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 Additional 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:

17 MAR 1978 17:55:45.18

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

```
Simulation begin real X; integer U, COUNT;                         B1
     ref(Head) WAITQUEUE, OUTQUEUE;

     Process class ENTITY; virtual: procedure SNAPSHOT;
     begin real TIMEUSED;                                           B2
           procedure SNAPSHOT(TITLE); name TITLE; text TITLE;
           begin external assembly procedure SYMBDUMP;
                 TITLE:="DEBUGGING VERSION";
                 SYMBDUMP(4,"SITUATION BEFORE START");
           end OF SNAPSHOT;

           TIMEUSED:=Time;
           into(WAITQUEUE);
           reactivate this ENTITY delay Normal(38,3,U);
           TIMEUSED:=Time-TIMEUSED;
           COUNT:=COUNT-1; CHARS(COUNT):='*';
           Wait(OUTQUEUE);
     end OF ENTITY;                                                 E2

     boolean STARTED;
     character array CHARS(8:127);



     U:=Inint;
     WAITQUEUE:-new Head; OUTQUEUE:-new Head;
     for COUNT:=1 step 1 until 5 do activate new ENTITY;

     Inspect new Printfile(Intext(8)) do
     begin text SUBTITLE;                                           B4

           Open(Blanks(80));
           Outtext("SIMULATION PROTOCOL:");
           SUBTITLE:-Image.Sub(Pos+1,Length-Pos);
           WAITQUEUE.Last qua ENTITY.SNAPSHOT(SUBTITLE);
           Close;
     end OF INSPECTION;                                             E4

     STARTED:=true; Passivate; comment WILL GIVE A RT-ERROR:

     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 0009 (SITUATION BEFORE START) ------------

OPERATING CHAIN :

SYSOUT  INSPECTED AT  SYSTEM LEVEL: IMAGE==#099058/POS=1 OF 132/="
SYSIN   INSPECTED AT  SYSTEM LEVEL: IMAGE==#0990E8/POS=10 OF
                      80/="3SIMPRINT

PROCEDURE SNAPSHOT ON LINE 0006 (LOCAL TO ENTITY(5) ON LINE 0004),
CALLED FROM LINE 0035:
       TITLE          IS SUBTITLE OF BLOCK ON LINE 0030

BLOCK ON LINE 0030:
       SUBTITLE       ==#099398.SUB(22,59)/POS=1/="DEBUGGING VERSION

PRINTFILE(2) ON LINE 0000 IN *PREDEFINED*, TERMINATED,
VISIBLE THROUGH INSPECTION:
       IMAGE          ==#099398/POS=21 OF 80/="SIMULATION PROTOCOL:
                        DEBUGGING VERSION
       DDNAME         :SIMPRINT
       LINE           =1
       LINESPERPAGE   =46
       SPACING        =1

SIMULATION BLOCK ON LINE 0001:
       MAIN           ==MAINPROGRAM(1)
       CURRENT        ==MAINPROGRAM(1)
       U              =-1317061513
       COUNT          =6
       WAITQUEUE      ==HEAD(1)
       OUTQUEUE       ==HEAD(2)
       #099188 ARRAY CHARS(0:127):

SQS :  SCHEDULING TIMES       PROCESSES (SIMULATION BLOCK AT LEVEL 1,
                              TIME=0.0)

       0.0                    MAINPROGRAM(1) ON LINE 0000 IN
                              *PREDEFINED*,
                              DETACHED AT LINE 0000
       2.74757251739502E+01   ENTITY(5) ON LINE 0004,
                              DETACHED AT LINE 0014
       3.069438171386 72E+01  ENTITY(2) ON LINE 0004,
                              DETACHED AT LINE 0014
       3.160680389404 30E+01  ENTITY(4) ON LINE 0004,
                              DETACHED AT LINE 0014
       3.293387699127 20E+01  ENTITY(1) ON LINE 0004,
                              DETACHED AT LINE 0014
       3.713046264648 44E+01  ENTITY(3) ON LINE 0004,
                              DETACHED AT LINE 0014

--- END OF SYMBDUMP CALL AT LINE 0009 --------------------------------
```

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.