
11
12            TIMEUSED:=Time;
13            into(WAITQUEUE);
14            reactivate this ENTITY delay Normal(38,3,U);
15            TIMEUSED:=Time-TIMEUSED;
16            COUNT:=COUNT-1; CHARS(COUNT):='*';
17            Wait(OUTQUEUE);
18      end OF ENTITY;                                                   E2
19
20      boolean STARTED;
21      character array CHARS(8:127);
22
23
24
25      U:=Inint;
26      WAITQUEUE:-new Head; OUTQUEUE:-new Head;
27      for COUNT:=1 step 1 until 5 do activate new ENTITY;
28
29      Inspect new Printfile(Intext(8)) do
30      begin text SUBTITLE;                                             B4
31
32            Open(Blanks(80));
33            Outtext("SIMULATION PROTOCOL:");
34            SUBTITLE:-Image.Sub(Pos+1,Length-Pos);
35            WAITQUEUE.Last qua ENTITY.SNAPSHOT(SUBTITLE);
36            Close;
37      end OF INSPECTION;                                               E4
38
39      STARTED:=true; Passivate; comment WILL GIVE A RT-ERROR:
40
41      end OF PROGRAM                                                   E1
```

NO DIAGNOSTICS FOR THIS COMPILATION.

Note that RESWD=3 was used in order to get the kew-words and standard indentifiers in lower case, also the indentation was taken care of by the compiler alone).

The first part of the SYSOUT output is produced by the systems utility SYMBDUMP :

```
--- SYMBDUMP CALLED AT LINE @@@9 (SITUATION BEFORE START) -----------

OPERATING CHAIN :

SYSOUT  INSPECTED AT SYSTEM LEVEL: IMAGE==#@99@58/POS=1 OF 132/="
SYSIN   INSPECTED AT SYSTEM LEVEL: IMAGE==#@99@E8/POS=1@ OF
                   8@/="3SIMPRINT

PROCEDURE SNAPSHOT ON LINE @@@6 (LOCAL TO ENTITY(5) ON LINE @@@4),
CALLED FROM LINE @@35:
       TITLE         IS SUBTITLE OF BLOCK ON LINE @@3@

BLOCK ON LINE @@3@:
       SUBTITLE      ==#@99398.SUB(22,59)/POS=1/="DEBUGGING VERSION

PRINTFILE(2) ON LINE @@@@ IN *PREDEFINED*, TERMINATED,
VISIBLE THROUGH INSPECTION:
       IMAGE         ==#@99398/POS=21 OF 8@/="SIMULATION PROTOCOL:
                       DEBUGGING VERSION
       DDNAME        :SIMPRINT
       LINE          =1
       LINESPERPAGE  =46
       SPACING       =1

SIMULATION BLOCK ON LINE @@@1:
       MAIN          ==MAINPROGRAM(1)
       CURRENT       ==MAINPROGRAM(1)
       U             =-1317@61513
       COUNT         =6
       WAITQUEUE     ==HEAD(1)
       OUTQUEUE      ==HEAD(2)
       #@99188 ARRAY CHARS(@:127):

SQS :  SCHEDULING TIMES      PROCESSES (SIMULATION BLOCK AT LEVEL 1,
                             TIME=@.@)

       @.@                   MAINPROGRAM(1) ON LINE @@@@ IN
                             *PREDEFINED*.
                             DETACHED AT LINE @@@@
       2.7475725173952E+@1   ENTITY(5) ON LINE @@@4,
                             DETACHED AT LINE @@14
       3.@6943817138672E+@1  ENTITY(2) ON LINE @@@4,
                             DETACHED AT LINE @@14
       3.16@68@3894@43@E+@1  ENTITY(4) ON LINE @@@4,
                             DETACHED AT LINE @@14
       3.2933876991272@E+@1  ENTITY(1) ON LINE @@@4,
                             DETACHED AT LINE @@14
       3.713@4626464844E+@1  ENTITY(3) ON LINE @@@4,
                             DETACHED AT LINE @@14

--- END OF SYMBDUMP CALL AT LINE @@@9 -------------------------------
```

This was the output produced before the execution error predicted on line 0039 disabled a normal program completion. The appropriate diagnostics accompanied by the operating chain dump follows on the next page.

```
ZYQ@67****PASSIV SQS.LAST AT LINE @@17 ********************************

OPERATING CHAIN :

SYSOUT  INSPECTED AT SYSTEM LEVEL: IMAGE==#@99@58/POS=1 OF 132/="
SYSIN   INSPECTED AT SYSTEM LEVEL: IMAGE==#@99@E8/POS=1@ OF
                                   8@/="3SIMPRINT

ENTITY(3) ON LINE @@@4, DETACHED AT LINE @@17:
        SUC             ==HEAD(2)
        PRED            ==ENTITY(1)
        SNAPSHOT        HAS MATCH AT LINE @@@6
        TIMEUSED        =3.713@4626464844E+@1

SIMULATION BLOCK ON LINE @@@1:
        MAIN            ==MAINPROGRAM(1)
        CURRENT         ==ENTITY(3)
        U               =-1317@61513
        COUNT           =1
        WAITQUEUE       ==HEAD(1)
        OUTQUEUE        ==HEAD(2)
        STARTED         =TRUE
        #@99188 ARRAY CHARS(@:127): (1)='*' (2)='*' (3)='*' (4)='*'
                                    (5)='*'
```

6.3.6. External utilities related to the SYMBDUMP facility.

SYMBDUMP

function: prints a symbolic snapshot of the program under execution. The output format is basically the same as that used in case of a run-time error detection with RT option SYMBDUMP > 1.

declaration: external assembly procedure SYMBDUMP

parameters:

- (short) integer which determines the extent of the snapshot as follows:

value	displayed
<=1	nothing
2	headings of blocks on the operating chain
3	above plus the symbolic dump of the involved blocks
4	above plus SQS contents
5	above plus headings of all referencable blocks
6	above plus full dump of all referencable blocks

- a single reference variable (of arbitrary qualification) causing the snapshot to be limited to the object referenced.

- a single text variable or a text constant (string) whose value is used as the snapshot heading.

- (short) integer from within one of the following intervals:

  <1 : current block level> or
  <1 - current block level : 0>

  which will cause the snapshot taken to be limited to a single block instance on the static chain counted upwards from the outermost block in case of positive value of backwards from the current block if negative.

result: no value returned.

notes:

- the order of the parameters is irrelevant with the exception of the (short) integer parameter ambiguity implied above.

- in case that a specific block dump is requested either by a reference parameter specification or by a display indication, the ordinary dump (of the operating chain, SQS and pool 1) is suppressed.

- if the total volume of the dump produced is at premium rather than its structure (e.g. when SYSOUT is connected to a display screen), the system may be instructed to output as many quants on a line as possible by using as the first parameter a value which is a 16-multiple of any of the values shown above.

- the length of the lines output SYMBDUMP is directly controlled by the SYSOUT setting of LRECL.

ID

function: object identification through its class membership.

declaration: external assembly text procedures ID

parameter: one and only parameter which must be a simple reference to an object of arbitrary qualification.

result: reference to a text object the value of which is the identifier of a class which the object passed as a parameter belongs.

notes:

- the class identifier returned is truncated to the first 12 characters if necessary.
- in case that the reference passed as the actual parameter does not currently refer to any object, the value returned is notext.

NO

function: object identification via internal count value or adress.

declaration: external assembly integer procedure NO

parameter: one and only one parameter which must be a simple reference to an object of arbitrary qualification.

result: integer value which is a numeric identification of the object referenced by the actual parameter. The following possibilities may occur:

sign(NO)>0	the value returned is the ordinal number of this particular object within the given class.
sign(NO)=0	which identifies a case when the reference value of the actual parameter is none.
sign(NO)<0	(occurs in case that the program was compiled with SYMBDUMP<2 and thus no internal object counters were provided). The value returned is the -1* <address of the object>.

note: when a program re-execution is affected without reloading (e.g.) using the SIMCNT monitor), the object counters remain at the values reached in the previous execution i.e. no resetting to zero occurs.

6.4 Dump.

A formatted core is optionally printed when an object program is terminated because of a run-time error. The DUMP parameter of the object program (2.2.2.1) determines the dump level.

DUMP=	meaning
0	No debugging information is provided. The job step is aborted if a run-time error occurs.
1	A diagnostic message and a register dump is printed if a run-time error occurs.
2	In addition to the information of level 1 the operation chain is printed.
3	Prints the information of 2, the sequencing sets of all SIMULATION blocks and the local sequence controls of all scheduled processes.
4	Prints the information of 3 and all local sequence controls of non-terminated objects.
5	Prints the information of 4 and all referable data structures in hexadecimal format.

7 Programming considerations.

Commencing with release 08.00 there is a built-inn facility in the 360/370 SIMULA System for recording program behaviour during the execution. A user can obtain useful insight into the program dynamics by requesting results of various measurements to be output.

In the following the user will be given full orientation about the tools available for improving performance of the SIMULA programs. It is not necessarily intended for a thorough study - on the contrary, it is hoped that a user interested in using this facility will find all sufficient information in paragraphs 7.1 and 7.4, eventually supported by a glimpse of the example. However, should she/he have any enquiries it should be possible to find full answers in the rest of the text.

7.1 Objectives and design principles.

The purpose of this facility (referred to as the turner in the rest of this text) is to provide a sufficient insight into the dynamics of execution in order to reduce its cpu-time and storage requirements. To this end special focus is put on supervising the two most critical issues:

- data transmissions realized through various kinds of assignments (the most frequently encountered operation)
- allocation of data structures (a complex operation dictated by the dynamic nature of SIMULA).

The character of this facility, however, suggests that it can also be useful in the program debugging process.

Since the collection of the various statistics during the program execution represents whatever small but nevertheless a certain overhead the facility has been designed in such a way that its function (and thus the overhead) can be completely eliminated at the user's will. Therefore the user should carefully consult paragraph 7.4 below to make sure of correct activation of the various tuner components.

7.2 Scope of the facility.

Even though useful in itself the bulk of this programming aid is just an offspring of other program development tools SYMBDUMP and TRACE (see separate documentation). In particular the latter and specifically its data flow monitoring subsystem are of primary importance for understanding the dynamics of the program behaviour. However, since these tools are basically designed for program debugging, their use for program tuning can be somewhat cumbersome.

In contrast to these facilities, the tuner concentrates on giving just the resulting contour of the dynamic profile of program execution. This image is created by several components outlined below.

7.2.1 Assignment counter

This is a global counter which is increased by one for each assignment statement carried out by the system.

Included here are:

- explicit value- and reference-assignments affecting user declared data items;
- implicit assignments in for-statements and those of parameter transmission to procedures and classes by value or by reference,
- text value assignments implicit to text PUT-attributes and the corresponding OUT-procedures of files (assignments to image subtexts).

Excluded (i.e. not recorded in the assignment counter) are the following assignments:

- those leading to alteration of system declared data items e.g. text position indicator setting, manipulation of SUC and PRED through SIMSET procedures, manipulation of the SQS through SIMULATION utilities and parameter transmission for standard procedure calls,
- parameter transmission of value and reference parameters to virtual, formal and external procedures,
- all assignments in SIMULA program modules which were compiled with option avoiding assignments recording.

The current value of the global assigment counter is available during the program execution through use of an external procedure TRACECNT (cf. TRACE documentation). This can be conveniently used for dynamic initiating/terminating of other program execution supervising facilities. The final value is output at the end of execution in the Run Time System trailing line. This value tends to be a fair measure of the volume of the performed computation.

7.2.2 Frequency monitor.

This is a control flow monitor updating local counters associated with both explicit and implicit assignment statements. Each time an assignment statement (of the kind affecting assignment counter, see above) is executed its local counter is increased by one so that at the termination of the program the values recorded in the local counters reflect fairly precisely the dynamic profile of the execution.

The contents of the local counters can be obtained at the end of execution both in numeric and graphic form. Furthermore indication is given of the most frequently executed as well as the void assignments. These tables give a deep insight into the program dynamics and specifically turn attention to the apparent execution time bottlenecks. Be aware though that a flat histogram does not necessarily imply the most efficient algorithm.

7.2.3 Data structure register.

This is a device for recording the number of generated instances of each data structure defined in the program. These include:

- block instances of various kinds (however, only those which are user defined appear in the final account).
- user and system generated text objects.
- all declared arrays and arrays passed as parameters by value.

Not counted are:

- system generated control blocks (drivers and event notices since these are at least partly reused).
- system generated storages for holding temporary results which may be quite many but are unfortunately not under direct user control.
- static instances of standard procedures, buffers and other accessories necessary for the operating system interface because these do not take space in the working storage.

While the current value of a particular object counter serves at each time as a unique identification of the next generated instance of this kind, the final list of these counters is most valuable information for the assessment of program storage requirements. In addition the ratio of the respective data structures in the total program storage demand is estimated. Obviously, this kind of information is a more direct hint on how to improve program performance than just a number of store collapses.

With the considerable overhead of SIMULA storage management in mind, one can achieve drastic reductions of cpu-time usage by minimizing e.g. the number of generated procedure instances. However, one more aspect which must be considered here is the prevailing way in which a procedure is invoked (direct or remote call, virtual, formal procedure etc.).

7.3. Output format of the respective components.

All the information provided by this facility appears on the standard printfile SYSOUT appended to the user program produced printout. With the exception of the histogram part of the frequently monitor the output can be intercepted also on a teletype compatible device with line length of 80 characters.

With the exception of the global assignment count the output is organised by modules: the first come execution frequency counters and the list of allocated data structures concerning the main program alone and then follow corresponding tables for the separately compiled modules used in the program (if any) in turn.

When organizing the output modulewise the following rules are observed:

- it is the maximum assignment frequency within a particular module that is used for forming out the histogram and for marking the lines of specific interest in the module.

- the data structure counts indicate the outcome of the complete program execution. Consequently also the logical partitioning of all allocated storage on the fragments taken by the respective data structures shown in the last (%) column is related to the complete execution.

- Obviously text and array objects may be allocated in the course of execution of any module. However their statistics are only given globally and appear together with the dynamic profile of the main program.

As regards the particular format of the respective tuner components see illustration in the paragraph 5 and also consider the remarks below.

7.3.1 Final assignment count.

Final assignment count appears (if requested, see par. 4) at the RTS trailing line in the form:

<count> ASSIGNMENTS RECORDED

Be aware that the true value can be greater than the figure given in case that one or more separately compiled modules used in the program had assignment counting suppressed.

7.3.2 Assignment frequencies.

Assignment frequencies (if registered) appeared under the title:

DYNAMIC PROFILE OF THE PROGRAM EXECUTION BASED
ON DATA FLOW MONITORING

The output consists basically of two parts:

- a numeric table of absolute frequencies given for the respective assignments in the given module,
- a histogram (adjacent to the table) showing schematically relation between the maximum frequency on a line and the maximum frequency within the module.

Note that the original source text is not reproduced as is sometimes customary in these reports. Instead references are made to its respective lines through the line numbers.

Several further details may be of interest:

- o respective frequencies for up to 7 (first) assignments are registered on each line - the user must alone match the figures given with the corresponding source listing (identified in the report),
- o the maximum frequency on a line appears in a clearly marked column next to the line number,
- o the maximum frequency over the whole module is marked by a sequence of exclamation marks at the left of the line number.

Finally note that the following convention is used when generating these tables: in default only lines where there has been at least one assignment executed appear in the table. Optionally however, the user can require a report on every single line. If this is the case the following rules apply:

- o just the line number alone is given for lines without any executable code,
- o the line number followed by a colon with the rest of the line blank signifies a line with executable code not containing any assignment,
- o all lines containing assignment statements of which none was performed in the course of the program execution are marked by a sequence of question marks to the left of the line number.

7.3.3 Generated data structures.

List of the generated data structures appears under the title:

USER GENERATED DATA STRUCTURES

Listed in the table are various block patterns defined in the module in question, some instances of which were generated during the program execution. For each block pattern, the following information is given:

- pattern identification (its kind and commencing line number in the source (text),
- total number of generated instances,
- size (in bytes) of instance. Note that this is the true size i.e. includes space for user data, the necessary system overhead, block counter location plus eventual padding to meet the storage allignment requirements (if any),
- share of the total program storage demand attributed to this particular block kind (in percent rounded to the nearest integer).

Following the table the total sum of all generated block instances is given together with an indication of how large a portion of the total storage used this represented. (Note that when estimating total storage use only true store collapses are considered i.e. the effect of garbage collections occurring as a side effect e.g. various program development utilities are disregarded.)

If the related module is the main program the total number of generated text and array objects together with the indication of how large a portion of the total storage used these represented is also given in percent.

Note that the percent figures will not sum up to 100%: the discrepancy is accounted for by the system generated data structures (drivers, event notices and temporary and standard objects) which do not appear anywhere in this table.

Finally it is worth mentioning that also this table is compressed to the block patterns of which no instances were generated at all are suppressed in default. Since their appearance may have som value for program documentation, the user can optionally request them to be output together with nonempty lines.

7.4 Control.

The production of the tuner printouts is under full user control by means of the compiler and the runtime system options.

The following combinations are required for output of the respective items:

total number of assignments registered during program execution will appear if:

- the program was compiled with SYMBDUMP>3
- and run with the TEST option.

assignment frequency table (and the histogram) will appear if:

- the program (module) was compiled with SYMBDUMP>3,
- the program was run with SYMBDUMP>3
- and the program terminated with return code zero (i.e. without any RT-error).

the generated data structures are listed if:

- the program was compiled with SYMBDUMP>1
- and run with SYMBDUMP>3.

Note that default settings at SIMULA system distribution are SYMBDUMP=0 for compiler and SYMBDUMP=1 for the runtime system so that they have to be either permanently reset at the installation or overriden at individual runs in order to activate the tuner.

As regards the expansion of the tables to their full length (i.e. including also the void entries which are suppressed in default, see par. 3) the following conventions apply:

The specification of the RTS option SYMBDUMP must be followed immediately by

- o the seguence AA to force "All Assignments" to appear, or by
- o the sequence AB to force "All Blocks" to appear, eventually by
- o the letter A alone get All assignments and blocks,

e.g.

```
//EXEC SIMCLG,PARM.SIM='SYMBDUMP=4',PARM.GO='SYMBDUMP=4AB'
```

will cause the full table of user defined block patterns to appear while the assignment frequency table which will also be generated will include only non-empty entries.

Note that the best way how to alter default values of the compiler and the RTS option SYMBDUMP is through the use of the macros SIMCDF and SIMRDF which are part of the system delivery. This would however not suffice for the expansion of the tuner tables where the letters A, AA or AB can only be specified at execution.

7.5 Example

Let us suppose that the following program is compiled with SYMBDUMP=4 (shown below is an extract of the compilation listing):

```
begin class BOARD(E,F,RIMCHAR); integer E,F; character RIMCHAR;
      begin character array C(0:E,0:F); integer I,J,M,N;

            procedure PRINT;
            for I:=0 step 1 until E do
                begin for J:=0 step 1 until F
                      do Outchar(C(I,J));
                      Outimage
                end***PRINT***;

            procedure READ;
            for I:=1 step 1 until M do
                begin Inimage;
                      for J:=1 step 1 until N do
                          if Inchar=' '
                            then C(I,J):=' ' else C(I,J):='*';
                end***READ***;

            M:=E-1; N:=F-1;

            for I:=0 step 1 until N do
                C(0,I):=C(E,I+1):=RIMCHAR;
            for J:=0 step 1 until M do
                C(J+1,0):=C(J,F):=RIMCHAR;
      end***BOARD***;

      BOARD class LIFE;
      begin integer procedure NEIGHBOURSOF(I,J); integer I,J;
            begin integer T;
                  if C(I-1,J-1)='*' then T:=T+1;
                  if C(I-1,J  )='*' then T:=T+1;
                  if C(I-1,J+1)='*' then T:=T+1;
                  if C(I  ,J-1)='*' then T:=T+1;
                  if C(I  ,J+1)='*' then T:=T+1;
                  if C(I+1,J-1)='*' then T:=T+1;
                  if C(I+1,J  )='*' then T:=T+1;
                  if C(I+1,J+1)='*' then T:=T+1; NEIGHBOURSOF:=T;
            end***NEIGHBOURSOF***;

            procedure WRITE(GENERATION); integer GENERATION;
            begin Outtext("GENERATION NO "); Outint(GENERATION,6);
                  Outimage; PRINT
            end***WRITE***;
      end***LIFE***;
```

Continued on the next page....

....Continued from the previous page:

```
      procedure SWAP;
      begin TEMP:-OLDB; OLDB:-NEWB; NEWB:-TEMP end***SWAP***;

      ref(LIFE) OLDB,NEWB,TEMP;       character C1;
      integer Linesperpage,P,Q,GENS,Z, LIFELENGTH;
      external real procedure BALAS;

      LIFELENGTH:=BALAS(Inint,Inint);

      Outtext("INITIAL STATE"); Outimage;
      P:=Inint;    Q:=Inint;   Linesperpage:=64;

      OLDB:-new LIFE(P,Q,'+'); OLDB.READ;  OLDB.PRINT;
      NEWB:-new LIFE(P,Q,'+');

      for GENS :=1 step 1 until LIFELENGTH do
          begin inspect OLDB
                      do begin for P.=1 step 1 until M do
                           for Q:=1 step 1 until N do
                               begin Z:=NEIGHBOURSOF(P,Q);  C1:=C(P,Q);
                                     if Z=3
                                        then C1:='*'   else
                                     if Z=2 and C1='*'
                                        then C1:='*'   else C1:=' ';
                                    NEWB.C(P,Q):=C1;
                               end
                         end;
                NEWB.WRITE(GENS);  SWAP
          end
end
```

Note in particular the declaration of an external procedure BALAS at line 53 and its use at line 57. Let us assume that the procedure BALAS was compiled with SYMBDUMP=3 and that the program above is run with the run time option TEST and SYMBDUMP=4.

Disregarding the normal output which is of no particular interest to this demonstration the following two lines will at last appear on SYSOUT:

```
END OF SIMULA PROGRAM EXECUTION AT 17:@6:46.86
EXECUTION TIME @.@9 SEC.
RETURN CODE IS #@@@@@@@@
@@@@@ STORECOLLAPSES, DATA STORAGE USED : 9184 BYTES
21@8 ASSIGNMENTS RECORDED
```

These are the RTS trailing lines (usually prefixed by the ZYQ994 message identifier). The very last item is the final value of the global assignment counter implying in this case that altogether 2108 data transmission were recorded when executing this program. (Be aware though that this does not include assignments needed for execution of the procedure BALAS because that was compiled with SYMBDUMP less than 4).

On the next page comes the assignment frequency table and the histogram shown in Appendix A. Note that only "active" lines appear in this overview. Obviously line 8 contains the most frequently executed construction.

(As regards the for-statements with step-until elements remember that these contain two implicit assignment statements, see Common Base, section 6.2.3., case 2. These may appear on two successive lines if the for-statement is split).

The dynamic profile of the main program then continues on the next page by

USER GENERATED DATA STRUCTURES:

BLOCK PATTERN	LINE	TOTAL	SIZE	%
SUB-BLOCK	@@@2	1	64	1
PROCEDURE PRINT	@@@5	8	24	2
PROCEDURE READ	@@12	1	24	@
CLASS LIFE	@@28	2	48	1
PROCEDURE NEIGHBOURSOF	@@29	112	32	39
PROCEDURE WRITE	@@41	7	16	1
PROCEDURE SWAP	@@47	7	16	1
TOTAL NUMBER OF BLOCK INSTANCES:		138	-	46
TOTAL NUMBER OF TEXT OBJECTS :		2	-	3
TOTAL NUMBER OF ARRAY OBJECTS :		8	-	7

THE LINE NUMBERS CORRESPOND TO THE SOURCE COMPILED ON 25 NOV 198@ AT 17:@5:28.13

Finally, the very last page holds the dynamis profile (or what is available of it remember the compilation of the external mode was carried out with SYMBDUMP=3) concerning the module BALAS:

DYNAMIC PROFILE OF THE PROGRAM EXECUTION ** MODULE: BALAS **

USER GENERATED DATA STRUCTURES:

BLOCK PATTERN	LINE	TOTAL	SIZE	%
PROCEDURE BALAS	@@14	1	8@	1
PROCEDURE MAXX	@@22	3	32	1
PROCEDURE MAXT	@@29	4	48	2
PROCEDURE CHANGE	@@5@	4	32	1
PROCEDURE SOLVER	@@61	2	32	1
TOTAL NUMBER OF BLOCK INSTANCES		14	-	6

THE LINE NUMBERS CORRESPOND TO THE SOURCE COMPILED ON 25 NOV 1980 AT 17:@5:@8.13

Note here that of the total storage required for the execution of the above program approximately 52% (46+6) was taken up by various instances of which most is clearly attributed to procedure NEIGHBOURSOF altogether 112 times.

DYNAMIC PROFILE OF THE PROGRAM EXECUTION BASED ON DATA FLOW MONITORING

NOTES	LINE	MAXIMUM	FREQUENCES OF THE RESPECTIVE ASSIGNMENTS				----- H I S T O G R A M -----
	@@@6:	48	8	48			********
	@@@7:	48	48				********
!!!!!	@@@8:	288	288	288			**********
	@@13:	4	1	4			*
	@@15:	16	4	16			***
	@@17:	12	4	12			**
	@@2@:	2	2	2			
	@@22:	1@	2	1@			**
	@@23:	1@	1@	1@			**
	@@24:	1@	2	1@			**
	@@25:	1@	1@	1@			**
	@@31:	3@	3@				*****
	@@32:	41	41				********
	@@33:	32	32				*****
	@@34:	41	41				********
	@@35:	43	43				********
	@@36:	32	32				*****
	@@37:	43	43				********
	@@38:	112	35	112			**********
	@@42:	7	7	7			*
	@@48:	7	7	7	7		*
	@@57:	1	1				
	@@59:	1	1				
	@@6@:	1	1	1	1		
	@@62:	1	1	1	1	1	
	@@63:	1	1	1	1	1	
	@@65:	7	1	7			*
	@@67:	28	7	28			*****
	@@68:	112	28	112			**********
	@@69:	112	112	112	112	112	**********
	@@71:	28	28				*****
	@@73:	63	21	63			**********
	@@74:	112	112				**********
	@@77:	7	7				*

8 Bibliography.

(1) SIMULA Users Guide. Publication No. S-24. Norwegian Computing Center, Oslo, Norway.

(2) IBM System/360 OS: Storage Estimates.

(3) IBM System/360 OS: Concepts and Facilities. Form C28-6535.

(4) IBM System/360: Principles of Operation. Form A22-6821-7.

(5) IBM System/360 OS: Assembler Language. Form C28-6514-5.

(6) IBM System/360 OS: Job Control Language. Form C28-6539-8.

(7) IBM System 360 OS: Supervisor and Data Management Services. Form C28-6646-2.

(8) IBM System/360 OS: Supervisor and Data Management Macro Instructions. Form C28-6647-3.

(9) IBM System/360 OS: Utilities. Form C28-6586-9.

(10) IBM System/360 OS: Linkage Editor. Form C28-6538.

(11) IBM System/360 OS: Messages and Codes. Form C28-6631-6.

(12) Ole-Johan Dahl, Bjnrn Myrhaug, Kristen Nygaard: SIMULA 67 Common Base Language. Publication No. S-22. Norwegian Computing Center, Oslo, Norway.

(13) P. Naur (Ed.): Revised Report on the Algorithmic Language Algol 60.

(14) IBM System/360 OS: FORTRAN IV library subprograms. Form C28-6569.

(15) IBM System/360 OS: FORTRAN IV (G & H) Programmers Guide. Form C28-6817.

Note: Page number given in references to IBM documentation are only approximates, since they may vary with the release number.

HARDWARE REPRESENTATION OF THE SOURCE LANGUAGE

A SIMULA program which is to be processed by the compiler should be contained in the job-input stream or put in a sequential data set with fixed or fixed blocked record format and 80 byte record length. It can also be a member of a partitioned data set with the same record characteristics. The program must be contained in columns 1-72 of the lines, and columns 73-80 may be used for sequence numbers (these columns will not be processed, but they will appear on the program listing). The EBCDIC code is used.

Basic symbols.

A basic symbol is represented as a keyword, a special character or a sequence of special characters (table A.1).

Identifiers.

Identifiers should follow the syntax of ALGOL 60, and the syntatic class <letter> consists both of lower and upper case letters. Lower case and upper case letters are not distinguished, except in text or character constants. 12 characters are significant in an identifier. An identifier must not be a keyword used to represent a basic symbol.

Constants.

Arithmetic constants follow the ALGOL 60 syntax. If an integer constant is out of integer range (3.1, p.2) it is regarded as a real constant. A constant can be given in hexadecimal as 1 to 16 hexadecimal digits, preceded by #. The constant is regarded as real if it is followed by an R.

A character constant is represented as the desired character, enclosed in single quotes (').

A text constant (<string>) is represented as the desired sequence of characters, enclosed in double quotes ("). A text constant may not contain a double quote without the use of a special convention: If a text string contains two double quotes in sequence, this is interpreted as one double quote within the text, and not the ending double qoute.

Any basic symbol, identifier or constant must be contained in one line, and it must not contain interspersed blanks. A text constant may, however, continue over several lines. Adjacent identifiers or basic symbols represented by keywords must be separated by line shift or at least one blank.

Comments.

A comments consists of the basic symbol COMMENT, followed by a nonalphameric character, and the successive characters up to and including the next semicolon (; or $).

An end comment is the string of characters following the basic symbol END up to, but not including the next semicolon, END, ELSE, WHEN or OTHERWISE.

Comments and end comments are printed on the program listing, but they are not further processed by the compiler.

A warning message is issued if an end comment contains the basic symbols ":=", ":-", "(", or "GOTO".

basic symbol	representations (EBCDIC)		basic symbol	representations (EBCDIC)
=	=	EQ	==	==
	<>	NE	=/=	=/=
>	>	GT	:	:
<	<	LT	:=	:=
	>=	GE	.	.
	<=	LE		
	OR		;	; $
&	AND		(	(
	IMP		)	)
	EQV			(
	¬	NOT		)
+	+		,	,
-	-			&
*	*			
	**			
/	/			
	//			

Table A.1 Basic symbol representations.

Basic symbols consisting of boldface or underlined keywords in the reference language are represented as the corresponding word in capital letters. The basic symbol go to can be punched with or without space(s) between GO and TO.

Listing and input control lines.

The compilation listing and compiler input can be controlled by special lines put in the compiler input data set. Such lines are identified by a % sign in column 1, immediately followed by a control word. The allowed control lines are:

%TITLE text

The contents of columns 10-55 of this line are put in printer positions 24-69 of the headings of subsequent pages. Any old title text is overwritten. Source program listing continues with the next line on a new page. The %TITLE line itself is not listed.

%PAGE

The program listing continues with the next line listed on a new page. The %PAGE line is not listed.

%NOSOURCE

Suppress listing of the source program. The lines following this line up to the next %SOURCE line (if any) will not be listed. The line numbers will, however, be updated. THE %NOSOURCE line is not listed.

%SOURCE

Resume listing of the source program. The %SOURCE line is not listed.

%COPY text

The %COPY control line will obtain source-language coding from a partitioned or sequential data set and insert it into the program currently being compiled. The coding to be inserted is identified by the text on the line, which is interpreted as the name of a member in the partitioned data set.

The partitioned data set is identified by the SYSLIB DD-statement supplied to the compilation step.

If this member cannot be located, or if SYSLIB is not specified, the system will try to locate a sequential data set with the ddname identical to what was expected to be the member name.

The %COPY card will be listed.

%ENDCOPY

This line signals the end of predefined source-language coding obtained by a %COPY line. It must be the last line of any member in a partitioned data set which will be obtained by a %COPY line. The data set should have blocksize not greater than 1600. The code to be copied is inserted in the library by the IEBUPDTE or IEBUPDAT utility program (see (9)).

The %ENDCOPY line will be listed.

%RESWD=n

This has the same effect as the compiler parameter RESWD, see section 2.2.1.1.

%INDENT=n

This has the same effect as the compiler parameter INDENT, see section 2.2.1.1.

%NOINDENT

Suppress indentation of subsequent lines in the compilation listing, e.g. enabling visual distinction of label declarations in an otherwise automatically indented listing.

%INDENT

Resume the indentation at the earlier level.

COMPILER DIAGNOSTICS

The general format of a compiler diagnostic is:

```
SIMhhh  c  CARD  nnnn  text
```

hhh — three hexa-digits giving compiler component issuing message (first hexadigit) and message serial within component.

c — Severity code:

- W — Warning; the compiler has detected a possibly erroneus construction but compilation and object program generation continue.
- 3 — Recovery error; The program contained an errror that could be corrected by inserting or changing some symbol(s) in the program.
- 7 — Severe error; the compiler had to discard part of the program, but the compilation could continue. Object module output is suppressed.
- B — Severe error; the compilation is terminated after the current scan of the program text.
- E — Severe error; the compilation stops at once.
- F — Internal error; follow report procedure in Appendix K.

nnnn — Card number at which the error was detected.

text — Message text. Lower-case letters and the sequence XXXX in the message text denote fields that will be filled with relevant information when the message is printed.

Compiler messages:

If an internal error message is given for a compilation, this might be a result of a bad recovery from another error. Could this be the case, correct these errors and recompile the program.

In case only an internal error message is given, please report the error to NCC as described in the SIMULA Programmers Guide.

```
SIM000  F  LINE  nnnn  INTERNAL ERROR: PROGRAM INTERRUPT (n)
           AT  hhhhhhhh  RELATIVE TO LINKEDIT ORIGIN
```

User response: Follow report procedure.

```
SIM001  E  XXXX  CANNOT BE OPENED, DDCARD MAY BE MISSING
```

User response: Supply or correct the DD-card.

SIM002 7 SYSUT3 BUFFER SIZE SPECIFIED TOO BIG - REDUCED TO nn BYTES

Explanation: The two SYSUT3 buffers could not be placed in the area specified (by default, or by use of the SIZE parameter - see section 2.2.3.1 for a description of the use of the SIZE parameter and its default values).

User response: Change SIZE parameter if used, otherwise contact a systems programmer.

SIM003 F ATTEMPT TO READ PAST END OF FILE ON SYSLIB - INTERNAL COMPILER ERROR

User response: Follow report procedure.

SIM004 E LINE nnnn I/O ERROR

Explanation: An input/output error occured when the compiler was writing the object module. A SYNADAF message further identifying the cause of error is also printed.

User response: Check the corresponding DD-card for invalid parameters. If none are found, consult a system programmer.

SIM005 F LINE nnnn EOF ON SYSUTn

User response: Follow report procedure.

SIM006 E LINE nnnn I/O ERROR ON SYSUTn

User response: Check the corresponding DD-card. If it appears to be correct, consult a systems programmer.

SIM007 F LINE nnnn EOF ON SYSIN

User response: Follow report procedure.

SIM008 3 LINE 0000 SYSPUNCH COULD NOT BE OPENED, 'NODECK' ASSUMED

Explanation: The DD-card for SYSPUNCH was probably missing although a deck had been requested.

Compiler action: No object deck is punched.

SIM009 3 LINE 0000 SYSGO COULD NOT BE OPENED, 'NOLOAD' ASSUMED

Explanation: Analogous to SIM008.

SIM00A E LINE nnnn I/O ERROR ON SYSIN

User response: Check that the SYSIN DD-card defines a SIMULA source program (App.A). If no error is found, contact a systems programmer.

SIM00B E LINE nnnn ATTEMPT TO WRITE ON SYSUT1 OR SYSUT2 WITH BLKSIZE xxx > LIMBLKSZ xxx

Internal message.

SIM00C W 0000 MAXERROR SPECIFIED TOO LARGE, 50 USED

Explanation: The user has requested too many errors to be listed.

Compiler action: A maximum of 50 errors will be listed.

SIM00D E LINE nnnn ESTIMATED NUMBER OF PAGES EXCEEDED

User response: Increase value of the compiler parameter MAXPAGES if more printed output is required.

SIM00E E LINE nnnn ESTIMATED COMPILATION TIME EXCEEDED

User response: Increase value of the compiler parameter TIME, remember though that the value should remain less than the EXEC step time limit (if any).

SIM00F F LINE nnnn ILLEGAL SYMBOL (hhhhhhhh) DETECTED BY OUTSYM - INTERNAL ERROR

User response: Follow report procedure.

SIM010 F LINE nnnn ILLEGAL SYMBOL (hhhhhhhh) DETECTED BY INSYM - INTERNAL ERROR

User response: Follow report procedure.

SIMOFF F RELEASE nnnn IS OUT OF DATE AND MUST BE REPLACED

Explanation: All delivered systems have only limited duration and must be withdrawn from time to time. Updated fresh copies are delivered in good time for replacement so that the responsability for installation of new releases resides with the licencee.

Messages from Pass 1:

SIM101 3 LINE nnnn NEAR COLUMN nn, EXCESS DECIMAL POINT

Explanation: More than one decimal point was found in a numeric item.

Compiler action: The excess decimal point is ignored.

SIM102 3 LINE nnnn NEAR COLUMN nn, MISSING DIGIT

Compiler action: The digit 0 is inserted.

SIM103 3 LINE nnnn NEAR COLUMN nn, EXCESS EXPONENT

Explanation: More than one & in a numeric item.

Compiler action: Previous exponent used as mantissa, new exponent developed.

SIM104 8 MISSING TEXT QUOTE

Explanation: The source program ended inside a text constant.

User response: Check all text constants (note that any text quotes must be before or in column 72).

SIM105 7 LINE nnnn NEAR COLUMN nn, MISSING CHARACTER QUOTE

Compiler action: The quote that was found is ignored.

User response: Check if a character was intended. If so, add the missing quote. Note that a character constant must be on one card only, and that the power-of-ten symbol is (&), not (').

SIM106 7 LINE nnnn NEAR COLUMN nn, ILLEGAL CHARACTER

Explanation: A character which cannot be part of a SIMULA source program is found outside a comment or a character or text constant.

Compiler action: Character is treated as blank.

SIM107 7 LINE nnnn NEAR COLUMN nn, 'GO' NOT FOLLOWED BY 'TO'

Compiler action: 'GO' is regarded as 'GOTO'.

SIM108 7 LINE nnnn NEAR COLUMN nn, 'REAL' EXPECTED AFTER 'LONG'

Compiler action: 'REAL' is inserted.

SIM109 7 LINE nnnn NEAR COLUMN nn, 'INTEGER' EXPECTED AFTER 'SHORT'

Compiler action: 'INTEGER' is inserted.

SIM10A 7 LINE nnnn NEAR COLUMN nn, '=' EXPECTED

Explanation: '=/' should be followed by '='.

Compiler action: '=/' is assumed to mean '<>'.

SIM10B E LINE nnnn MORE THAN nnnn DIFFERENTLY SPELLED IDENTIFIERS, ID TABLE FULL - CORE SIZE TOO SMALL

User response: If there are more than 3072 differently spelled identifiers, the program must be re-written, otherwise it may be compiled in a larger core area.

SIM10C 3 LINE nnnn NEAR COLUMN nn, DIGIT EXPECTED AFTER '&'.

Compiler action: '&' is disregarded.

SIM10D 3 LINE nnnn NEAR COLUMN nn, INTEGER LARGER THAN 2*31-1, TAKEN AS REAL

Compiler action: The integer constant is converted to a real constant.

SIM10E B LINE nnnn END OF FILE WHILE SCANNING COMMENT

SIM10F W LINE nnnn NEAR COLUMN nn, SEMICOLON MAY BE MISSING FOLLOWING END

Explanation: A ':=', ':-', '(' or 'GOTO' is found in an end comment. This may happen if the semicolon terminating the ende comment has been omitted.

SIM110 7 LINE 0000 NO SOURCE PROGRAM TEXT

User response: Check the SYSIN DD-card.

SIM111 3 LINE nnnn NEAR COLUMN nn, NUMBER TOO BIG

Compiler action: The largest representable number is used (appr. 10**75).

SIM112 3 LINE nnnn NEAR COLUMN nn, NUMBER UNDERFLOW, 0 USED AS VALUE

User response: Read section 3.1 of this manual.

SIM113 3 LINE nnnn NEAR COLUMN nn, EXPONENT > 75

Compiler action: cf. SIM111.

SIM114 7 LINE nnnn NEAR COLUMN nn, UNMATCHED 'END'

Explanation: More 'END':s than 'BEGIN':s.

User response: Check the program structure. BEGIN-END numbering in right margin should be useful for this purpose.

SIM115 3 LINE nnnn TOO MANY BITS IN A NONDECIMAL CONSTANT - TRUNCATED

Explanation: This message is related to use of unsupported feature (nondecimal numerals).

SIM116 B LINE nnnn DEBUG CODE INCORRECTLY SPECIFIED

User response: Check for mis-punch in column 1.

SIM117 7 LINE nnnn NEAR COLUMN nn, 'TO' NOT PRECEDED BY 'GO'

Compiler action: 'TO' is ignored.

User response: Use a different identifier if 'TO' is meant as such, otherwise insert 'GO'.

SIM118 7 LINE nnnn 'EXTERN' NOT SPECIFIED - OBJECT MODULE OUTPUT CANCELLED

Explanation: The source program is a class or procedure definition.

User response: Correct the program of supply the EXTERN parameter to the compiler.

SIM119 7 LINE nnnn '%COPY' NOT FOLLOWED BY VALID NAME, OR '%COPY' FOUND IN COPIED CODE

Explanation: 1. The first non-blank character after '%COPY' was not alphabetic, or no non-blank character was found in the remaining portion of the card
or 2. The code to be copied contained a '%COPY' statement.

Guide to user: 1. Specify name of existing source module on same card as '%COPY'.
2. Avoid recursive copy.

Compiler action: Card ignored.

SIM11A 7 LINE nnnn SYSLIB CANNOT BE OPENED - DD-CARD MAY BE MISSING

Compiler action: No copy code accepted.

SIM11B 7 LINE nnnn MEMBER TO BE COPIED NOT FOUND ON SYSLIB

Compiler action: '%COPY' statement is ignored.

User response: Check the '%COPY' statement and, if necessary, the source library.

SIM11C 7 LINE nnnn SYSLIB BLKSIZE GREATER THAN 3200

Compiler action: '%COPY' statement is ignored.

User response: Check the SYSLIB DD-card and, if necessary, reblock the library (see (9)).

SIM11D 7 LINE nnnn I/O ERROR ON SYSLIB DURING DIRECTORY SEARCH

Compiler action: '%COPY' statement is ignored.

User respons: Consult a systems programmer. (List and correct the PDS directory, or recreate the PDS).

SIM11E 7 LINE nnnn I/O ERROR ON SYSLIB WHILE READING A MEMBER

Compiler action: '%COPY member' is ignored.

User response: Recreate the number or, if necessary, the entire library.

SIM11F W LINE nnnn LENGTH OF TEXT CONSTANT>132 (ONE PRINTED LINE)

User response: Chech for missing or improperly punched text quote. If no quote is missing, ignore message.

SIM120 B LINE nnnn NUMBER OF IDENTIFIERS (INCLUDING STANDARD IDENTIFIERS)=nn > 3072.

Explanation: The maximum number of differently spelled identifiers allowed in this implementation is 3072.

User response: Redeclare quantities where possible.

SIM121 B LINE nnnn 'EXTERNAL CLASS' NOT FOLLOWED BY VALID ID

SIM122 B LINE nnnn NOT ENOUGH SPACE FOR TREATING IDS OF EXTERNAL CLASS, (NIDS=xxx, yyy BYTES LACKING)

User response: Recompile in a longer partition.

SIM124 B LINE nnnn EXTERNAL CLASS DECLARATION IN COPIED CODE - ATTEMPT TO READ FROM SYSLIB RECURSIVELY

Explanation: Code segments inserted by %COPY statement must not contain other %COPY statement or external class declaration.

SIM125 B LINE nnnn UNEXPECTED END OF FILE WITHIN AN EXTERNAL CLASS

User response: Verify that entire separately compiled class is submitted - if the problem persists, contact NCC.

SIM126 B LINE nnnn INCORRECT FORMAT IN EXTERNAL CLASS CODE

Explanation: Internal error, please report to NCC, together with a hexadecimal dump of the precompiled class and the listing showing the error.

SIM127 B LINE nnnn INCORRECT SYNTAX IN EXTERNAL CLASS DECLARATION

Explanation: Internal error, please report to NCC, together with a hexadecimal dump of the precompiled class and the listing showing the error.

SIM128 3 LINE nnnn DIGIT >= RADIX IN A NON-DECIMAL CONSTANT

Explanation: The error is related to use of unsupported feature.

SIM129 7 LINE nnnn RADIX NOT 2,4,8 OR 16

Explanation: The error is related to use of unsupported feature.

SIM12A 7 LINE nnnn EXTERNAL CLASS OUTPUT SUPRESSED - SYSPUNCH CANNOT BE OPENED

User response: Check presence and validity of SYSPUNCH data set definition. It must be either a sequential data set or a pds member.

SIM12B W LINE nnnn INTEGER LESS THAN -2**31, TAKEN AS REAL

Explanation: A value of a fixed-point literal in the program text exceeds the hardware dictated limit.

SIM12C W LINE nnnn TOO MANY BITS IN A NONDECIMAL INTEGER CONSTANT, TAKEN AS LONG REAL

Explanation: A hexadecimal constant with more than 8 hexadecimal digits occurring in the program text, not appended by L.

SIM12D 7 LINE nnnn EXTERN SPECIFIED, BUT NEITHER PROCEDURE NOR CLASS SUPPLIED

Explanation: Compiler parameter EXTERN value was different from zero, however a main program in the form of a (prefixed) block or a compound statement was detected on SYSIN.

SIM12E W LINE nnnn xxxxxxxx IS AN UNKNOWN KIND OF EXTERNAL PROCEDURE, TREATED AS 'ASSEMBLY'

Explanation: A word following external was different from FORTRAN, ASSEMBLY, PROCEDURE or any known type.

SIM12F W LINE nnnn NOT EXACTLY COMPLETE PROGRAM, AT LEAST 'END' MISSING

Explanation: A SIMULA program must have form of a compound statement or a (prefixed) block. Neither of these were found on SYSIN.

SIM130 3 LINE nnnn MISSING DELIMITER IN EXTERNAL DECLARATION

Explanation: A comma missing as a separator in the identifier list of an external declaration.

SIM131 W LINE nnnn HIDDEN/PROTECTED FEATURE IS NOT IMPLEMENTED

Explanation: Attribute protection feature was used in the compiled program but it will not give the expected effect because the present release does not contain an implementation of this feature's semantics.

Messages from Pass 3:

SIM301 3 LINE nnnn THE SPECIFICATION OF THE PARAMETER XXXX HAS BEEN OMITTED

Compiler action: The parameter is given the specification INTEGER called by value.

Guide to user: If this specification was not intended superfluous messages may follow.

SIM302 3 LINE nnnn THE PARAMETER XXXX IS NOT IN THE FORMAL PARAMETER LIST

Explanation: The mode part of a procedure of class declaration specifies a parameter not in the formal parameter list.

Compiler action: The specification is ignored. This may cause 'UNDECLARED' messages later.

SIM303 E LINE nnnn STACK OVERFLOW

Explanation: The program has been embedded to too many levels of 'BEGIN', 'IF' (... symbols and/or formal parameters.

SIM304 B LINE nnnn STACK UNDEFFLOW: YOU PROBABLY HAVE TOO MANY 'END'S

Explanation: There are too many 'END':s or some missing 'BEGIN':s in the program.

Compiler action: The program text is discarded until a new 'BEGIN' symbol is found. Then the syntax for a program is applied to the remaining part, and message SIM318 is issued.

SIM305 7 LINE nnnn MISSING DELIMITER BEFORE XXXX

Explanation: A constant or identifier is not separated from the succeeding identifier by a delimiter.

Compiler action: Statement is discarded.

SIM306 7 LINE nnnn WRONG DELIMITER: XXXX IS ILLEGALY PLACED

Compiler action: Statement is discarded.

SIM307 7 LINE nnnn SEMICOLON OMITTED AFTER FORMAL PARAMETER PART.

Compiler action: The erroneous program text is discarded.

SIM308 7 LINE nnnn IMPROPER QUALIFICATION: IT SHOULD HAVE THE FORM REF(<IDENTIFIER>)

Compiler action: Declaration/specification discarded.

SIM309 7 LINE nnnn MISPLACED CONSTANT

Explanation: An identifier or constant is immediately followed by a constant.

Compiler action: Statement discarded.

SIM30A 7 LINE nnnn XXXX IS ILLEGALLY PLACED

Compiler action: Statement discarded.

SIM30B 7 LINE nnnn MISSING IDENTIFIER AFTER 'IN', 'IS' OR 'QUA'

Compiler action: Statement discarded.

SIM30C 7 LINE nnnn ILLEGAL CONSTRUCTION: THE CLAUSE BEFORE XXXX IS NOT PROPERLY ENDED

Compiler action: Statement discarded.

Guide to user: May be caused by an earlier error.

SIM30D 7 LINE nnnn IMPROPER ACTIVATION STATEMENT

Compiler action: Statement discarded.

SIM30E 3 LINE nnnn ILLEGAL USE OF PRIOR: 'PRIOR' MAY BE USED ONLY WITH 'AT' OR 'DELAY'

Compiler action: 'PRIOR' is ignored.

SIM30F 3 LINE nnnn THE FORMAL PARAMETER XXXX IS SPECIFIED MORE THAN ONCE

Compiler action: The latest specification is used and any mode definition for this parameter is ignored.

SIM310 3 LINE nnnn A SWITCH, LABEL, REF QUANTITY OR PROCEDURE - SUCH AS XXXX - MAY NOT BE PASSED BY VALUE: THE DEFAULT MODE IS SET.

SIM311 3 LINE nnnn THE FORMAL PARAMETER XXXX HAS DUPLICATE NAME/VALUE MODES

Compiler action: The first specified mode is used.

SIM312 7 LINE nnnn ILLEGAL SPECIFICATION: 'VIRTUAL' MUST BE FOLLOWED BY A COLON

Compiler action: Statement discarded.

SIM313 F INTERNAL ERROR - CONTACT A SYSTEMS PROGRAMMER

User response: Follow report procedure.

SIM314 3 LINE nnnn DUPLICATE 'INNER': AT MOST ONE INNER MAY APPEAR PER CLASS BODY

Compiler action: 'INNER' is ignored.

SIM315 3 LINE nnnn MISPLACED 'INNER': INNER MUST APPEAR AS A STATEMENT ON ITS OWN AT THE OUTERMOST BLOCK LEVEL OF A CLASS BODY

Compiler action: Statement is ignored.

SIM316 7 LINE nnnn ILLEGAL WHEN BRANCH: THE IDENTIFIER IS MISSING IN 'WHEN' <IDENTIFIER> DO'

Compiler action: The clause is ignored.

SIM317 7 LINE nnnn ILLEGAL WHEN BRANCH: THE DO IS MISSING IN 'WHEN <IDENTIFIER> DO'

Compiler action: An extra do has been inserted.

SIM318 B LINE nnnn UNBALANCED PROGRAM: A FURTHER 'BEGIN' HAS BEEN FOUND AFTER THE 'END' WHICH MATCHED THE INITIAL 'BEGIN'

Compiler action: Start from scratch.

SIM319 3 LINE nnnn UNBALANCED PROGRAM DUE TO INSUFFICIENT 'END'S: EXTRA 'END'S HAVE BEEN INSERTED AT THE FINISH OF YOUR PROGRAM

SIM31A 7 LINE nnnn UNBALANCED PARENTHESES BEFORE ':=' OR ':-'

Compiler action: Statement is discarded.

SIM31B 7 LINE nnnn A BRACKETED EXPRESSION IS NOT A LEGAL STATEMENT

Compiler action: Expression is ignored.

SIM31C 3 LINE nnnn MISPLACED NAME/VALUE PART - THE DEFAULT MODE HAS BEEN TAKEN. PLACE IT BEFORE THE SPECIFICATIONS.

SIM31D 3 LINE nnnn PASS 3 RECOVERY HAS INSERTED A 'BEGIN' HERE. THIS CHANGES IN SCOPE MAY GIVE RISE TO SPURIOUS ERROR MESSAGES IN LATER PASSES.

SIM31E 7 LINE nnnn THE PROGRAM TEXT STOPPED IN THE MIDDLE OF A CONSTRUCTION - PROBABLY DUE TO MISSING LINES.

SIM31F 3 LINE nnnn NO SEMI-COLON FOUND AFTER A PARENTHESIZED FORMAL-PARAMETER-LIST. AN EXTRA ';' HAS BEEN INSERTED.

SIM320 3 LINE nnnn THE CONSTRUCTION 'IF ... THEN INSPECT/FOR/WHILE ELSE' IS ILLEGAL. WRAP THE STATEMENT FOLLOWING THE 'THEN' IN A 'BEGIN - END' PAIR.

SIM321 7 LINE nnnn MISSING IDENTIFIER AFTER THE KEYWORD 'CLASS/PROCEDURE'.

SIM322 7 LINE nnnn MISSING ATTRIBUTE IDENTIFIER AFTER A DOT ('.').

SIM323 7 LINE nnnn TEXT CONSTANTS ARE NOT ALLOWED TO THE IMMEDIATE LEFT OR RIGHT OF REFERENCE COMPARATORS ('==' OR '=/=').

SIM324 8 LINE nnnn LABEL, SWITCH OR PROCEDURE MAY NOT BE USED AS A CLASS PARAMETER SPECIFIER.

SIM325 3 LINE nnnn LEFT PARENTHESIS OMITTED IN 'REF(IDENTIFIER)'. AN EXTRA '(' HAS BEEN INSERTED.

SIM326 3 LINE nnnn RIGHT PARENTHESIS OMITTED IN 'REF(IDENTIFIER)'. AN EXTRA ')' HAS BEEN INSERTED.

Messages from Pass 4:

SIM401 F LINE nnnn INTERNAL ERROR - IMPROPER BLOCK NESTING

User response: Follow report procedure.

SIM402 F LINE nnnn STACK OVERFLOW IN STACK nn (nn ENTRIES WERE ALLOCATED)

User response: Follow report procedure.

SIM403 F LINE nnnn INTERNAL ERROR - STACK UNDERFLOW IN STACK nn

User response: Follow report procedure.

SIM404 F LINE nnnn INTERNAL ERROR - MICPLACED SYMBOL XXXX

User response: Follow report procedure.

SIM405 7 LINE nnnn NUMBER OF PARAMETERS TO PROC/CLASS EXCEEDS 127

Explanation: See section 3.3.

SIM406 7 LINE nnnn NUMBER OF VIRTUAL SPECIFICATIONS EXCEEDS 255

Explanation: See section 3.3.

SIM407 7 LINE nnnn FOR-STATEMENT NESTING LEVEL EXCEEDS 30

SIM408 7 LINE nnnn NUMBER OF SWITCH ELEMENTS EXCEEDS 127

SIM409 7 LINE nnnn ARRAY DIMENSION EXCEEDS 127

Explanation: An array declaration indicated more than 127 subscripts.

SIM40A 7 LINE nnnn MORE THAN 127 IDENTIFIERS IN ID LIST

Explanation: A declaration list (i.e. a list of identifiers separated by commas in a declaration part) had more than 127 elements.

User response: Divide the list into lists with less than 127 elements.

SIM40B F LINE nnnn INTERNAL ERROR - STACK INDEX (nn) OUT OF RANGE

User response: Follow report procedure.

Messages from Pass 5:

SIM501 F LINE nnnn INTERNAL ERROR - XXXX WAS READ INSTEAD OF 'SUBLQL' SYMBOL

User response: Follow report procedure.

SIM502 F LINE nnnn INTERNAL ERROR - XXXX WAS READ INSTEAD OF 'LQLEND' SYMBOL

User response: Follow report procedure.

SIM503 7 LINE nnnn CLAS XXXX HAS UNDECLARED PREFIX XXXX

Compiler action: Prefix is ignored.

SIM504 7 LINE nnnn ILLEGAL PREFIX XXXX TO CLASS XXXX, XXXX IS NOT A CLASS

Compiler action: Prefix is ignored.

SIM505 7 LINE nnnn CLASS XXXX, DECLARED AT BLOCK LEVEL nn, IS USED AS PREFIX AT LEVEL nn

User response: Declare the subclass or prefixed block at the same block level as the prefix.

SIM506 7 LINE nnnn ILLEGAL PREFIX TO CLASS XXXX, XXXX BECAME VISIBLE THROUGH CONNECTION

User response: Declare the class in the class body of the connected class.

SIM507 7 LINE nnnn CLASS XXXX OCCURS IN ITS OWN PREFIX CHAIN

Compiler action: Prefix is ignored.

SIM508 7 LINE nnnn AMBIGUOUS IDENTIFIER XXXX (THIS DECLARATION OF XXXX REPLACES THE LAST MADE ON LINE nnnn)

Explanation: An identifier has been declared twice in a block.

SIM509 7 LINE nnnn CONFLICTING VIRTUAL SPECIFICATION OF XXXX IN CLASS XXXX

Explanation: XXXX (first occurrence) has been specified VIRTUAL twice in a prefix chain.

SIM50A B LINE nnnn QUALIFYING CLASS XXXX OF REFERENCE QUANTITY XXXX IS UNDECLARED (XXXX IS ASSUMED TO BE OF TYPE 'INTEGER')

Guide to user: The altered meaning of the quantity will probably trigger other error message.

SIM50B B LINE nnnn QUALIFIER XXXX OF REFERENCE QUANTITY XXXX IS NOT A CLASS (XXXX IS ASSUMED TO BE OF TYPE 'INTEGER')

Guide to user: cf. SIM50A.

SIM50C 7 LINE nnnn MATCHING QUANTITY XXXX IN CLASS XXXX IS OF IMPROPER KIND

Compiler action: The quantity will not be regarded as a match, and the virtual specification will become invisible (i.e. no further matches will be accepted).

SIM50D 7 LINE nnnn MATCHING QUANTITY XXXX IN CLASS XXXX IS OF IMPROPER TYPE

Compiler action: cf. SIM50C.

Guide to user: Read section 4.2.

SIM50E 7 LINE nnnn MATCHING PROCEDURE XXXX IN CLASS XXXX IS NOT SUBORDINATE

Compiler action: cf. SIM50C.

Guide to user: Read section 4.2.

SIM50F E LINE nnnn CAPACITY ERROR - ALLOCATE STACK OVERFLOW: TRY DECLARING SUBCLASSES BEFORE THEIR PREFIXES

Guide to user: This compiler capasity limitation can always be overcome by preceding a sufficient number of prefixed classes by their subclasses.

SIM510 E LINE nnnn ILLEGAL PREFIX XXXX - PREFIX LEVEL EXCEEDS 62

Explanation: See section 3.3.

SIM511 B LINE nnnn UNDECLARED BLOCK PREFIX XXXX

Compiler action: Prefix is ignored.

SIM512 E LINE nnnn STATIC BLOCK LEVEL GREATER THAN 15

Explanation: See section 3.3.

SIM513 W LINE nnnn QUALIFICATIONS XXXX AND XXXX HAVE INCOMPATIBLE BLOCK LEVELS

Explanation: The qualifications are either not on the same prefix chain or one of them has become visible by connection.

SIM514 B LINE nnnn QUALIFICATIONS XXXX AND XXXX ARE INCOMPATIBLE

Explanation: The qualifications are not on the same prefix chain.

SIM515 F LINE nnnn INTERNAL ERROR - XXXX IS UNKNOWN KEY SYMBOL

User response: Follow report procedure.

SIM516 7 LINE nnnn SYSTEM CLASS XXXX CANNOT BE USED FOR PREFIXING

Explanation: See section 4.1.

SIM517 7 LINE nnnn XXXX IS AN ILLEGAL BLOCK PREFIX (DUE TO LOCAL OBJECT(S) IN CLASS XXXX)

SIM518 B LINE nnnn MISPLACED TEXT EXPRESSION BEFORE XXXX

SIM519 B LINE nnnn MISPLACED OBJECT EXPRESSION BEFORE XXXX

SIM51A B LINE nnnn MISPLACED VALUE EXPRESSION BEFORE XXXX

Guide to user: May result from the compiler recovery action in SIM50A or SIM50B.

SIM51B B LINE nnnn MISPLACED NON-OBJECT EXPRESSION BEFORE XXXX

Guide to user: cf. SIM51A.

SIM51C B LINE nnnn MISPLACED NON-REFERENCE EXPRESSION AFTER XXXX

SIM51D B LINE nnnn MISPLACED NON-ARITHMETIC EXPRESSION AFTER XXXX

SIM51E E LINE nnnn OPERAND STACK CAPACITY EXCEEDED - SIMPLIFY EXPRESSION

User response: If error persists after simplification, follow report procedure.

SIM51F F LINE nnnn INTERNAL ERROR - OPERAND STACK UNDERFLOW

User response: Follow report procedure.

SIM520 7 LINE nnnn ATTEMPT TO MATCH VIRTUAL SPECIFICATION XXXX OF XXXX BY A FORMAL PARAMETER

Compiler action: The formal parameter will become invisible, but the virtual specification will be further matchable.

SIM521 B LINE nnnn IDENTIFIER XXXX IN OBJECT GENERATOR IS NOT A CLASS IDENTIFIER

SIM522 F LINE nnnn INTERNAL ERROR - SYMBOL 'SUBLQL' SCANNED IN 5B

User response: Follow report procedure.

SIM523 F LINE nnnn INTERNAL ERROR - OPERAND STACK DATA ENTRY hhhhhhhh UNSTACKED

User response: Follow report procedure.

SIM524 F LINE nnnn INTERNAL ERROR - SYMBOL 'OTHR' SCANNED IN 5B

User response: Follow report procedure.

SIM525 7 LINE nnnn IDENTIFIER XXXX IS UNDECLARED

Compiler action: A dummy declaration is inserted, which may cause other errors (51A, 51B, 51C).

SIM526 B LINE nnnn MISPLACED TEXT OR REFERENCE EXPRESSION AFTER XXXX

SIM527 B LINE nnnn IMPROPER TYPE IN XXXX RELATION

Explanation: A binary relation had incompatible operands.

SIM528 B LINE nnnn DYNAMIC QUALIFICATIONS XXXX AND XXXX IN REFERENCE RELATION DISAGREE

Explanation: Either the qualifications were not on the same prefix chain or at least one of the qualifications had become visible by connection.

SIM529 F LINE nnnn INTERNAL ERROR - XXXX IS INVALID CONTROL TABLE INDEX

User response: Follow report procedure.

SIM52A F LINE nnnn INTERNAL ERROR - XXXX FOLLOWS CLASS ID IN OBJECT GENERATOR

User response: Follow report procedure.

SIM52B B LINE nnnn NUMBER OF ACTUAL PARAMETERS (nn) NOT EQUAL TO NUMBER OF FORMAL PARAMETERS (nn) OF XXXX

SIM52C 3 LINE nnnn MATCH TO VIRTUAL REF SPECIFICATION XXXX IN CLASS XXXX IS INCORRECTLY QUALIFIED - SPECIFIED QUALIFICATION ASSUMED

Explanation: See section 4.2.

SIM52D B LINE nnnn 'THIS XXXX' IS AN INVALID LOCAL OBJECT

Explanation: The local object did not occur inside a class body or connection block qualified by XXXX.

SIM52E 7 LINE nnnn ILLEGAL USE OF LOCAL OBJECT ('THIS XXXX') IN A BLOCK PREFIXED BY XXXX

Explanation: A reference expression must not evaluate to a reference to a prefixed block.

SIM52F B LINE nnnn INCOMPATIBLE TRUE/FALSE CLAUSES IN A CONDITIONAL EXPRESSION

SIM530 B LINE nnnn XXXX IS NOT A CLASS IDENTIFIER

Explanation: The identifier was used as prefix or after 'QUA', 'IS', 'IN' or 'WHEN'.

Compiler action: Prefix or expression ignored.

SIM531 7 LINE nnnn ILLEGAL USE OF DOT NOTATION, CLASS XXXX CONTAINS LOCAL CLASSES

SIM532 B LINE nnnn XXXX IS NOT AN ATTRIBUTE OF CLASS XXXX

Explanation: In a <remote identifier> the identifier after the dot was not an attribute of the object reference expression before the dot.

SIM533 B LINE nnnn XXXX IS NOT A TEXT ATTRIBUTE IDENTIFIER

Explanation: A text expression followed by a dot was not followed by a text attribute.

SIM534 E LINE nnnn REDECLARATION STACK OVERFLOW

Explanation: The sum of the block level and the forstatement nesting level is greater than 30. See section 3.3.

SIM535 E LINE nnnn CLASS XXXX HAS MORE THAN 127 PARAMETERS (INCLUDING THOSE IN PREFIXES)

Explanation: See section 3.3.

SIM536 E LINE nnnn CLASS XXXX HAS MORE THAN 255 VIRTUAL SPECIFICATIONS (INCLUDING THOSE IN PREFIXES)

Explanation: See section 3.3.

SIM537 B LINE nnnn BLOCK PREFIX XXXX IS NOT A CLASS

Compiler action: Prefix ignored.

SIM538 W LINE nnnn MATCHING QUANTITY XXXX IN CLASS XXXX HAS INCONSISTENT NUMBER OF PARAMETERS OR SUBSCRIPTS

Explanation: Two matches of the same virtual procedure or array specification had different number of parameters/subscripts.

Guide to user: This message shows that the quantity must be used with care if execution errors are to be avoided.

SIM539 W LINE nnnn INCONSISTENCY BETWEEN ACTUAL PARAMETER COUNT (nn) AND FORMAL PARAMETER COUNT (nn) IN MATCH TO VIRTUAL PROCEDURE XXXX

Guide to user: When this message appears, but not SIM538, then the program certainly contains parts that can never be executed without a run-time error.

SIM53A B LINE nnnn MIXED TYPES IN REFERENCE ASSIGNMENT

Explanation: REF and TEXT types were mixed.

SIM53B W LINE nnnn CALL ON UNMATCHED VIRTUAL PROCEDURE XXXX

Explanation: There was no match at this access level. Warning only.

SIM53C B LINE nnnn ATTEMPT TO ACTIVATE A NON-PROCESS OBJECT OF CLASS XXXX

SIM53D E LINE nnnn DYNAMIC BLOCK LEVEL GREATER THAN 15

Explanation: See section 3.3.

SIM53E 7 LINE nnnn MATCHING QUANTITY XXXX IN BLOCK PREFIXED BY XXXX IS OF IMPROPER KIND

Compiler action: Same as SIM50C.

SIM53F 7 LINE nnnn MATCHING QUANTITY XXXX IN BLOCK PREFIXED BY XXXX IS OF IMPROPER TYPE

Compiler action: Same as SIM50C.

SIM540 7 LINE nnnn MATCHING PROCEDURE XXXX IN BLOCK PREFIXED BY XXXX IS NOT SUBORDINATE

Compiler action: Same as SIM50C.

SIM541 7 LINE nnnn ILLEGAL BLOCK PREFIX - XXXX CONTAINS LOCAL OBJECTS (IN CLASS XXXX AMONGST OTHERS)

SIM542 7 LINE nnnn MATCH XXXX IN CLASS XXXX REJECTED DUE TO REDECLARATION OF SPECIFIED QUALIFYING CLASS

Explanation: The qualifying class of a virtual specification is no longer visible, so no further matches can be requested.

SIM543 7 LINE nnnn INCORRECTLY QUALIFIED MATCH XXXX IN CLASS XXXX (NOTE THAT VIRTUAL SPECIFICATION XXXX IS UNMATCHABLE)

Explanation: The specified qualifying class is invisible and therefore the specification is unmatchable at this point.

SIM544 B LINE nnnn MISPLACED CLASS IDENTIFIER XXXX

SIM545 B LINE nnnn ACTUAL PARAMETER TO PROCEDURE/CLASS XXXX HAS TYPE INCOMPATIBLE WITH THAT OF CORRESPONDING FORMAL PARAMETER XXXX

Guide to user: Check that an array, switch or procedure was not passed with associated subscript(s), index or parameter(s) if the formal parameter was specified array, switch or procedure.

SIM546 7 LINE nnnn CLASS SIMSET/SIMULATION USED AS A PREFIX WITHIN BLOCK OR CLASS PREFIXED BY SIMSET/SIMULATION

Explanation: See section 4.1.

SIM547 7 LINE nnnn BLOCK PREFIX XXXX BECAME VISIBLE THROUGH CONNECTION

SIM548 3 LINE nnnn MATCH TO VIRTUAL REF SPECIFICATION XXXX IN BLOCK PREFIXED BY XXXX IS WRONGLY QUALIFIED - SPECIFIED QUALIFICATION IS ASSUMED

Explanation: See section 4.2.

SIM549 7 LINE nnnn MATCH XXXX IN BLOCK PREFIXED BY XXXX IS REJECTED DUE TO REDECLARATION OF SPECIFIED QUALIFYING CLASS

Explanation: See SIM542.

SIM54A 7 LINE nnnn INCORRECTLY QUALIFIED MATCH XXXX IN BLOCK PREFIXED BY XXXX (NOTE THAT VIRTUAL SPECIFICATION XXXX IS UNMATCHABLE)

Explanation: See SIM543.

SIM54B W LINE nnnn MATCHING QUANTITY XXXX IN BLOCK PREFIXED BY XXXX HAS INCONSISTENT NUMBER OF PARAMETER OR SUBSCRIPTS

Explanation: See SIM538.

SIM54C 7 LINE nnnn MATCHING PROCEDURE XXXX IN CLASS XXXX IS NOT SUBORDINATE (PROBABLY DUE TO A PREFIX LOOP)

Explanation: Section 4.2.

SIM54D W LINE nnnn COMPATIBILITY OF QUALIFICATIONS XXXX AND XXXX MUST BE CHECKED AT RUN-TIME

Explanation: In a reference assignment (A:-X) the qualification of the identifier A is a subclass of the qualification of the expression X.

Guide to user: The run-time check will degrade program performance. If appearing in an "inner loop" it could pay off to alter the qualification of A, but this may imply other changes in the program.

SIM54E 7 LINE nnnn MATCHING PROCEDURE XXXX IN CLASS XXXX IS NOT SUBORDINATE (PROBABLY DUE TO USE OF UNDECLARED PREFIX)

Explanation: Section 4.2.

SIM54F 7 LINE nnnn (IN LOWER BOUND OF ARRAY XXXX DECLARATION) - IDENTIFIER XXXX IS UNDECLARED

Compiler action: cf. SIM525.

Guide to user: nnnn is the cardnumber of the left bracket of the array bounds list.

SIM550 7 LINE nnnn MATCHING PROCEDURE XXXX IN CLASS XXXX IS NOT SUBORDINATE (PROBABLY DUE TO USE OF NON-CLASS PREFIX)

Compiler action: cf. SIM50C.

SIM551 7 LINE nnnn (IN UPPER BOUND OF ARRAY XXXX DECLARATION) - IDENTIFIER XXXX IS UNDECLARED

Compiler action: cf. SIM525.

Guide to user: cf. SIM54F.

SIM552 B LINE nnnn MISPLACED OBJECT EXPRESSION AFTER ':='

SIM553 B LINE nnnn MISPLACED VALUE EXPRESSION AFTER ':-'

SIM554 E LINE nnnn CAPACITY ERROR - FREE STORAGE AREA TOO SMALL TO CONTAIN DICTIONARY

User response: Try to compile in a larger core area. If no more core is available, the number of differently spelled identifiers must be decreased by parallell and nested redeclarations.

SIM555 B LINE nnnn (IN LOWER ARRAY BOUND) - NO. ACTUAL PARMS (nn) NOT EQUAL TO NO. FORMAL PARMS (nn) OF XXXX

Guide to user: cf. SIM54F.

SIM556 B LINE nnnn (IN UPPER ARRAY BOUND) - NO. ACTUAL PARMS (nn) NOT EQUAL TO NO. FORMAL PARAMS (nn) OF XXXX

Guide to user: cf. SIM54F.

SIM557 B LINE nnnn (IN LOWER BOUND nn OF ARRAY XXXX DECLN) - MICPLACED TEXT EXPRESSION

Guide to user: cf. SIM54F.

SIM558 B LINE nnnn (IN UPPER BOUND nn OF ARRAY XXXX DECLN) - MISPLACED TEXT EXPRESSION

Guide to user: cf. SIM54F.

SIM559 B LINE nnnn (IN LOWER BOUND nn OF ARRAY XXXX DECLN) - MISPLACED OBJECT EXPRESSION

Guide to user: cf. SIM54F.

SIM55A B LINE nnnn (IN UPPER BOUND nn OF ARRAY XXXX DECLN) - MISPLACED OBJECT EXPRESSION

Guide to user: cf. SIM54F.

SIM55B W LINE nnnn (IN LOWER ARRAY BOUND) - MATCH TO VIRTUAL PROCEDURE XXXX HAS INCONSISTENT ACTUAL/FORMAL PARAMETER COUNTS (nn/nn)

Guide to user: cf. SIM539, SIM54F

SIM55C W LINE nnnn (IN UPPER ARRAY BOUND) - MATCH TO VIRTUAL PROCEDURE XXXX HAS INCONSISTENT ACTUAL/FORMAL PARAMETER COUNTS (nn/nn)

Guide to user: cf. SIM539, SIM54F

SIM55D W LINE nnnn (IN LOWER BOUND OF ARRAY XXXX DECLARATION) - CALL ON UNMATCHED VIRTUAL PROCEDURE XXXX

Explanation: cf. SIM53B.

Guide to user: cf. SIM54F.

SIM55E W LINE nnnn (IN UPPER BOUND OF ARRAY XXXX DECLARATION) - CALL ON UNMATCHED VIRTUAL PROCEDURE XXXX

Explanation: cf. SIM53B.

Guide to user: cf. SIM54F.

SIM55F B LINE nnnn ACTUAL PARAMETER TO STANDARD PROCEDURE/CLASS XXXX HAS TYPE INCOMPATIBLE WITH THAT OF CORRESPONDING FORMAL PARAMETER

SIM560 B (IN ARRAY XXXX BOUNDS) - IDENTIFIER XXXX IS INVALID: DECLARED AT SAME BLOCK LEVEL AS ARRAY XXXX

SIM561 B ACTUAL PARM TO PROC/CLASS XXXX, CORRG TO FORMAL PARM XXXX, SHOULD NOT INCLUDE PARMS/SUBSCRIPTS/INDECES

Explanation: The corresponding formal parameter is a procedure, array or switch. Parameters, subscripts or indeces should not be supplied with the actual parameters.

SIM562 B ACTUAL PARM TO STANDARD PROC/CLASS SHOULD NOT INCLUDE PARMS/SUBSCRIPTS/INDECES

Explanation: cf. SIM561.

SIM563 B MISPLACED TEXT EXPRESSION AFTER 'GOTO'

SIM564 B MISPLACED OBJECT EXPRESSION AFTER 'GOTO'

SIM565 E BLOCK SIZE EXCEEDS 4096 BYTES

Explanation: This implementation restricts the maximum block size to 4096 bytes (see Section 3.2.)

SIM566 W FURTHER OCCURRENCE OF IDENTIFIER XXXX (PREVIOUSLY UNDECLARED AND GIVEN A DUMMY DECLARATION)

Guide to user: On the first occurrence of an undeclared identifier, message SIM525 is issued. On subsequent occurrences, this message (SIM566) is issued (up to 10 times). If the identifier occurs more than 10 times, message SIM567 is issued and further occurrences are not noted.

SIM567 W FURTHER OCCURENCE OF IDENTIFIER XXXX (PREVIOUSLY UNDECLARED) - FURTHER WARNINGS FOR THIS ID ARE SUPPRESSED

Explanation: see SIM566.

SIM568 F LINE nnnn PAGING ERROR - SYSUT4 I/O ERROR

User response: Check the DD-card for SYSUT4. Note that a preallocated data set can not be used. If no error is found, consult a systems programmer.

SIM569 F LINE nnnn PAGING ERROR - SYSUT4 DD CARD MISSING

Explanation: A DD-card for SYSUT4 had not been supplied, and the compiler tables and stacks were too large to be held in core.

User response: Supply the DD-card or re-submit the job to be run in a larger region.

SIM56A F LINE nnnn PAGING ERROR - WRONG PAGE LOADED FROM SYSUT4

User response: Follow report procedure.

SIM56B F LINE nnnn PAGING ERROR - INVALID LOAD FROM VIRTUAL MEMORY, VIRTUAL ADDRESS = hhhhhhhh

User response: Follow report procedure.

SIM56C E LINE nnnn PAGING ERROR - CORE STORAGE TOO SMALL: COMPILATION TERMINATED

User response: Run in a larger core area. If error persists, consult a systems programmer.

SIM56D F LINE nnnn PAGING ERROR - VIRTUAL ADDRESS hhhhhhhh OUT OF RANGE

User response: Follow report procedure.

SIM56E F LINE nnnn PAGING ERROR - ATTEMPT TO STORE INTO FIXED PART OF VIRTUAL MEMORY AT ADDRESS hhhhhhhh

User response: Follow report procedure.

SIM56F B LINE nnnn MISPLACED NON-PROCEDURE IDENTIFIER xxxxxxxx

Explanation: Only a procedure identifier can form a statement on its own.

SIM570 W LINE nnnn INCONSISTENCY BETWEEN PARAMETER/SUBSCRIPT COUNTS IN OCCURENCES OF PROCEDURE/ARRAY xxxxxxxx CALLED BY NAME

SIM571 7 LINE nnnn INCONSISTENCY BETWEEN PARAMETER/SUBSCRIPT COUNTS IN OCCURRENCES OF PROCEDURE/ARRAY xxxxxxxx

SIM572 3 LINE nnnn STANDARD PROCEDURE xxxxxxxx MAY NOT BE PASSED AS A PARAMETER TO PROCEDURE yyyyyyyy

Messages from Pass 7:

SIM700 F LINE nnnn INTERNAL ERROR xxxxxxxx

User response: Please report to NCC with the listing demonstration the error.

SIM701 3 LINE nnnn NON-BOOLEAN OPERAND XXXX IN BOOLEAN EXPRESSION

Compiler action: Operand is replaced by value 'FALSE'.

SIM702 3 LINE nnnn OPERAND XXXX NOT SIMPLE IN EXPRESSION

Explanation: A procedure, array or class identifier was used without parameters or subscripts in an expression.

Compiler action: Operand is replaced by a constant (NONE, NOTEXT, FALSE, CHAR(0) or a zero of appropriate type).

SIM703 3 LINE nnnn LEFT (XXXX) AND RIGHT (XXXX) PART OF ASSIGNMENT INCOMPATIBLE

Explanation: Types were incompatible.

Compiler action: The assignment is ignored.

SIM704 7 LINE nnnn ASSIGNMENT RIGHT HAND SIDE ILLEGAL

SIM705 3 LINE nnnn IMPROPER TYPE OF OPERAND (XXXX) IN ARITHMETIC EXPRESSION

Explanation: An arithmetic operator (+, -, *, //, /, or **) had a non-arithmetic operand.

Compiler action: The resulting operand is replaced by zero.

SIM706 3 LINE nnnn IMPROPER TYPE OF OPERAND (XXXX) IN VALUE RELATION

Explanation: An operand of a value relation was not of value type (arithmetic, CHARACTER or TEXT).

Compiler action: The relation is replaced by 'FALSE'.

SIM707 3 LINE nnnn BOOLEAN OPERAND (XXXX) IN VALUE RELATION

Compiler action: The relation is replaced by 'FALSE'.

SIM708 3 LINE nnnn IMPROPER TYPE OF OPERAND (XXXX) IN REFERENCE RELATION

Compiler action: The relation is replaced by 'FALSE'.

SIM709 3 LINE nnnn MIXED TYPES IN REFERENCE RELATION

Explanation: Types TEXT and REF were mixed.

Compiler action: The relation is replaced by 'FALSE'.

SIM70A B LINE nnnn CONTROLLED VARIABLE (XXXX) WAS NOT VALUE TYPE

Compiler action: A dummy LONG REAL is used which may trigger other messages in the FOR-list.

SIM70B B LINE nnnn CONTROLLED VARIABLE (XXXX) WAS NOT REF

Compiler action: A dummy REF is used. cf. SIM70A.

SIM70C E LINE nnnn CONSTANT TABLE OVERFLOW - SIMPLIFY SOURCE CODE

Explanation: More than 16 constants were used in one statement.

SIM70D F LINE nnnn FEATURE NOT IMPLEMENTED XXXXXXXX

User response: Follow report procedure.

SIM70E 3 LINE nnnn A TEXT VALUE IS BEING ASSIGNED TO NON-TEXT VARIABLE XXXX

Compiler action: Assignment by-passed.

SIM70F 3 LINE nnnn GO TO OPERAND (XXXX) IS NOT SIMPLE

Compiler action: GO TO statement is ignored.

SIM710 3 LINE nnnn GO TO OPERAND (XXXX) IS NOT LABEL

Compiler action: GO TO statement is ignored.

SIM711 3 LINE nnnnn TYPES IN DENOTES ARE INCOMPATIBLE

SIM712 3 LINE nnnn LEFT SIDE OF DENOTES (XXXX) IS NOT REF OR TEXT

SIM713 3 LINE nnnn SWITCH ELEMENT NOT SIMPLE

SIM714 3 LINE nnnn SWITCH ELEMENT (XXXX) NOT OF TYPE LABEL

SIM715 3 LINE nnnn EXPRESSION SUCCEEDING 'WHILE' (XXXX) WAS NOT BOOLEAN

SIM716 3 LINE nnnn OPERAND SUCCEEDING 'WHILE' (XXXX) WAS NOT SIMPLE

SIM717 3 LINE nnnn IMPROPER USE OF AN ARRAY IDENTIFIER XXXX

Explanation: No subscripts were used.

SIM718 3 LINE nnnn EXPRESSION BETWEEN 'IF' AND 'THEN' WAS NOT BOOLEAN

Compiler action: 'FALSE' is used.

SIM719 3 LINE nnnn EXPRESSION FOLLOWING 'NOT' WAS NOT BOOLEAN

SIM71A 3 LINE nnnn PARAMETER TO 'OUTCHAR' WAS NOT OF CHARACTER TYPE

SIM71B 3 LINE nnnn FIRST PARAMETER TO 'DISCRETE' OR 'HISTD' WAS NOT ARRAY

SIM71C 3 LINE nnnn ARRAY PARAMETER (XXXX) TO 'DISCRETE' OR 'HISTD' WAS NOT OF TYPE REAL

SIM71D 3 LINE nnnn THE LAST PARAMETER TO A RANDOM DRAWING PROCEDURE WAS INCORRECT

Explanation: The last parameter to a random drawing procedure was an expression.

SIM71E 3 LINE nnnn THE LAST PARAMETER (XXXX) TO A RANDOM DRAWING PROCEDURE WAS NOT SIMPLE

SIM71F 3 LINE nnnn THE LAST PARAMETER (XXXX) TO A RANDOM DRAWING PROCEDURE WAS NOT OF TYPE INTEGER

SIM720 3 LINE nnnn IMPROPER PARAMETER (XXXX) TO 'ENTIER'

SIM721 W LINE nnnn PARAMETER XXXX TO ENTIER WAS INTEGER

SIM722 3 PARAMETER TO 'RANK' WAS NOT OF CHARACTER TYPE

SIM723 3 NON-ARITHMETIC PARAMETER (XXXX) TO 'SIGN'

SIM724 3 PARAMETER XXXX TO MATHEMATICAL FUNCTION WAS NOT ARITHMETIC

SIM725 3 LINE nnnn PARAMETER XXXX TO 'OUTCHAR' WAS NOT SIMPLE

SIM726 3 LINE nnnn IMPROPER TEXT REFERENCE

SIM727 3 LINE nnnn A RESULT IS BEING USED AS A TEXT REFERENCE

SIM728 3 LINE nnnn TYPE OF ACTUAL CONSTANT PARAMETER CORRESPONDING TO FORMAL XXXX WAS BOOLEAN OR CHARACTER

SIM729 W LINE nnnn ZERO CONSTANT STEP IN STEP-UNTIL CLAUSE

SIM72A 3 LINE nnnn TEXT CONSTANT ILLEGAL AS NAME PARAMETER

SIM72B 3 LINE nnnn IMPROPER PARAMETER xxxxxxxx TO FORTRAN PROCEDURE

Explanation: Note the following restrictions apply for parameters to FORTRAN procedures:
- Only identifiers or literals can be used as a parameter.
- Parameter type must not be ref, label or character.

SIM74E 3 LINE nnnn IMPROPER USE OF NOTYPE PROCEDURE xxxxxxxx

SIM74F 3 LINE nnnn REGISTER NOT RELEASED XXXXXXXX - INTERNAL ERROR

Follow report procedure.

SIM750 3 LINE nnnn SIMPLIFY STATEMENT, TOO MANY FORMAL PARAMETERS

User response: Simplify the expression by reducing the number of left hand side formal parameters or simplify the levels of nesting (parentheses or implied parentheses) on the right hand side.

SIM751 3 LINE nnnn SIMPLIFY EXPRESSION, TOO MANY REFERENCE QUANTITIES

SIM752 3 LINE nnnn SIMPLIFY STATEMENT, TOO MANY TEXT QUANTITIES

User response: Simplify by reducing the number of multiple text assignments or reduce complexity of text expression.

SIM753 3 LINE nnnn SIMPLIFY EXPRESSION, TOO MANY NESTED VALUE QUANTITIES

User response: Simplify the expression by reducing the amount of nesting (or implied nesting) of value quantities. If necessary, make the calculation in two or more steps.

SIM754 3 LINE nnnn THE RESULT OF A MULTIPLICATION OF TWO INTEGER CONSTANTS WAS TOO LARGE TO BE CONTAINED IN AN INTEGER

User response: One of the constants should be changed to type real.

SIM755 W LINE nnnn I GREATER THAN 31 IN CONSTANT EXPRESSION 2**I

SIM756 W LINE nnnn X**1 IS X

SIM757 W LINE nnnn EQUALITY/INEQUALITY RELATION CONTAINING REAL QUANTITIES MAY NOT BE MEANINGFUL

SIM75B F LINE nnnn OPERAND STACK UNDERFLOW - INTERNAL ERROR

SIM75C E LINE nnnn EXTERNAL SYMBOL TABLE OVERFLOW - CAPACITY ERROR

Explanation: The present implementation does not allow more external names. This also includes the external names in the run-time system for the modules which have been included by the compiler.

SIM75D 3 LINE nnnn ILLEGAL VALUE ASSIGNMENT OF xxxx TO yyyy

SIM75E 3 LINE nnnn ILLEGAL REFERENCE RELATION - xxxx CANNOT BE COMPARED WITH yyyy

SIM75F 3 LINE nnnn ATTEMPT TO ASSIGN A VALUE TO PROCEDURE IDENTIFIER xxxx OUTSIDE THE BODY OF xxxx

SIM760 3 LINE nnnn ATTEMPT TO ASSIGN A REFERENCE TO PROCEDURE IDENTIFIER xxxx OUTSIDE THE BODY OF xxxx

SIM761 3 LINE nnnn WRONG NUMBER OF SUBSCRIPTS (kkkk) OF ARRAY aaaa - aaaa WAS DECLARED AS mmmm-DIMENSIONAL ARRAY

SIM762 3 LINE nnnn TOO MANY INDICES (mmmm) FOR SWITCH ssss - A SWITCH MAY ONLY HAVE ONE INDEX

SIM763 3 LINE nnnn EXTERNAL FORTRAN PROCEDURE xxxx CANNOT BE OF TEXT TYPE.

SIM764 3 LINE nnnn ILLEGAL OCCURENCE OF A LABEL/SWITCH IDENTIFIER xxxx AT THE LEFT HAND SIDE OF AN ASSIGNMENT

SIM765 3 LINE nnnn TYPE OF ACTUAL ARRAY PARAMETER xxxx IS NOT IDENTICAL TO THE TYPE OF THE CORRESPONDING FORMAL ARRAY yyyy

Explanation: Types of array parameters must coincide.

SIM766 3 LINE nnnn FORMAL TEXT PARAMETER xxxx OF PROCEDURE/CLASS aaaa MAY NOT HAVE A STRING AS ACTUAL PARAMETER IN DEFAULT MODE

SIM767 3 LINE nnnn ILLEGAL OCCURENCE OF A NON-ARITHMETIC CONSTANT

SIM768 3 LINE nnnn ACTUAL PARAMETER xxxx IS NOT AN ARRAY, BUT THE CORRESPONDING FORMAL yyyy WAS SPECIFIED AS AN ARRAY

SIM769 3 LINE nnnn FORMAL PARAMETER xxxx AND ACTUAL PARAMETER yyyy IN VALUE/DEFAULT MODE ARE OF INCOMPATIBLE TYPES

SIM76A E LINE nnnn TOO MANY NESTED PROCEDURE CALLS - CAPACITY ERROR

SIM76B 3 LINE nnnn LABEL/SWITCH xxxx MAY NOT BE USED REMOTELY

SIM76C 3 LINE nnnn FORMAL PARAMETER xxxx AND ACTUAL PARAMETER yyyy ARE OF INCOMPATIBLE KINDS

SIM76D 3 LINE nnnn FORMAL PARAMETER xxxx AND ACTUAL PARAMETER yyyy ARE OF INCOMPATIBLE TYPES

SIM76E 3 LINE nnnn REMOTE CALL ON EXTERNAL PROCEDURE xxxx

Explanation: It makes little sense to distinguish between external attribute belonging to different objects of some class - they are all identical.

SIM76F 7 LINE nnnn MISPLACED SUBSCRIPT EXPRESSION FOLLOWING SIMPLE OPERAND xxxx

SIM770 F LINE nnnn PROGRAM COMPILATION IMPROPERLY ENDED - INTERNAL ERROR

SIM771 F LINE nnnn EXTERNAL SYMBOL xxxx NOT FOUND IN ESD - INTERNAL ERROR

SIM772 7 LINE nnnn FIRST TWO PARAMETERS TO HISTO MUST BE OF KIND ARRAY

SIM773 7 LINE nnnn LAST TWO PARAMETERS TO HISTO MUST BE SIMPLE OPERANDS

SIM774 7 LINE nnnn FIRST TWO PARAMETERS TO HISTO MAY ONLY BE OF INTEGER, SHORT INTEGER OR REAL TYPE

SIM775 7 LINE nnnn LAST TWO PARAMETERS TO HISTO MUST BE OF ARITHMETIC TYPE

SIM776 7 LINE nnnn THE TYPE OF THE ACTUAL ARRAY xxxx IS NOT IDENTICAL WITH THAT OF THE FORMAL ARRAY

SIM777 7 LINE nnnn xxxx IS ILLEGAL CONTROLLED VARIABLE

Explanation: The controlled variable cannot be a name parameter or procedure identifier.

SIM778 3 LINE nnnn CONSTANT ARITHMETIC ERROR : xxxxxxxx

SIM779 W LINE nnnn xxxxxxxx IS TO LONG FOR AN EXTERNAL NAME, ONLY SEVEN CHARACTERS WERE ACCEPTED

SIM77A W LINE nnnn ACTUAL PARAMETER NUMBER mmmm OF EXTERNAL ASSEMBLY/FORTRAN PROCEDURE xxxx IS AN EXPRESSION. THIS MAY CAUSE ERROR ZYQ062 UNDER EXECUTION.

SIM77B 3 LINE nnnn jjjj CANNOT BE CONVERTED TO iiii BECAUSE IT IS OUT OF iiii RANGE

Explanation: Compiler attempt to convert a (long) real/integer literal value to integer/short integer type failed because the former has too large a magnitude.

Compiler action: Magnitude 1 is assumed.

SIM77C W LINE nnnn WAS USE OF THE CONNECTED LABEL xxxx INTENDED ?

Explanation: A label which is visible through inspection occurred in a goto-statement or a switch element list. Unless the inspected object is also operating the use of this label will lead to ZYQ007 error in RT.

SIM77D W LINE nnnn SYMBDUMP VALUE TOO HIGH WITH RESPECT TO CORE AVAILABLE. INCREASE SYSFREE SUBPARAMETER OF SIZE (AND REGION?) BY nK.

Explanation: Either follow this instruction (note that SYSFREE is the third subparameter to SIZE option - usually 4K in default) or eventually recompile with SYMBDUMP<3.

SIM77E 3 LINE nnnn DUE TO A STRING IN ONE OF BRANCHES, THE CONDITIONAL TEXT VALUE CANNOT BE USED IN A REFERENCE CONTEXT

Explanation: An IF-THEN-ELSE construction with at least one of the branches being a string is taken as a conditional text value (i.e. "conditional string") and as such cannot be used where string could be illegal.

SIM7FF B LINE nnnn REG AND REGFLAG INCOMPATIBLE XXXXXXXX - INTERNAL ERROR

Follow report procedure.

Messages from Pass 8:

SIM800 F LINE nnnn NON-EXISTENT FIXUP - INTERNAL ERROR

SIM801 F LINE nnnn COMPILER CAPACITY EXCEEDED XXXX

Rerun with partition size increased by at least XXXX (hexadecimal) bytes.

SIM802 F LINE nnnn COMPILER CAPACITY EXCEEDED XXXX

Rerun with partition size increased by at least XXXX (hexadecimal) bytes.

RUN-TIME DIAGNOSTICS

1) Messages.

Information messages are identified by message numbers 900-999.

ZYQ999 PSW = hhhhhhhh hhhhhhhh ********

Explanation: This is the first line printed after a program interruption, and gives the old Program Status World at the interrupt. The succeeding line will give the reasion for the interrupt.

ZYQ998 DDNAME = cccccccc

Explanation: This message identifies the file on which the error listed on the preceding line occurred.

ZYQ997 LENGTH = dddd POS = dddd CONTENTS: /*ccc...c*/

Explanation: This message gives length, pos and the first characters of a text on which a local standard procedure failed. The preceding line indicates the kind of error.

ZYQ996W dddd EDIT OVERFLOWS HAVE OCCURRED

Explanation: This message is printed at end of program execution if one or more edit overflows have occurred.

ZYQ995 SYNAD MESSAGE: ccc...

Explanation: This message is edited by the control program and gives the reason for an I/O error that has occurred (see (8)).

2) Diagnostics

ZYQ000 INTERNAL ERROR

Explanation: An error has been detected that should not occur, possibly because of an error in the compiler or runtime system.

User response: If there are non-Simula external procedures in the program, these should be carefully checked, otherwise follow the standard report procedure.

ZYQ001 CALL VIRTUAL PRO

Explanation: A virtual procedure, called in the class body, had no match in the object in which it was called.

ZYQ002 CON. VIRT. PROC.

Explanation: A virtual procedure, called in a connection block, had no match in the connected object.

ZYQ003 REM. VIRT. PROC.

Explanation: A virtual procedure, called by remote referencing, had no match.

ZYQ004 RESUME TERMINATE

Explanation: The parameter of RESUME was a terminated object.

ZYQ005 NO VIRTUAL MATCH

Explanation: A virtual quantity, except for those identified by messages 1-4 and 9, had no match in the referenced object.

ZYQ006 CALL TERMINATED

Explanation: The parameter of CALL was a terminated object.

ZYQ007 UNDEFINED GOTO

Explanation: A goto statement led into an object not in the operating chain (e.g. a transfer from a procedure called by remote referencing or connection into the class object).

ZYQ008 ILLEGAL GOTO

Explanation: A goto statement led out of a detached object, but not out of the quasi-parallel system of that object (see (12), 9.4.2).

ZYQ009 VIRTUAL LABEL

Explanation: A virtual label mentioned in a goto statement had no match in the object.

ZYQ010 QUA ERROR

Explanation: An instantaneous qualfication failed, i.e. the object expression was not identical to or a subclass of the class mentioned after QUA.

ZYQ011 REF ASSIGNMENT

Explanation: An implicit qua check in a reference assignment failed.

ZYQ012 CHAR OUT OF RANG

Explanation: The parameter to CHAR was <0 or >255, i.e. it did not correspond to any character.

ZYQ013 ARRAY DECL. L>U

Explanation: The upper bound was less than the lower bound in a bound pair of an array declaration.

ZYQ014 ARRAY BOUNDS ERR

Explanation: A subscripted variable referred to an address outside the array, i.e. one or more subscripts were out of range.

ZYQ015 CONVERSION RANGE

Explanation: A real quantity could no be converted to integer because it was out of integer range.

ZYQ016 TEXT VALUE ASSGN

Explanation: A text value assignment was illegal because the length of the right hand side was greater than the length of the left hand side.

ZYQ017 STORAGE EXHAUSTD

Explanation: More storages was requested than was available in the SIMULA working storage pool, i.e. the parameter q(3) to the object program was exceeded (2.2.3.1)

User response: Check whether the program seems to have executed correctly. If so, specify a smaller value for q(2) or request a larger partition with the REGION operand of the JOB or EXEC statement (MVT, MFT). One may have to request smaller and fewer I/O buffers to execute successfully if q(2) is decreased.

ZYQ018 DATA LIMIT

Explanation: The total size of the declared quantities exceeded the limit q(1) specified to the object program.

ZYQ019 FILES NOT CLOSED

Explanation: One or more files (except for sysout and sysin) were not closed when the program ended.

The last image of a sequential output file may be lost.

ZYQ020 NUMB OF PARAMS

Explanation: The number of actual parameters given in a call of a formal, external or virtual procedure was not the same as the number of formal parameters of the actual procedure, the external procedure or the virtual match.

ZYQ021 TEXT LENGTH

Explanation: BLANKS or INTEXT was called with a parameter which was negative or greater than the maximum length of a text, 2**15 - 20.

ZYQ022 PARAM KINDS 1

Explanation: An actual parameter was a <type> PROCEDURE while the corresponding formal parameter was specified ARRAY, LABEL or SWITCH in a call on a formal, virtual or external procedure.

ZYQ023 PARAM KINDS 2

Explanation: An actual parameter was not a <type> PROCEDURE and the kind of the corresponding formal parameter was incompatible with the actual kind in a call on a formal, virtual or external procedure.

ZYQ024 ACT.TYPE NONARIT

Explanation: A formal parameter was of arithmetic type but the corresponding actual parameter was not.

ZYQ025 PARAM TYPE

Explanation: A formal parameter type was incompatible with the type of the corresponding actual parameter.

ZYQ026 ARRAY TYPES

Explanation: Formal and actual type did not coincide for an arithmetic array parameter, or qualifications did not coincide for a reference mode REF array.

ZYQ027 ACT.NOT SUBORD.

Explanation: For a REF(...)PROCEDURE specified parameter, the actual type was not subordinate to the formal type.

ZYQ028 PARAM QUALIF.

Explanation: The qualification of an actual parameter was incompatible with the qualifications of the corresponding formal parameter.

ZYQ029 EXTERNAL TYPE

Explanation: The type of an external procedure declaration was not the same as that of the procedure.

ZYQ030 NOTEXT EDIT

Explanation: The text of a number editing procedure was notext.

ZYQ031 NUMB. OF SUBSCR.

Explanation: The number of subscripts given in a reference to a formal or virtual array was not the same as the number of subscripts in the actual array or virtual match.

ZYQ032 DDCARD MISSING

Explanation: There was no dd-statement in the job step matching a file that was opened.

User response: Check if the dd-card is there, if not add it. If it is there, check that its ddname is correctly spelled and that it has not been put after a DD * data set if the system is PCP.

ZYQ033 FILE WAS OPEN

Explanation: The parameter of OPEN was a file that was already open.

ZYQ034 FILE CLOSED

Explanation: The parameter of CLOSE was a file that was already closed.

ZYQ035 FILE NOT OPEN

Explanation: When an input or output request was issued on a file, it was not open.

ZYQ036 IMAGE.LENGTH>256

Explanation: The image of a printfile was longer than 256.

ZYQ037 I/O ERROR

Explanation: A permanent I/O error occurred on a file. The preceding line gives the kind of error.

User response: Check that the data set has the correct characteristics for the request, then check the operands of the dd-statement. If no error is found, consult a systems programmer.

ZYQ038 IMAGE TOO SHORT

Explanation: For an infile or a directfile the image was shorter than the record that was to be transmitted.

ZYQ039 END OF FILE

Explanation: The program tried to read past end of file on an infile, or two successive calls on OUTIMAGE/INIMAGE were issued with LOC out of range on a directfile.

ZYQ040 FIELD ERROR

Explanation: The field length parameter of an output procedure was non-positive or greater than the image length.

ZYQ041 EOF IN ININT

Explanation: End-of-file occurred in ININT before a non-blank character was found.

ZYQ042 EOF IN INFRAC

Explanation: Same as ZYQ041, but INFRAC was called.

ZYQ043 EOF IN INREAL

Explanation: Same as ZYQ041, but the INREAL was called.

ZYQ044 IMAGE.NOTEXT

Explanation: When a file was accessed, the specific operation could not be performed because its image was NOTEXT.

ZYQ045 IMAGE TOO LONG

Explanation: When OUTIMAGE was called for an outfile connected to a data set with fixed record length, the image length was greater than the record length, or a directfile was used with an image longer than the blocksize.

ZYQ046 RECORD FORMAT

Explanation: A directfile was connected to a data set which had an illegal record format.

ZYQ047 SPACING ERROR

Explanation: The procedure SPACING was called with a negative parameter.

ZYQ048 POS ERR IN GETCH

Explanation: GETCHAR was called for a text in which pos=length+1.

ZYQ049 POS ERR PUTCHAR

Explanation: Same as ZYQ048 for PUTCHAR.

ZYQ050 NO DIGITS:GETINT

Explanation: GETINT or ININT was called, but the first non-blank character found was not a digit.

ZYQ051 INT RANGE:GETINT

Explanation: More than 15 significant digits were scanned by GETINT or ININT.

ZYQ052 NO DIGIT:GETFRAC

Explanation: GETFRAC or INFRAC was called, but the first non-blank character was not a digit.

ZYQ053 ILL. N:PUTFRAC

Explanation: The parameter n of putfrac was not in $0 =< n =< 12$.

ZYQ054 SHORT FIELD

Explanation: The parameter of PUTFRAC, OUTFRAC, PUTFIX or OUTFIX were such that the decimal point falls outside the text.

ZYQ055 ILL.N:PUTFIX

Explanation: Same as ZYQ058 for PUTFIX or OUTFIX.

ZYQ056 LENGTH ERR: SUB

Explanation: The parameters of SUB specify a text that is not contained in the original text, or the specified length is negative.

ZYQ057 ILLEG I: SUB

Explanation: The first parameter of SUB specified a non-positive starting position for the subtext.

ZYQ058 ILLEG N:PUTREAL

Explanation: PUTREAL or OUTREAL was called with n not within 0 =< n =< 15.

ZYQ059 NO DIGIT:GETREAL

Explanation: GETREAL or INREAL was called, but no digit was found.

ZYQ060 FORT/ASS PRM KND

Explanation: In a call on a Fortran or assembly procedure, an actual parameter was of illegal kind (PROCEDURE or SWITCH). See section 4.16.

ZYQ061 NON-LOCAL LABEL

Explanation: In a call on an assembly procedure, a non-local label was passed as parameter. See section 4.16.

ZYQ062 PARAMETER FORM

Explanation: A parameter to a Fortran or assembly procedure did not have a legal form (Section 4.16).

ZYQ063 TOO MANY PARAMS

Explanation: More than 18 parameters were passed to a Fortran or assembly procedure (Section 3.3).

ZYQ064 EVT OF TERM OBJ

Explanation: On a call Z.EVTIME, Z is a terminated object.

ZYQ065 EVT OF PASS OBJ

Explanation: On a call Z.EVTIME, Z is a passive object.

ZYQ066 PARAMETER TYPE

Explanation: A parameter passed to a Fortran procedure was of illegal type (Section 4.16).

ZYQ067 PASSIV SQS.LAST

Explanation: A call to passivate the last SQS member is illegal.

ZYQ068 REMOVE SQS.LAST

Explanation: A call to remove the last SQS member is illegal.

ZYQ069 RE/ACT NON PROC

Explanation: The object to be activated or reactivated is not a PROCESS object.

ZYQ070 BEF/AFT NON PROC

Explanation: The objects in the BEFORE or AFTER clause of an activation statement is not qualified by PROCESS.

ZYQ071 WAIT: NON HEAD

Explanation: The object H passed as parameter in a call WAIT(H) is not qualified by HEAD.

ZYQ072 CANCEL: NON PROC

Explanation: The object P passed as parameter in a call CANCEL(P) is not qualified by PROCESS.

ZYQ073 RANDINT B<A

Explanation: The upper bound of the interval passed to RANDINT was less than the lower bound.

ZYQ074 ERLANG A<=0

Explanation: The first parameter to ERLANG was non-positive.

ZYQ075 ERLANG B<=0

Explanation: The second parameter to ERLANG was non-positive.

ZYQ076 LINEAR: ARRAYS

Explanation: The two arrays passed to LINEAR were not both REAL, one-dimensional arrays with equal subscript bounds.

ZYQ077 <interrupt cause>

Explanation: A program interrupt occurred. For an addressing interrupt the text 'OBJECT NONE' is supplied.

ZYQ080 POWER OP, BASE=0

Explanation: The base was zero and the exponent non-positive for an INTEGER to INTEGER exponentiation.

ZYQ081 POWER OP, BASE=0

Explanation: Same as ZYQ080 for a REAL to REAL exponentiation.

ZYQ082 POWER OP, BASE=0

Explanation: Same as ZYQ080 for a LONG REAL to LONG REAL exponentiation.

ZYQ083 POWER OP, BASE=0

Explanation: Same as ZYQ080 for a REAL to INTEGER exponentiation.

ZYQ084 POWER OP, BASE=0

Explanation: Same as ZYQ080 for a LONG REAL to INTEGER exponentiation.

ZYQ085 ARCSIN/COS abs(X)>1

Explanation: The double precision parameter passed to ARCSIN or ARCCOS had a modulus greater than 1.

ZYQ086 SINH/COSH X>MAX

Explanation: The parameter of SINH or COSH was too large to be representable as a real quantity (Section 3.1). The argument was in double precision.

ZYQ087 SQRT NEG ARG

Explanation: The double precision parameter to SQRT was negative.

ZYQ088 TAN/COT abs(X)>MAX

Explanation: The double precision parameter to TAN or COT was too large to permit accurate computation of the function value.

ZYQ089 TAN/COT INFINITE

Explanation: The function value of TAN or COT with double precision argument was too large or infinite (Section 3.1).

ZYQ090 ARCSIN/COS abs(X)>1

Explanation: Same as ZYQ085 but argument was in single precision.

ZYQ091 SINH/COSH X>MAX

Explanation: Same as ZYQ086 but argument was in single precision.

ZYQ092 SQRT NEG ARG

Explanation: Same as ZYQ087 but argument was in single precision.

ZYQ093 TAN/COT abs(X)>MAX

Explanation: Same as ZYQ088 but argument was in single precision.

ZYQ094 TAN/COT INFINITE

Explanation: Same as ZYQ089 but argument was in single precision.

ZYQ095 EXP ARG>174.673

Explanation: The value of EXP could not be represented (Section 3.1). Argument was in double precision.

ZYQ096 LOG ARG<=0

Explanation: The double precision argument of LOG was non-positive.

ZYQ097 SIN/COS ARG>MAX

Explanation: The double precision argument of SIN or COS was too large to permit computation of the function value.

ZYQ098 EXP ARG>174.673

Explanation: Same as ZYQ095 with single precision argument.

ZYQ099 LOG ARG<=0

Explanation: Same as ZYQ096 with single precision argument.

ZYQ100 SIN/COS ARG>MAX

Explanation: Same as ZYQ097 with single precision argument.

ZYQ101 ACT.PARAM STRING

Explanation: A found text parameter called by name occurs in a reference context with the actual parameter being a string.

ZYQ102 UNIFORM LB > UB

ZYQ103 TIME LIMIT OVFLW

ZYQ104 FORCED ERROR

ZYQ105 POWER OP, BASE<0

Explanation: The base was negative for a (long) real to (long) real exponentiation.

ZYQ106 MAXPAGES LIMIT

User response: Increase the value of the run time parameter MAXPAGES if more printed output is required.

ZYQ107 EJECT PARM LE 0

Explanation: The procedure EJECT was called with a negative parameter.

ZYQ108 INVALID DDNAME

Explanation: A text parameter to file object does not conform to the operation system conventions for a DDname. Most likely the value is notext.

ZYQ109 TRANSPLANTATION

Explanation: The qualifying class of a reference actual parameter to a virtual, formal or external procedure was declared in another block instance than that of the corresponding formal parameter specification.

ZYQ110 NEGEXP : A<=0

Explanation: First parameter to standard random drawing procedure Negexp was nonpositive.

ZYQ111 RANDOM SEED IS 0

Explanation: Last parameter to a random drawing procedure has value zero and cannot thus be used as a random stream base.

ZYQ112 DETACH INACTIVE

Explanation: The standard procedure Detach was called on behalf on a class object which was not on the operation chain.

ZYQ113 CALL OPERATING

Explanation: The standard procedure Call had as a parameter a reference to an operating class object.

ZYQ114 NESTED SIMSET

Explanation: The SIMSET or SIMULATION environment occurs simultaneously at several block levels due to dynamic nesting of procedure calls. The implementation is no geared to handle several SIMSET/SIMULATION contexts at the same time.

(NB: permitted restriction of the Common Base).

SIZES OF RUN-TIME LIBRARY ELEMENTS

Name	Size (hex)	When present
ZYQACC	8	Logging support
ZYQACCUM	110	Accum
ZYQACTIV	3E8	Scheduling statement
ZYQARRAY	270	Arrays
ZYQATTAC	48	CALL
ZYQATTN	128	When running the interact.deb.sys.
ZYQBC	90	Object generator
ZYQBSC	1B10 1)	Always
ZYQBLANK	110	BLANKS
ZYQBPB	A8	Prefixed block
ZYQCALL	E0	Call
ZYQCANCE	A8	Simulation block
ZYQCCP	38	Call on connected procedure
ZYQCCVP	60	Call on connected virtual procedure
ZYQCDGEN	38	Connected or remote procedure passed as a parameter
ZYQCDP	38	Call on remote procedure
ZYQCDVP	58	Call on remote virtual procedure
ZYQCEP	48	Call on external procedure
ZYQCFP	80	Call on formal procedure
ZYQCLOSE	1B10 1)	Always
ZYQCOM	E20	Always
ZYQCOMUN	98	Always
ZYQCONNE	68	Connection block
ZYQCOPY	C8	COPY
ZYQCSW	90	Switch
ZYQCURRE	10	CURRENT
ZYQCVP	40	Call virtual procedure
ZYQDATE	68	Always
ZYQDBGDT	AF8	SYMBDUMP compile parm > 0
ZYQDEBUG	FC8	SYMBDUMP compile parm > 0
ZYQDEFLT	60	Always
ZYQDETAC	170	DETACH
ZYQDFTRC	818	SYMBDUMP compile parm > 0
ZYQDIREC	580	DIRECTFILE
ZYQDISCR	A8	DISCRETE
ZYQDRAW	40	DRAW
ZYQDUMP	380	Always, DUMP > 3
ZYQDXPD	54	Exponentiation real to real (long)
ZYQDXPI	70	Exponentiation real to integer
ZYQECB	F0	Always
ZYQEJECT	7C	EJECT, LINE, LINESPERPAGE, SPACING
ZYQENT	1B10 1)	Always (program entry point)

Notes

1) Size includes all parts of resident RTS

Name	Size (hex)	When present
ZYQENTVI	5D0	Formal, virtual or external procedure
ZYQEPBPA	50	Prefixed block with parameter(s)
ZYQERLAN	D8	ERLANG
ZYQERR	BB8	Always, error exit routine, DUMP>0
ZYQERRDP	20C	Always, error exit routine, DUMP>0
ZYQERREX	38	Always, error exit routine, DUMP>0
ZYQEVD	60	Same as ZYQENTVI
ZYQEVTIM	30	EVTIME
ZYQFILE	1B10 1)	Always
ZYQFIXPI	78	Exponentiation integer to integer
ZYQFORT	308	Call on external assembly or Fortran procedure
ZYQFSA	1B10 1)	Always (RTS, Fixed Storage Area)
ZYQGC6	200	Always
ZYQGETCH	28	GETCHAR
ZYQGETFR	188	GETFRAC, INFRAC
ZYQGETIN	130	GETINT, ININT
ZYQGETRE	2A0	GETREAL, INREAL
ZYQGL	190	Goto statement
ZYQGVL	38	Goto statement with virtual label
ZYQHISTD	90	HISTD
ZYQHISTO	140	HISTO
ZYQHOLD	E0	HOLD
ZYQIDLE	28	IDLE
ZYQIFEX	1B10 1)	Always
ZYQINCHA	4C	INCHAR, OUTCHAR
ZYQINFIL	1B10 1)	Always
ZYQINFRA	48	INFRAC
ZYQININT	48	ININT
ZYQINIT	B90	Always
ZYQINREA	48	INREAL
ZYQINTEX	208	INTEXT
ZYQLACOS	140	ARCCOS (long)
ZYQLASIN	138	ARCSIN (long)
ZYQLASTI	288	LASTITEM, LETTER, DIGIT, INREAL, INFRAC
ZYQLATAN	128	ARCTAN (long)
ZYQLCOS	158	COS (long)
ZYQLCOSH	108	COSH (long)
ZYQLCOTN	158	COT (long)
ZYQLDIG	104	Always
ZYQLEXP	1B8	EXP (long)
ZYQLINEA	D0	LINEAR
ZYQLLG10	140	LOG (long)
ZYQLLN	40	Always
ZYQLLOG	140	LN (long)
ZYQLNMAP	158	Always, DUMP > 1
ZYQLNO	210	Always, DUMP > 1
ZYQLNODT	50	Always, DUMP > 1
ZYQLSIN	158	SIN (long)
ZYQLSINH	108	SINH (long)
ZYQLSQRT	84	SQRT (long)

Name	Size (hex)	When present
ZYQLTAN	158	TAN (long)
ZYQLTANH	E0	TANH (long)
ZYQMAINT	30	MAIN (text procedure)
ZYQMAP	478	Always, DUMP > 1
ZYQMAPTR	190	Always, DUMP > 1
ZYQMVCL	68	Always
ZYQNEGEX	70	NEGEXP
ZYQNEXTE	78	NEXTEV
ZYQNORMA	E8	NORMAL
ZYQODDNM	70	SYMBDUMP compile parm > 0
ZYQOPEN	1B10 1)	Always
ZYQOUTAR	4F0	SYMBDUMP compile parm > 0
ZYQOUTBL	530	SYMBDUMP compile parm > 0
ZYQOUTCH	70	OUTCHAR
ZYQOUTFR	28	OUTFRAC
ZYQOUTFX	28	OUTFIX
ZYQOUTIM	2A8	Always
ZYQOUTIN	28	OUTINT
ZYQOUTIO	168	SYMBDUMP compile parm > 0
ZYQOUTOC	170	SYMBDUMP compile parm > 0
ZYQOUTPO	108	SYMBDUMP compile parm > 0
ZYQOUTRE	28	OUTREAL
ZYQOUTSQ	2E8	SYMBDUMP compile parm > 0
ZYQOUTTE	90	OUTTEXT
ZYQOUTVI	280	SYMBDUMP compile parm > 0
ZYQPAGE	1B10 1)	Always
ZYQPASSI	98	PASSIVATE
ZYQPFD	68	OUTFRAC, PUTFRAC
ZYQPOISS	158	POISSON
ZYQPRD	98	OUTFIX, OUTREAL, PUTFIX, PUTREAL
ZYQPRED	28	SIMSET/SIMULATION
ZYQPRINT	1B10 1)	Always
ZYQPUTCH	68	PUTCHAR
ZYQPUTFI	320	PUTFIX, OUTFIX
ZYQPUTFR	250	PUTFRAC, OUTFRAC
ZYQPUTIN	228	PUTINT, OUTINT
ZYQPUTRE	338	Always
ZYQQUA	30	Parameter of type REF
ZYQRANDI	48	RANDINT
ZYQRDAT	38	All drawing procedures
ZYQREFER	4A8	Logging support
ZYQRESET	1B10	Always
ZYQRESUM	C8	RESUME
ZYQRTSCM	C40	Always
ZYQRXPI	40	Exponentiation real to real (short)
ZYQSACOS	D8	ARCCOS (short)
ZYQSASIN	D8	ARCSIN (short)
ZYQSATAN	C0	ARCTAN (short)
ZYQSCALE	C0	Always
ZYQSCOS	F0	COS (short)
ZYQSCOSH	C0	COSH (short)
ZYQSCOTN	110	COT (short)
ZYQSETPO	20	SETPOS

Name	Size (hex)	When present
ZYQSEXP	F0	EXP (short)
ZYQSIMSC	438	SIMSET/SIMULATION
ZYQSIMUC	404	SIMULATION
ZYQSIMUL	30	SIMULATION
ZYQSIMUP	180	SIMULATION
ZYQSLG10	E0	LOG(10) (short)
ZYQSLOG	E0	LN (short)
ZYQSNAP	30	Always
ZYQSSIN	F0	SIN (short)
ZYQSSINH	C0	SINH (short)
ZYQSSQRT	88	SQRT (short)
ZYQSTAN	110	TAN, COT (short)
ZYQSTANH	A0	TANH (short)
ZYQSTCDA	128	Always
ZYQSTCNT	1B10 1)	Always
ZYQSTORE	F18	Always
ZYQSTRIP	60	STRIP
ZYQSUB	58	SUB
ZYQSYN	1B10 1)	Always
ZYQTERM	460	Always
ZYQTERMI	18	TERMINATED
ZYQTERMS	450	SYMBDUMP compile parm > 0
ZYQTIME	18	TIME
ZYQTIMEX	70	Always, Time exit routine
ZYQTRACD	5F0	Tracing is specified
ZYQTRACE	E10	Tracing is specified
ZYQTSPIE	28	Always, Time exit routine
ZYQTVASS	C0	Text value assignment
ZYQTVREL	50	Text value relation
ZYQTXTCM	98	Always
ZYQUNIFO	48	UNIFORM
ZYQUSERX	8	Always, User exit routine
ZYQWAIT	C0	WAIT
ZYQWRITE	1B10 1)	Always

HOW TO DESIGN AN OVERLAY STRUCTURE.

The size of the object program load module can be decreased significantly if it is edited as an overlay program. Routines used to start and finish the object program, as well as diagnostic routines and cardnumber tables, are not needed during execution and they can be put in a segment parallel to the object code.

In order to get an overlay, the parameter OVLY must be specified to the linkage editor and the structure of overlays is defined by control cards:

```
// EXEC  SIMCL,...,  PARM.LKED='MAP,LET,OVLY'
//SIM.SYSIN  DD *
      <program>

/*
//LKED.SYSIN  DD *
   OVERLAY  X  TRANSIENT SEGMENT
   INSERT (ZYQERROR,ZYQCOM,ZYQINIT,ZYQTERM,ZYQCNT)
   INSERT (ZYQMAP,ZYQMAPTR,ZYQDUMP)
   OVERLAY  X  OBJECT CODE SEGMENT
   INSERT ZYQMAIN
/*
```

For each external procedure, the card

```
INSERT  EP#
```

can be put in the transient segment (where EP is the name of the procedure truncated to 7 characters, and

```
INSERT  EP$
```

can be put in the object code segment.

INTERNAL REPRESENTATION OF DATA STRUCTURES

In order to use Assembly or Fortran procedures in a SIMULA program one must know how data and data structures are represented internally. Variables of type INTEGER, SHORT INTEGER, REAL, LONG REAL, and CHARACTER have their obvious internal representations: fullword, halfword, single precision floating point, double precision floating point and EBCDIC character.

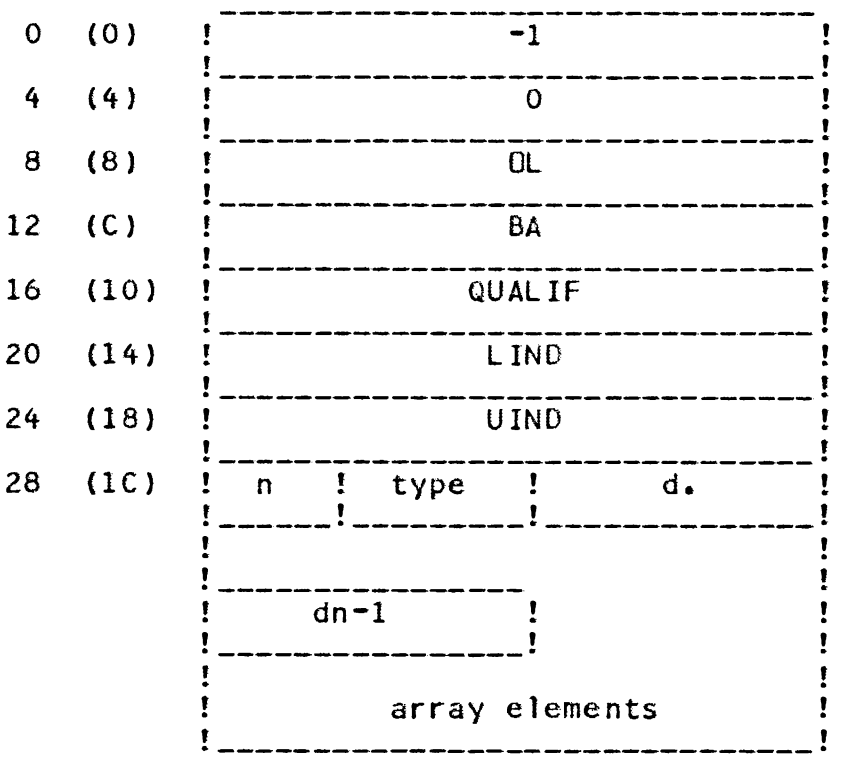
A BOOLEAN is X'00' for FALSE and X'01' for TRUE.

A REF (...) variable is the fullword block instance address, or, if none, X'00FF0000', and an array is the fullword array object address.

A text variable is represented within a block as a 3-word text descriptor. The first word is the address of the text storage block (text object), the second is the address of the first byte of the text -1. The third word is divided into two halfwords: the first is the length of the text and the second is the position indicator.

In a block the quantities are allocated in the same sequence as they are declared, with the spaces and alignments given in table 3.2. The first quantity of a block is allocated at the displacement 8 from the blocks starting address.

Array object format.



-1	in the first word indicates that this is an array object.
OL	is the array object length.
BA	is the address of the element A(0,0,...,0).
QUALIF	is a word identifying the qualification of a REF array, or unused.
n	number of subscripts.
d	dope vector
LIND	lower index
UIND	upper index
type	array type code (App. G).

Dope vector and index checking.

Assume the array declaration

```
A (l(i) : u, ... , l(n) : u(n));
```

Then

```
d(1) = u(1) - l(1) + 1

d(i) = d(i-1)*(u(i) - l(i) + 1), i = 2, ..., n - 1

d(0) = 1 (not present in object)

LIND := 0;

for  i := 1 step 1 until n do LIND := LIND + l(i)*d(i-1);

UIND := 0;

for  i := 1 step 1 until n do LIND := LIND + u(i)*d(i-1);
```

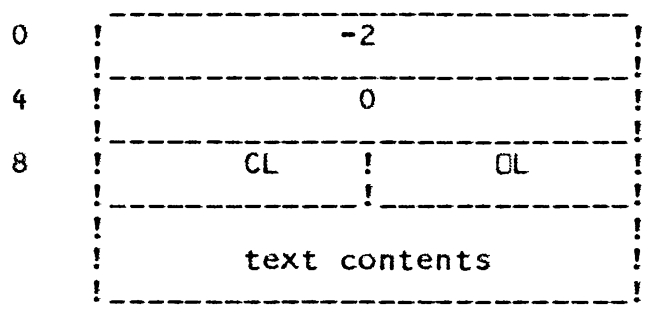
The computation of the adress of A(i , ..., i(n)) is described by the following algorithm:

```
t := 0;

for  k := 1 step 1 until n do t := t + i(k)*d(k-1);

error ("subscriptbounds");
address := t * elementlength + BA;
```

Text object format.



-2 indicates that this is a text object.

CL is the length of the text contents.

OL is the text object length.

OL = (CL + 12 + 7)// 8 * 8

HOW TO WRITE AN EXTERNAL ASSEMBLY OR FORTRAN PROCEDURE

Parameters to assembly and Fortran procedures are passed in the standard OS way: a list of the parameter addresses is formed, and the address of this list is passed in register 1. The uppermost bit of the last parameter address word is set to indicate the end of the list. When the procedure is entered, R14 holds the return address, R15 has the address of the entry point (to the procedure), and R13 has the address of a save area, into which the values of the registers should be saved.

In a call on an assembly procedure, the type of each parameter is indicated by the upper byte of the parameter address word:

INTEGER : 1, REAL : 2, SHORT INTEGER : 3, LONG REAL : 4, TEXT : 6,
REF(...) : 7, CHARACTER : 8, BOOLEAN : 9, LABEL : 10.

If the parameter is an array the type code is OR'ed with X'10' and if it is the last parameter it is OR'ed with X'80'. It is thus possible, within the assembly procedure, to check that the parameters are correct.

The parameter address is the variable address, the array object address (App. F), or the object code label address.

For an EXTERNAL <type> ASSEMBLY PROCEDURE, the resulting value should be loaded into register 0 (INTEGER, SHORT INTEGER, CHARACTER, BOOLEAN), floating point register 0 (REAL, LONG REAL), register 1 (REF(...)), or registers 0-2 (TEXT).

When a branch is made to a label parameter, the label address must be loaded to register 14.

An assembly procedure must always restore registers 3-14 before returning or exiting.

For a Fortran procedure call there is no type and end of parameter list indication, so it is always the responsibility of the programmer writing the call to check that the parameter list is correct. For an array parameter the address of the first element is put in the parameter list.

SPECIFYING USER EXITS

In a program it is possible to specify your own user exit routine. The run-time element ZYQUSERX is defined for this purpose.

The routine is called after sysin and sysout have been opened, but before the first i/o has been performed.

The DCB addresses of sysin and sysout are passed as parameters in the standard OS manner.

In the default case, the routine is only a dummy routine.

SAMPLE JOB LISTING

```
//SIMT JOB (7254,99999942000000,1,6),'NCC',MSGLEVEL=(1,0),    JOB 280
//    CLASS=C,REGION=110K,TIME=3
***         *** INVOKE SIMCG PROCEDURE ***
***
//          EXEC SIMCG,GOPARM='LINECNT=40,TRACE=200'                    1)
++SIMCG     PROC EXLIB=SIMLIB,GOPARM=                                   1)
++SIM       EXEC PGM=SIMULA,PARM='LINECNT=40,XREF',REGION=110K          2)
++STEPLIB   DD   DSN=SIMULA.SIMULA,DISP=SHR                             2)
++SYSPRINT  DD   SYSOUT=A                                               2)
++SYSUT1    DD   UNIT=SYSDA,SPACE=(2000,(20,20))                        2)
++SYSUT2    DD   UNIT=(SYSDA,SEP=SYSUT1),SPACE=(2000,(20,20))           2)
++SYSUT3   DD  UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),SPACE=(2000,(20,10))  2)
++SYSUT4    DD   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),SPACE=(1032,256),    2)
++    DCB=DSORG=DA                                                      2)
++SYSGO   DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=(SYSDA,SEP=SYSUT1),    2)
++    SPACE=(1600,(30,30)),DCB=BLKSIZE=1600                             2)
//SIM.SYSIN DD *                                                       2)
                                                                       3)
++GO        EXEC PGM=LOADER,PARM='MAP,PRINT,LET,PR=ZYQENT/&GOPARM',
IEF6531 SUBSTITUTION JCL - PGM=LOADER,PARM='MAP,PRINT,LET,
++    EP=ZYQENT/LINECNT=40,TRACE=200',
++    COND=(4,LT,SIM)                                                  4)
++SYSLIN    DD   DSN=&&LOADSET,DISP=(OLD,PASS)                         4)
++SYSLIB    DD   DSN=&EXLIB,DISP=SHR                                   4)
IFF6531 SUBSTITUTION JCL - DSN=SIMLIB,DISP=SHR                         4)
++          DD   DSN=SIMLIB,DISP=SHR                                   4)
++SYSLOUT   DD   SYSOUT=A                                              4)
++SYSOUT    DD   SYSOUT=A                                              4)
//GO.SYSIN DD *                                                        4)
                                                                       5)
                                                                       5)
                                                                       5)
IEF3751 JOB /SIMT    / START 71077.1315
IEF3761 JOB /SIMT    / STOP  71077.1316 CPU    0MIN 05.08SEC

 -- HASP-II JOB STATISTICS --
     118 CARDS READ
     279 LINES PRINTED
       0 CARDS PUNCHED
    2.49 MINUTES EXECUTION TIME
```

1) Invoke JCL proc SIMCG
2) Compilation step: Compile SIMULA program, print cross-reference listing
3) SIMULA source program on cards
4) Execution step: link, edit and execute SIMULA program using LOADER. Specify object program parameters and runt-time routine library
5) Input data to SIMULA program cards

```
          COMPILER OPTIONS

          NOLIST                                                        1)
          NODECK                                                        1)
            LOAD                                                        1)
            WARN                                                        1)
            SUBCHK                                                      1)
          NOFXTERN                                                      1)
            XREF                                                        1)
            SOURCE                                                      1)
          NORESWD                                                       1)

            LINECNT =         40                                        1)
            MAXERROR=         10                                        1)
            SIZE    =  (3072,1632,8192,4096,262144,7294)                1)

1)  Compiler options and parameter values in effect

                                                                        *)
01  SIMSET
02  BEGIN CLASS RUNE(T,PART,SECTION);VALUE T;INTEGER PART;              B1
03                    REAL SECTION; TEXT T;
04    BEGIN REF(RUNE)LL,RL;                                             B2
05       REF(HEAD)PTR;
06       PROCEDURE RANK;
07       BEGIN REF(RUNE)ROOT;                                          B3
08             ROOT:-ROOT;
09       SCAN: IF ROOT.T=T THEN NEW PAGE(PART,SECTION).INTO(ROOT.PTR)
10                ELSE BEGIN IF ROOT.T>T THEN                          B4
11                                  BEGIN IF ROOT.LL==NONE THEN      B5
12                                    BEGIN ROOT.LL:-THIS RUNE;      B6
13                                          NEW PAGE(PART,SECTION)
14                                                .INTO(PTR):
15                                    END ELSE                       E6
16                                    BEGIN ROOT:-ROOT.LL;           B7
17                                          GOTO SCAN;
18                                    END                            E7
19                                  END                              E5
20                                  ELSE
21                                  BEGIN IF ROOT.RL==NONE THEN      B8
22                                    BEGIN ROOT.RL:-THIS RUNE;      B9
23                                          NEW PAGE(PART,SECTION)
24                                             .INTO(PTR);
25                                    END ELSE              E9
26                                       BEGIN ROOT:-ROOT.RL;        B10
27                                             GOTO SCAN;
28                                       END                         E10
29                                  END                              E8
30                             END                                   E4
31             END***RANK***;                                        E3
32             PTR:-NEW HEAD;
33        END***RUNE***;                                             E2
34        PROCEDURE TRAVERSE(7);REF(RUNE)7;
35        BEGIN REF(PAGE)X;                                          B11
36              INSPECT Z DO
37              BEGIN TRAVERSE(LL);                                  B12
38                    OUTTEXT(T);  SETPOS(40);
```

```
39                      X:-PTR.FIRST;
40                      WHILE X=/=NONE DO
41                      BEGIN OUTINT(X.PART,1);                          B13
42                            OUTCHAR(':');
43                            OUTFIX(X.SECTION,1,3);
44                            OUTCHAR(',');
45                            X:-X.SUC;
46                      END;                                             E13
47                      OUTIMAGE;
48                      TRAVERSE(RL);
49                END;                                                   E12
50          END***TRAVERSE***;                                           E11
51          LINE CLASS PAGE(PART,SECTION); INTEGER PART; REAL SECTION;
52          BEGIN                                                        B14
53          END***PAGE***;                                               E14
54          TEXT WORD,PARTNO,SECTIONNO;
55          REF(RUNE)ROOT;
56          ROOT:-NEW RUNE("**********",0,0.0);
57          INSPECT SYSIN DO
58          BEGIN WORD:-IMAGE.SUB(1,40);                                 B15
59                PARTNO:-IMAGE.SUB(41,1);
60                SECTIONNO:-IMAE.SUB(43,30);
61          END**SUB-FIELDS;                                             E15
62           INIMAGE;
63           WHILE NOT ENDFILE DO
64           BEGIN NEW RUNE(WORD.STRIP,PARTNO.GETINT,                    B16
65                 SECTIONNO.GETREAL).RANK; INIMAGE;
66           END;                                                        E16
67           TRAVERSE(ROOT);
68  END;                                                                 E1
```

*) begin-end numbering

NO DIAGNOSTICS FOR THIS COMPILATION.

IDENTIFIER	LINE									
ENDFILE	63									
FIRST	39									
GETINT	64									
GETREAL	64									
HEAD	5	32								
IMAGE	58	59	60							
INIMAGE	62	65								
INTO	9	14	24							
LINK	51									
LL	4	11	12	16	37					
OUTCHAR	42	44								
OUTFIX	43									
OUTIMAGE	47									
OUTINT	41									
OUTTEXT	38									
PAGE	9	13	23	35	51					
PART	2*	9	13	23	41	51*				
PARTNO	54	59	64							
ROOT	8	55	56	67						
PTR	5	9	14	24	32	39				
RANK	6	64								
R1	4	21	22	26	48					
ROOT	7	8	9*	10	11	12	16*	21	22	26*
RUNE	2	4	7	12	22	34	55	56	64	
SCAN	9	17	27							
SECTION	2*	9	13	23	43	51*				
SECTIONNO	54	60	64							
SETPOS	38									
SIMSET	1									
STRIP	64									
SUB	58	59	60							
SUC	45									
SYSIN	57									
T	2*	3	9*	10*	38					
TRAVERSE	34	37	48	67						
WORD	54	58	64							
X	35	39	40	41	43	45*				
Z	34*	36								

TOTAL NUMBER OF DIFFERENTLY SPELLED IDENTIFIERS AVAILABLE IN
THIS PROGRAM : 129

BASIC SYMBOL	2:2.0,2:3.0,
BRACES	2:3.0,2:4.0,
BRACKET	2:3.0,
CLASS-IDENTIFIER	2:5.0,
DIGIT	2:3.0,
IN	2:5.0,
IS	2:5.0,
KEY WORD	2:2.0,
LETTER	2:3.0,
OBJECT-RELATION	2:5.0,
SIMPLE-OBJECT-EXPRESSION	2:5.0,
SYNTACTIC UNIT	2:3.0,2:5.0,
SYNTACTIC VARIABLE	2:1.0,2:3.0,
**********	0:0.0,

CATALOGUED PROCEDURES

The following pages contain the catalogued procedures used in 2.4. Small changes may have to be made in order to comply with the standards of a particular installation.

Catalogued procedure SIMC

```
//SIMC     PROC
//SIM      EXEC PGM=SIMULA,REGION=86K
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=SYSDA,SPACE=(2@@@,(2@,2@))
//SYSUT2   DD   UNIT=(SYSDA,SEP=SYSUT1),SPACE=(2@@@,(2@,2@))
//SYSUT3   DD   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),SPACE=(2@@@,(2@,1@))
//SYSUT4   DD   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),SPACE=(1@32,256),
//   DCB=DSORG=DA
//SYSGO    DD   DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=(SYSDA,SEP=SYSUT1),
//   SPACE=(16@@,(3@,3@)),DCB=BLKSIZE=16@@
//   PEND
```

Catalogued procedure SIMCL

```
//SIMCL    PROC PROG=GO,LIB='&GOSET',EXLIB=SIMLIB,LDISP=MOD
//SIM      EXEC PGM=SIMULA,REGION=86K
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=SYSDA,SPACE=(2@@@,(2@,2@))
//SYSUT2   DD   UNIT=(SYSDA,SEP=SYSUT1),SPACE=(2@@@,(2@,2@))
//SYSUT3   DD   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),SPACE=(2@@@,(2@,1@))
//SYSUT4   DD   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),SPACE=(1@32,256),
//   DCB=DSORG=DA
//SYSGO    DD   DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=(SYSDA,SEP=SYSUT1),
//   SPACE=(16@@,(3@,3@)),DCB=BLKSIZE=16@@
//LKED     EXEC PGM=IEWL,PARM='MAP,LIST,LET',COND=(4,LT,SIM),
//   REGION=1@4K
//SYSPRINT DD   SYSOUT=A
//SYSLIB   DD   DSN=&EXLIB,DISP=SHR
//         DD   DSN=SIMLIB,DISP=SHR
//SYSUT1   DD   UNIT=SYSDA,SPACE=(1@@@,1@@)
//SYSLMOD  DD   DSN=&LIB.(&PROG),DISP=(&LDISP,PASS),UNIT=SYSDA,
//   SPACE=(1@24,(4@@,5@,1))
//SYSLIN   DD   DSN=&&LOADSET,DISP=(OLD,PASS)
//         DD   DDNAME=SYSIN
//   PEND
```

Catalogued procedure SIMCLG

```
//SIMCLG   PROC EXLIB=SIMLIB
//SIM      EXEC PGM=SIMULA,REGION=86K
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=SYSDA,SPACE=(2@@@,(2@,2@))
//SYSUT2   DD   UNIT=(SYSDA,SEP=SYSUT1),SPACE=(2@@@,(2@,2@))
//SYSUT3   DD   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),SPACE=(2@@@,(2@,1@))
//SYSUT4   DD   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),SPACE=(1@32,256),
//   DCB=DSORG=DA
//SYSGO    DD   DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=(SYSDA,SEP=SYSUT1),
//   SPACE=(16@@,(3@,3@)),DCB=BLKSIZE=16@@
//LKED     EXEC PGM=IEWL,PARM='MAP,LIST,LET',COND=(4,LT,SIM),
//   REGION=1@4K
//SYSPRINT DD   SYSOUT=A
//SYSLIB   DD   DSN=&EXLIB,DISP=SHR
//         DD   DSN=SIMLIB,DISP=SHR
//SYSUT1   DD   UNIT=SYSDA,SPACE=(1@@@,1@@)
//SYSLMOD  DD   DSN=&&GOSET(GO),DISP=(MOD,PASS),UNIT=SYSDA,
//   SPACE=(1@24,(4@@,5@,1))
//SYSLIN   DD   DSN=&&LOADSET,DISP=(OLD,PASS)
//         DD   DDNAME=SYSIN
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=((4,LT,SIM),(4,LT,LKED))
//SYSOUT   DD   SYSOUT=A
//   PEND
```

Catalogued procedure SIMG

```
//SIMG     PROC PROG=GO,LIB='&&GOSET'
//GO       EXEC PGM=&PROG
//STEPLIB  DD   DSN=&LIB,DISP=(OLD,PASS)
//SYSOUT   DD   SYSOUT=A
//   PEND
```

Catalogued procedure SIMCG

```
//SIMCG    PROC EXLIB=SIMLIB,GOPARM=
//SIM      EXEC PGM=SIMULA,REGION=86K
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=SYSDA,SPACE=(2@@@,(2@,2@))
//SYSUT2   DD   UNIT=(SYSDA,SEP=SYSUT1),SPACE=(2@@@,(2@,2@))
//SYSUT3   DD   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),SPACE=(2@@@,(2@,1@))
//SYSUT4   DD   UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2)),SPACE=(1@32,256),
//   DCB=DSORG=DA
//SYSGO    DD   DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=(SYSDA,SEP=SYSUT1),
//   SPACE=(16@@,(3@,3@)),DCB=BLKSIZE=16@@
//GO       EXEC PGM=LOADER,PARM='MAP,PRINT,LET,EP=ZYQENT/&GOPARM',
//   COND=(4,LT,SIM)
//SYSLIN   DD   DSN=&&LOADSET,DISP=(OLD,PASS)
//SYSLIB   DD   DSN=&EXLIB,DISP=SHR
//         DD   DSN=SIMLIB,DISP=SHR
//SYSLOUT  DD   SYSOUT=A
//SYSOUT   DD   SYSOUT=A
//   PEND
```

Catalogued procedure SIM

```
//SIM        PROC  P=,CP=,LP=,RP=,GP=
//GO         EXEC  PGM=SIMCNT,COND=EVEN,
//                 PARM='&P/&CP/&LP/TIME=1000,&RP/&GP'
//STEPLIB    DD    DSN=simlib,DISP=SHR
//SYSPRINT   DD    SYSOUT=A
//SYSUT1     DD    UNIT=SYSDA,SPACE=(2048,(20,20))
//SYSUT2     DD    UNIT=SYSDA,SPACE=(2048,(20,20))
//SYSUT3     DD    UNIT=SYSDA,SPACE=(1632,(30,15))
//SYSUT4     DD    UNIT=SYSDA,SPACE=(1032,256),DCB=DSORG=DA
//SYSGO      DD    UNIT=SYSDA,SPACE=(1600,(100,50)),
//                 DCB=(LRECL=80,RECFM=FB,BLKSIZE=1600)
```

REPORT PROCEDURE

If a fault is found, or is believed to be found, in the SIMULA system, it should be reported to the Norwegian Computing Center on a standard error report from provided for this purpose. Care should be taken to describe the error and its context correctly, and it is preferable to send the program listing with the error report form.

Reported errors will always be dealt with in their order of arrival at the Norwegian Computing Center.

The layout of the error report form is shown on the next page.

	Customers name:	Release no.:
	Location:	Date compiled:
	Project leader:	Reference no.:

This report should be sent to:

Norwegian Computing Center
Forskningsveien 1b
Postboks 335, Blindern
Oslo 3
Norway

Description of problem:

Enclosure: Compilation listing Console listing Source
Object program Test material Dump

For use by NCC

Analysis of problem:	Corrective action taken:

Registration no: