S - P O R T:

The Environment Interface
(Version 3.0)

By Geoffrey E. Millard, Program Library Unit, Edinburgh
Øystein Myhre, Simula a.s.
Gunnar Syrrist, Simula a.s.

This document is under supervision of the Standards Group for the Portable SIMULA System (SG/PSS). The statutes of the Standards Group are found in appendix D.

The document contains all decisions taken by the SG/PSS, whose last meeting was held in Edinburgh 1st and 2rd March 1983.

Simula a.s.,

21st June 1985

REMARK: This document is reassembled from the HTML version donated by Simula a.s.

Oslo 15. August 2021
Øystein Myhre Andersen

1 PREFACE

In 1979 the Norwegian Computing Center (Oslo) and the Program Library Unit (Edinburgh) initiated the "S-PORT" project: implementation of a portable SIMULA system. The system consists of three parts: a portable front-end compiler, a portable run time support system, and a machine dependent code generator; the latter includes the interfaces to the compile and run time environment. The present report gives the specifications of what the Portable SIMULA System will demand from the system environment.

Both the front-end compiler and the run-time system will be distributed in S-code. In order to implement SIMULA it will be necessary to program the code generator (the S-compiler) for the machine in question. This compiler will take care of two tasks:

- it will produce code for the target machine corresponding to the S-program, and

- it will insert the necessary links to the operating system.

The document is a revision of earlier documents, dated September 26th 1980, January 23rd 1981 and November 23rd 1981. These revisions are done after meetings in Edinburgh 3rd and 4th March 1981, in Oslo 21st and 22nd May 1981, in Geneva 7th September 1981 and a meeting held in Paris 2nd and 3rd March 1982.

This definition is based on a reworking of the previous ones due to the experience gained in the process of implementing the first S-code compilers. We believe that the document should now define a sufficient basis to supply what the system demands from the the environment in a convenient way. The implementations which have given us this experience are the ones for the HB level 66, VAX11/780 systems, Nord-500 and the UNIVAC 1100.

Any comments for the improvement of the definition will be welcome. Proposals to revisions of the document will be subject to discussions in the Standards Group for the Portable SIMULA System.

2 INTRODUCTION

The demands from the system to the environment are divided into two classes, those that must be provided for the system to function, and those which are candidates for efficiency improvements.

In the syntax of the routine profiles, routines are specified to be "peculiar", either as "system" or "known" routines. The specification implies:

System: These routines provide the necessarry links to the operating system environment. These routines must be provided for the system to function.

Known: These routines could be provided for efficiency reasons, although their definitions will be given in S-code. In most cases this would be done for services that can usually be done more efficiently at the implementor's level.

The classification above is more thoroughly described in the document "Definition of S-code". The present document cannot be read without extensive knowledge of this document. Other related literature is listed in the mentioned document.

The routines listed as system routines in the portable system environment are to provide those services which are machine or operating system dependent. These include:

- provision of workspace,
- the processing of control data,
- all input/output,
- the initial interpretation of exception conditions, and
- the processing of diagnostic records.

We describe here how the S-code assumes that the necessary information is organised, according to S-code standards.

3 THE ENVIRONMENT INTERFACE MODULE

In this document we give the profiles for the routines in question. The profiles will give an implicit definition of the calling sequence of a routine, i.e. the sequence of instructions encountered in connection with a call on the routine. The contents of the compile time stack when the S-compiler sees a call on a routine is implicitly defined through the routine profile. The contents of the stack when the compiler is finished processing the call is also defined by the profile.

For each routine we specify the profile and which actions the environment must perform on the basis of a call on the given routine. When a profile is specified "peculiar", we also give the tag of the only routine body that can be connected to the profile.

The definition of the profiles for the environment routines will all be specified within one module, the "interface module". A sketch of this module follows:

```
global module "enviro" "update-no"
global status INT range 0 36 system "status"
global itemsize INT range 0 72 system "itsize"
global curins OADDR system "curins"
global bioref OADDR system "bioref"
global tmpqnt QUANT system "tmpqnt"
const maxrank INT c-int "0" system "maxrnk"
const maxint INT c-int "0" system "maxint"
const minint INT c-int "0" system "minint"
const maxreal REAL c-real "0" system "maxrea"
const minreal REAL c-real "0" system "minrea"
const maxlreal LREAL c-lreal "0" system "maxlrl"
const minlreal LREAL c-lreal "0" system "minlrl"
record STRING
attr chradr GADDR
attr nchr INT
endrecord
const maxlen SIZE system "maxlen"
profile initialisep .....
profile terminatep .....
......
global encdrv OADDR
global curdrv OADDR
......
tag Status 0
tag Itemsize 1
tag STRING 2
tag maxlen 3
.....
body
.....
endmodule
```

The interface module specifies the assumptions that are made about the environment. The module also serves as a global area for variables.

All global variables and system variables and constants are declared within this module.

The global system variables status and itemsize have been given a name by which they can be referenced externally, in the same way as the system profiles will be given a name.

The rest of the global system variables are variables that have been given external names for optimization purposes. An implementation may gain considerable if these variables are kept in register storage. "curins" is the OADDR of the current instance, "bioref" is the OADDR of the BasicIO instance, and "tmpqnt" is a variable used widely to communicate evaluation results.

Note that the type QUANT is also defined in the interface module.

The constants have been included to give access to their system dependent values. The default values given in their definitions are, of course, not meaningful, they are included only for syntactic reasons and must always be replaced by an implementation.

All type declarations that are used in communication with the environment profiles are defined in this module.

All profiles to system routines are given in this module.

All tags that are visible from outside the module are given an external reference number through the tag instruction. The complete listing of the module is given in appendix A.

None of the routines in the environment will be recursive. None of the routines will be able to modify their return point, since none of them have exit specified in their profile.

The necessary initialisation of global variables may only be done once.

3.1 Initialisation of the environment

```
profile initialisep:newtag
system initialise:newtag "initia"
import ii_erhandl RADDR
endprofile
```

A call will be made to this routine in the initialisation code of the run-time system. This will give the S-compiler writer the possibility to ensure that the run time environment, the RTE, is initialised in implementations which allow direct entry to the user program from the operating system environment. Any system dependent actions that should be done, would then naturally be defined in this routine.

The import parameter to the routine will be the RADDR of the routine that is to be called if the environment gets control after an interrupt. This routine will be the exception handler. This routine and its profile are described in detail in section 10.2. We want a controlled way of returning to the user, giving a problem relevant description of what has gone wrong.

3.2 Terminate program

```
profile terminatep:newtag
system terminate:newtag "termin"
import t_code INT range 0 3
import t_message STRING
endprofile
```

Code: Completion code:

0 - normal termination
1 - user requested termination
2 - termination after error in user program
3 - termination after Simula system error

Message: Possible message to the user, see below.

Execution of a Simula program will be concluded by a call on terminate. The routine must ensure a consistent state of the environment interface, i.e.

- all files still open must be closed by terminate, and

- resources acquired for the program should be released (this action may be left to the environment if possible).

The completion code returned from the run time system indicates whether the program is terminated under controlled conditions (codes 0-2) or whether an irrecoverable error was discovered by the internal run time system consistency checks (code 3). In the former case all files should have been closed explicitly before terminate is activated, thus open files indicates a Simula system error which should be reported through a suitable diagnostic given by this routine. On the other hand it is to be expected that files are left open in case the completion code is 3 since the internal data structures probably are invalidated, so in this case no such diagnostic should be given.

A message from the run time system may be pending, either because all files have been closed by the Simula program thus there's nowhere to give a termination message, or because an error has invalidated the internal data structures of the runtime system. The message parameter provides for such situations: if non-empty it should be given on the standard output connected to the program (irrespective of the value of the completion code).

Finally control is returned to the environment. The manner in which this is done is completely implementation dependent.

This routine cannot change the value of status.

If status <> 0 on a call to the routine 'termin', the routine should report this as a system error.

Note: This routine should never attempt to return to where it was called.

4 INFORMATION FROM THE USER OR THE ENVIRONMENT

The central run-time system and the diagnostic package require information about the options selected by the user, the current mode of operation and the environmental conditions applicable to the particular implementation.

The front-end requires information about the compile time options selected by the user, and some information which is only known in each implemented system, such as default file names.

Seen from the environment the front-end is just another program, heavily using the general environment routines. Like other programs it needs input for its work. This input can be of two types:

- options or parameters for the compilation, and
- the program to be compiled.

Information typically seen as parameters to the front-end compiler would be where to find the source program, where should the generated S-code go, should a source listing be produced etc. In addition, capacity parameters used to dimension e.g. arrays internal to the front-end are also desirable to retreive from the environment so that a compilation to some extent may be adjusted to the needs of the source program.

The compilation may also be controlled by means of compiler directives. The general format of such directives is given in the SIMULA Standard, a list of directives particular to the S-port implementations is included in the release description for the system. This list may be extended as needed by the implementors (but such extensions will of course not affect the front end compiler).

Similarly, both the run time system and the symbolic debugger SIMOB will require information from the environment.

In some cases it will be convenient to use some of the parameters to the front-end for only parts of the compilation. This will be possible through the use of source input control statements. They must be included in the source text file, to specify for which part of the compilation they shall be in effect. A description of the control statements can also be found in the installation guide for the S-PORT system.

The run-time system will, similar to the compiler, make several requests for functions to be provided in the environment. The run-time system also has a number of parameters for an execution of a program. When the environment has specified that it wants to control transmission of parameters, they will be also asked for through the routines in this section.

The environment is responsible for processing the external representation of user options and providing the information in response to a call on any of the routines in this section. The routines have been defined in this way to give the system maximum flexibility in retrieving the information from the environment. The implementor is free to have the options presented from the user in any way convenient.

We have found it convenient to define the following routines to handle this. Some options need only return an integer, but some need to return a text string. For each routine we have specified the possible input parameters and what are the relevant return values.

The list of index interpretations under each routine may not be complete, seen from the users point of view. We are here only concerned with those options that must be known to the front-end or the run-time system.

If the front-end requests information that the user has not provided, the environment should respond with what has been chosen as default in this particular implementation.

Each of the routines described in this chapter has as its first parameter an "index". This document describes the indices valid at the time of printing. As the S-port system is under continuous development, new indices may be added as needed. A list of all indices valid for a particular S-port release is enclosed in the release description.

4.1 Integer valued information from the environment

The routine get_intinfo is defined to get all information from the environment that can be coded as an integer.

```
profile get_intinfop:newtag
system get_intinfo:newtag "gintin"
import gi_index INT range 0 127
```

export gi_result INT
endprofile

The parameter index is an integer specifying what information is requested. The result will be an integer whose interpretation gives the specified information. The result is given for each value of index as follows:

Index: Interpretation:

1 NOT USED

2 Is a source listing wanted?

Result: 0 - No.

>0 - Yes. The listing will be a copy of the source text, where each line is prefixed by its number.

3 *) Should begin/end counters be included in the source listing?

Result=0 means no, otherwise yes.

4 What is the maximum number of error messages to be given?

Result will be the number of messages.

5 Should warning messages be suppressed?

Result=0 means no, otherwise yes.

6 Should a cross-reference listing be produced?

Result=0 means no, otherwise yes.

7 *) What is the image length for the listing file?

Result will be the number of characters in the image.

*) These calls are only made after get_textinfo(2) (asking for name of the listing file) is called, and only if listing is wanted.

8 What is the line length of the source file?

Result will be the number of characters in the image.

9 Should test for none be ommitted at remote access?

Result=0 means no, otherwise yes.

10 Should checking of indices be ommitted at array access?

Result: 0 - Complete checking of array indices

1 - Partial checking of indices

2 - No checking will be done.

11 What is the level of information wanted from the symbolic dump routine?

Result: 0 - The symbolic dump module is not to be included

1 - The symbolic dump routine is included

2 - Individual instances of an object shall carry a count for identification

3 - Objects shall carry information of all identifier names.

12 Should inclusion be made for production of a dynamic profile of the program execution?

Result=0 means no inclusion, otherwise assignment counts are to be included

13 Should inclusion be made for full tracing of control flow at run-time?

Result=0 means no inclusion, otherwise full inclusion.

14 Should inclusion be made for interactive debugging of the program?

Result=0 means no inclusion, otherwise full inclusion.

15 What is the level of debug information wanted in case of a run time error?

Result: 0 - no debugging information is wanted

1 - a diagnostic message and the source program line number where the error occurred are wanted

2 - in addition to the above the static link and the dynamic link at the point of error are wanted

3 - in addition to the above the sequencing sets of all SIMULATION blocks and the local sequence controls of all scheduled processes are wanted

4 - in addition to the above the local sequence controls of all non-terminated objects are wanted

5 - in addition to the above all referable datastructures are wanted.

16 Is tracing of control flow wanted?

Result: 0 - no tracing is wanted

>0 - the number of messages wanted is given

<0 - tracing messages are printed to a buffer of size -(result), and only listed in case of a run-time error

17 Is tracing of data flow wanted?

Result: 0 - no tracing is wanted

>0 - the number of messages wanted is given

<0 - tracing messages are printed to a buffer of size -(result), and only listed in case of a run-time error

Comment: If both control and data flow tracing are specified, they will go to the same destination.

18 What is the maximum amount of time (cpu-time) to be used for the execution?

Result will give the time, specified in 1/100 sec.

19 Should the symbolic debugger SIMOB be entered prior to the execution of the program, and at program termination? An answer greater than zero will give this effect.

20 What is the significant linelength of the source file image?

21 What is the maximum index to setobj, getobj, and access instructions allowed in this implementation. A response of 0 will give the limit 255.

22 Mode of separate compilation ?

0: normal separate compilation

1: recompilation.

23 Amount of pass information from FEC on listing or message file ?

0: No information.

1: Minimum.

2: Medium.

3: Maximum.

24 How many work areas may be requested (see chapter 5)?

30 What is the level of data information which must be produced at compile-time, in order to observe the execution at run-time?

Result: 0 - minimal information for error reporting.

1 - information at the module and block level, but no information about the attributes.

2 - complete information generated, including information about all attributes.

SIMOB can always be used for observation, but the available information will depend on this response.

31 Should inclusion be made at compile-time for statement execution counts?

Result: 0 - no, statement execution counts not wanted

1 - yes, statement execution counts wanted

32 Should inclusion be made at compile-time for processor usage measurements?

Result: 0 - no, measuring of processor usage not wanted

1 - yes, measuring of processor usage wanted

33 Is interaction with the user possible in the current execution?

Result: 0 - no, this is not an interactive execution

1 - yes, this is an interactive execution

Note that SIMOB uses this.

34 Should inclusion for the possibility of statement start exceptions be made at compile-time by the S-Code Compiler?

Result: 0 - no, no start of statement exceptions will ever occur

1 - yes, start of statement exceptions may occur

35 Should inclusion be made at compile-time for storage usage measurements?

Result: 0 - no, measuring of storage usage not wanted

1 - yes, measuring of storage usage wanted

36 What is the maximum number of identifiers allowed in this program compilation. Result will be the number allowed.

37 What is the maximum number of constants allowed for this compilation. Result will define the number.

38 What is the maximum number of textual blocks allowed for this compilation. Result will define the number.

39 What is the maximum number of block levels allowed for this compilation. Result will define the number.

40 What is the maximum source depth allowed for this compilation. Result will define the maximum.

41 What is the maximum dynamic depth in this program compilation. Result will define the maximum.

42 What is the maximum number of parameters in a procedure call for this compilation. Result will define the maximum.

43-127 As defined in the current Release Description.

If the environment returns a value of zero for any of these indices, some default value will be chosen by the system.

4.2 Text valued information from the environment

The routine get_textinfo is defined to get all information from the environment to the front-end that must be given as a text.

```
profile get_textinfop:newtag
system get_textinfo:newtag "gtexin"
import gt_index INT range 0 127
import gt_result STRING
export gt_filled INT
endprofile
```

The parameter index is an integer that specifies what information we request. The result will be a string which is filled into result. The export parameter filled gives the number of characters in the string.

Index: Interpretation:

1 What is the name of the source input file?

Result will give the name.

2 What is the name of the listing file?

Result will give the file name. If no such output is to be produced, then the string will be empty.

3 What is the name of the file for separate output of diagnostics messages?

Result will give the file name. If no such output is to be produced, then the string will be empty.

4 What is the name of the file for storing the normal (byte packed) S-code?

Result will give the file name. If no such output is to be produced, then the string will be empty.

5 What is the name of the file for storing decimally coded S-code?

Result will give the file name. If no such output is to be produced, then the string will be empty.

6 What is the name of the file for storing textually coded S-code?

Result will give the file name. If no such output is to be produced, then the string will be empty.

7 What is the name of the scratch file to be used for storing the intermediate code between the passes in the front-end?

Result will give the name of the file.

8 What is the name of the scratch file to be used for storing the declaration structures between the passes in the front-end?

Result will give the name of the file.

9 Reserved for further scratch file requests.

Result will give the name of the file.

10 Reserved for further scratch file requests.

Result will give the name of the file.

11 What is the name of the attribute file for the current compilation? This will only be necessary for a separate compilation. This file will be used as the front-end's attribute file. (Cf. section 4.6)

Result will give the name of the file.

12 What is the name of the attribute file for an external declaration?

Before this request is issued, the environment will have received the identifier and the external identifier for the external declaration through the routine give_textinfo described below. (Cf. section 4.6)

Result will give the name of the file.

13 What is the environment part of the program head? (See below)

Result will give the text string.

14 What is the module identifier to be used for the current compilation? (Cf. section 4.6)

This call will only be made for a separate compilation.

Result will give the text string.

15 What is the check code to be used for the current compilation? (Cf. section 4.6)

This call will only be made for a separate compilation.

Result will give the text string.

If the empty string is delivered then the date_and_time string identifying this compilation will be used.

16 What is the system debugging option string?

The default answer here should be the empty string.

17 Not used.

18 Not used.

19 What is the file name of the attribute file for the predefined classes and procedures (the class PREDEF)?

20 What is the file name for the attribute file for the class SIMSET?

21 What is the file name for the attribute file for the class SIMULATION?

22 What is the file name of a file containing seldom used information for the front end compiler, such as extended error messages.

23 What is the file name of a file containing seldom used information for the run time system, such as extended error messages.

24 What is the identification string of the current execution? The answer should be as defined for procedure "simulaid" in the SIMULA Standard, section 9.6. If the answer is the empty string, RTS will fill out the first field (SIMULA system name) with an identification of the current S-port release, and leave the remainding fields empty (i.e. the release info will be terminated by 21 exclamation marks).

A short comment is necessary on the program head string.

According to the definition an S-program should start with the
keyword program followed by a string. This string is used to identify the compilation.
The string will be given according to the following syntax:
"< a >'< b >'< c >"

The three parts of the string are:

< a > This is the date and time of compilation given through the result from a call on the routine date_and_time.

< b > This is an identifiaction of the front-end compiler chosen by itself to identify the version of the compiler.

< c > This is an identification of the S-code compiler supplied to the front-end compiler when get_textinfo is called with index 13 (see page 15).

4.3 Information to the environment

The routine give intinfo is defined to submit information from the front-end compiler or the run-time system to the environment. This information is gathered from the source input under compilation.

```
profile give_intinfop:newtag
system give_intinfo:newtag "gviinf"
import gii_index INT range 0 127
import gii_info INT
endprofile
```

The parameter index is an integer that specifies what information follows. Info will be an integer carrying the following interpretation:

Index: Interpretation:

1 A call with this index is done immediately before the termination of each pass, and the value of info signals the situation after this pass, by the following coding:

0 - No user errors found. Go on with next pass.

1 - User errors are found, but go on with next pass.

2 - Reserved for fututre use, continuation is possible.

3 - No user errors found, but because of options etc. (e.g. that S-code should not be produced), the next pass should not be started.

4 - User errors found, therefore do not start next pass.

5 - Too many or too difficult user errors encountered. Therefore the current pass is terminated, and the next pass should not be started.

6 - An internal error in the compiler has occurred. Therefore the current pass is terminated, and the next pass should not be started.

2 Info is the highest tag used in the S-code for this program.

3 Info is the number of source lines in the Simula program being compiled.

4 Info is the number of errors for this compilation.

5 Info is the number of warnings for this compilation.

6 Garbage collection information. Info=0 signals the start of a garbage collection, Info=1 signals termination of g.c. (see 5.2).

The routine give_textinfo is defined to submit information from the front-end compiler or the run-time system to the environment. This information is gathered from the source input under compilation.

```
profile give_textinfop:newtag
system give_textinfo:newtag "givinf"
import give_index INT range 0 3
import give_info STRING
endprofile
```

The parameter index is an integer that specifies what information follows. Info will be the string reference with the specific information as follows:

Index: Interpretation:

1 The string info is the identifier of a class or procedure being separately compiled.

2 The string info is the identifier given in an external declaration that is being processed.

3 The string info is the external identification given in an external declaration that is being processed.

4.4 Information passed for an external compilation/declaration

When the front end compiler starts reading the source program, it will syntactically find out if we have a separate compilation of a class or procedure. The following will then be performed:

- The identifier of the class or procedure will be sent to the environment through give_textinfo (index=1).

- The front-end will ask the environment what module identifier is to be used through get_textinfo (index=14).

- The front-end will ask the environment which check-code to use through get_textinfo (index=15).

- The front-end will ask the environment the file name of the attribute file for this separate compilation through get_textinfo (index=11).

When the front-end compiler meets an external declaration, another sequence of operations will be performed. An external class declaration is given in one of the following ways:

external class <class-id>;

external class <class-id> = <external-id>;

An external procedure declaration will be given in a similar way. The following will now be done:

- The front-end will give the environment the identifier (<class-id>) given in the external declaration through give_textinfo (index=2).

- The front-end will give the environment the external identification (<external-id>) given in the external declaration, or the empty string if no such identifier is given. The information is given through give_textinfo (index=3).

- The front-end will ask the environment the name of the attribute file for this external declaration through get_textinfo (index=12).

5 Dynamic Storage Administration

One of the most important components of the SIMULA run time system is the storage administrator. It will dynamically allocate storage to objects generated during program execution, and it will perform storage regeneration (also known as garbage collection) when needed. All dynamic allocations are done within a work space supplied by the environment, and this chapter describes those interfaces between the garbage collector and the environment that

- gives the garbage collector access to the work space
- permits a dynamic extension or contraction of the work space.

5.1 <u>Work Space Administration</u>

The work space will be divided into one or more work areas. A work area is a contiguous memory area. It is identified by a number, and these numbers order the work areas into a sequence. One work area is called the current work area, it is always the last in the sequence of work areas provided by the environment.

Initially, the storage administrator will ask the environment to provide the allowed number of work areas for this execution, through a call to the
routine get_intinfo(24).

Objects are always allocated within one work area, so the size of a work area determines the maximum size of an object that can be allocated within it. Object allocation is done after the following algorithm:

1. If there is enough room in some work area, the object is allocated (and the available space within the work area in question is decreased accordingly).

2. If space cannot be found for the object, the environment is asked whether a garbage collection should be performed, through a call to the known routine ExtendOrGC. If the answer given is false (meaning "perform garbage collection"), proceed with step 5.

3. The environment is asked to extend the current work area, through def work area(WantedSize,CurArea). If the call is successful (status=0) the object is allocated (and the algorithm stops).

4. The environment is asked to provide a new work area, through def work area(WantedSize,CurArea+1). If the call is successful (status=0) the object is allocated (and the algorithm stops). If unsuccessful (status<>0) and a garbage collection already has been performed, execution is terminated.

5. A garbage collection is performed. If that leaves enough room the object is allocated (and the algorithm stops).

6. If extension of the current work area wasn't attempted earlier (i.e. the algortihm passed directly from step 2 to step 5), proceed from step 3. Otherwise execution is terminated.

5.2 Garbage Collection

When a garbage collection is initiated, the environment is informed through give_intinfo(6,0). After the garbage collection is terminated, the environment is informed through give_intinfo(6,1). This provides the environment with the opportunity to inform the user (e.g. by means of a displayed "busy bee") that the system is working at the moment although it may not respond.

A garbage collection will traverse the work space (i.e. all work areas provided by the environment) and delete all objects that are no longer connected to the program execution. The surviving objects will then be moved so that each work area is divided into two parts: one part containing active objects, and one part (the "gap") which is available for object allocation. During this compaction objects may be moved from one work area to another, in an attempt to leave the largest gap at the end of the current work area.

Following some ad-hoc rules the garbage collector will contract or even delete the current work area, through an appropriate call on def work area. If an area is deleted, the preceding work area becomes the current work area.

5.3 Define Work Area

```
profile def_work_areap:newtag
system def_work_area:newtag "dwarea"
import dwa_lng SIZE
import dwa_warea INT range 0 255
export dwa_top OADDR
endprofile
```

Lng: The wanted size of the work area. If nosize the work area is deleted. Lng will always be either nosize, or the initial size of the work area in question plus a multiplum of the extension to that area (see routine get_sizeinfo below).

Warea: Identification of the work area in question.

Top: If Lng<>nosize, the result is the object address of the first allocation point within the reserved area. If Lng=nosize, the result is nowhere.

The routine def work area must return a reference to a contiguous area. If the identified work area has already been defined (and not deleted) the environment is asked to either extend or contract the current area.

This routine may change the value of the global variable status to one of the values given in app. C. Return of status 19 will indicate, on a request of redefinition, that the environment is not able to support this.

5.4 Size valued information from the environment

The routine get_sizeinfo is defined to get information from the environment that must be presented as a SIZE.

```
profile get_sizeinfop:newtag
system get_sizeinfo:newtag "sizein"
import gs_index INT range 0 2
import gs_warea INT range 0 255
export gs_result SIZE
endprofile
```

Index: Specifies the information requested (se below).

Warea: Identifies the work area in question.

Result: The wanted SIZE, according to the value of Index:

Index	Result
1	The minimum size of this work area.
2	The extension/contraction step size.
3	The minimum gap left in this work area after a garbage collection, if the area is the current work area.

If the work space is found to be full even after garbage collection, then the current work area must be extended. The garbage collector may also decide that the available work space is too large for the current needs of the execution, consequently it will contract the current work area.

In both cases, the results according to index 1 and 2 for the current work area are used to determine the new size of the current work area, as explained earlier in this chapter.

It is possible to get into a situation where time is spent almost exclusively in the garbage collector, because the gap left after garbage collection is too small (allthough sufficient for the immediate need). In an ad-hoc attempt to avoid this, the environment is asked to provide a minimum gap size for each work area. If the current work area does not contain a gap of at least that size after a garbage collection, more work space will be requisitioned from the environment, either by extending the current area, or in the form of a new work area. In the event that the current area is the highest possible area (as identified by the result of get_intinfo(24)) and cannot be extended, the execution is terminated.

Note that a work area may have a gap smaller than that area's minimum gap, as a consequence of storage compaction or object allocation (see step 1 in the allocation algorithm).

5.5 Known routine ExtendOrGC.

```
profile extend_or_GCp:newtag
known extend_or_GC:newtag "xorgc"
import ext_obj SIZE
export ext_result BOOL
endprofile
```

Obj: The size of the object under attempted allocation

Result: true if storage extension shall be attempted first, false if garbage collection shall be attempted immediately.

This routine allows the environment to guide the sequence of the allocation algorithm, see section 5.1.

6 MOVE INFORMATION

```
profile movep:newtag
system move:newtag "movein"
import mo_from OADDR
import mo_to OADDR
import mo_length SIZE
endprofile
```

From: The "lowest" object address of the area to be moved.

To: The "lowest" object address of the destination area.

Length: The size of the area to be moved.

The garbage collector (in the run time system) needs to move areas during the compactifying phase. In order to be able to take advantage of possible block transfer instructions, and because such moves otherwise might be rather time consuming, this routine is defined.

Two types of moves are employed. The first type will always move an area towards the high end of work area, and the involved areas will never overlap. The second type will always move towards the first allocation point of the work area, the involved areas may in this case overlap.

In both the case an implementation will be safe (with regards to overlap) if the area is moved sequentially starting with the object unit identified by "from" (this unit must be moved to the unit identified by "to").

7 FILE HANDLING

In order to avoid confusing the Simula file concept with the files of the file system we will call the latter data sets; a data set may be an actual file or it may be an I/O device.

The current chapter defines all routines necessary to implement the input/output interface of Simula programs.

Some general problems still exist. Some of these are not covered by the current text, and they will have to be solved at a later point:

- What if a file is "busy"? The file exists, but is not available.
- What if there is not enough space for creating a data set as specified?
- What about file hierarchies?

7.1 Identification of data sets

Whenever a Simula file object is generated (through new) the parameter to the file class must be a text containing a data set specification, intended to identify a data set on the target system. The link between such a data set specification and a data set is to be established when the data set is opened (through a call to routine open, see 7.4). How this link is established is completely system dependent: the data set specification may simply be the data set name, or it may identify a data set descriptor which contains the true data set name etc.

When the file is opened, the corresponding data set is opened, and for efficiency reasons it is no longer referred to by the data set speci-fication; actually the environment need not remember this specification at any time. The data set is referred to by means of a (data set) key. The key is an integer in the range 1..255 returned from the open routine and is used in all accesses to the open data set (including close).

Initially all keys are invalid, i.e. no data sets are open. A given key is valid only from the time it is exported from routine open until it is imported to routine close. Thus the keys must satisfy the following invariants:

- a key is either invalid, or it identifies precisely one open data set, and
- an open data set is associated with precisely one key.

7.2 File types

The files may be of the following types:

1 - infile (record oriented sequential read access)
2 - outfile (record oriented sequential write access)
3 - printfile (printer formatted outfile)
4 - directfile (record oriented random read/write access)
5 - inbytefile (stream oriented sequential read access)
6 - outbytefile (stream oriented sequential write access)
7 - directbytefile (byte oriented random read/write access)

Note that the types are properties of the files and not necessarily of the data sets. Data sets may usually be organised and accessed in different manners depending upon the target system thus it is only possible to give some general rules for the kind of data set that should be connected to a given file type.

7.2.1 Infile and outfile

The corresponding data sets should be record-oriented sequential; it is implementation dependent whether the record size must be fixed. Mechanisms should exist for opening a closed "outfile data set" as an "infile data set" and vice versa.

7.2.2 Directfile

The corresponding data set should be record-oriented random access; it is implementation dependent whether the record size should be fixed. Some systems may allow the opening of a closed "directfile data set" as an "infile data set", this may however impose an intolerable loss in efficiency in the implementation of either or both types, so this is not demanded as a part of the standard.

7.2.3 Printfile

Seen from Simula a printfile is a subclass of outfile; the implementation need not follow that philosophy. The corresponding (record-oriented) data set should be able to accept printer control information such as eject (see 7.12) etc. Normally one record will correspond to one printer line, but note the allowance made in the definitions of outimage (7.7) and printoutimage (7.13) for overprint and partial printing.

If possible the output to SYSOUT should be printed immediately, since interactive programs normally will expect SYSIN and SYSOUT to be connected to the terminal device. If several printfiles are open simultaneously at most one (SYSOUT) should be printed continuosly so as not to get the output mixed.

7.2.4 Inbytefile and outbytefile

The corresponding data sets should support stream-oriented sequential access and it must be possible to reopen a closed "outbytefile data set" as an "inbytefile data set".

7.2.5 Directbytefile

The corresponding data sets should support byte-oriented random access. In case the underlying system does not support this, the environment interface should provide the necessary buffering etc. to emulate the concept. Although desirable, it is not demanded that a closed "directbytefile data set" may be opened as an "infile data set", or that a closed "outbytefile data set" may be opened as a "directbytefile data set".

7.3 Look up data set

```
profile look_upp:newtag
system look_up:newtag "lookup"
import lu_dsetspec STRING
import lu_filetype INT range 1 7
import lu_imagelength INT
endprofile
```

Dsetspec: Identification of a data set.

Filetype: The type code of the corresponding file.

Imagelength: The length of the images in the file.

The specified data set is searched for (in the file system), following some implementation defined search rules. When found, the permitted access is checked against the accesses implied by the file type parameter.

The imagelength parameter may be necessary for some systems, on other system it may be ignored.

This routine may change the value of the global variable status to one of the values given in app. C.

7.4 Open data set

```
profile open_dsp:newtag
system open_ds:newtag "opfile"
import ods_dsetspec STRING
import ods_dsettype INT range 1 7
import ods_action STRING
import ods_imagelength INT
export ods_key INT range 0 255
endprofile
```

Dsetspec: Identification of a data set.

Dsettype: The type code of the corresponding file.

Action: A copy of the second parameter to the Simula open procedure, see below.

Imagelength: The length of the images in the file.

Key: The key associated with the data set, or zero.

The data set specified by the given dsetspec and filetype is searched for after implementation dependent rules, see 7.1.

If the data set does not exist, a new data set is created, compatible with the given file type. If the type is 1 or 5 this is impossible and status error 19 is set. The created data set must survive close.

The data set is opened and positioned at the first record (or byte). The interpretation of the action parameter is completely implementation dependent and may thus modify this default action (such as open for write append).

A unique key is associated with the data set and exported from the routine. The key will normally be a hitherto invalid key, but if the data set has been opened already the key returned will be the one already validated; status error 6 is set in this case. In all other abnormal situations the key returned must be zero.

Imagelength is the length of the first parameter to Simula procedure open; the implementation may choose to ignore it. For a bytefile, the imagelength will be 0.

This routine may change the value of the global variable status to one of the values given in app. C.

For the specification of the standard defined action strings, see section 7.16.

7.5 Close data set

```
profile close_dsetp:newtag
system close_dset:newtag "clfile"
import cds_key INT range 1 255
import cds_action STRING
endprofile
```

Key: The key associated with the data set.

Action: A copy of the second parameter to the Simula close procedure, see below.

The associated data set is closed, and the link between the data set and the file object is broken, i.e. the key is invalidated. Thus the key cannot be used vis-a-vis the environment unless it is again validated (via open).

Interpretation of the action parameter is implementation dependent in the sense that it may specify actions to be taken in addition to the above mentioned, such as rewind of a tape etc.

This routine may change the value of the global variable status to one of the values given in app. C.

The standard action strings are defined in section 7.16.

7.6 Inimage

```
profile inimagep:newtag
system inimage:newtag "inimag"
import ini_key INT range 1 255
import ini_image STRING
export ini_filled INT
endprofile
```

Key: The key associated with the data set.

Image: Input buffer.

Filled: The number of characters placed in image.

A record is read from the current position of the data set into the image. If the number of characters in the record exceeds the image length, the action taken is system dependent:

- If the system permits partial record read, image.length characters are read, filled := image.length, and status 34 is returned. In this case the next inimage (on this data set) should continue reading from the next position in the partially read record.

- If partial record reading is not possible, status error 12 is set and filled is set to zero; the remainder of the record is in this case skipped.

Except for the case of partial record reading the data set will be positioned at the sequentially next record.

Inimage is legal on infiles and directfiles only.

This routine may change the value of the global variable status to one of the values given in app. C.

If the status returned is non-zero, filled must be zero, except for the partial read case discussed above (status 12).

7.7 Outimage

```
profile outimagep:newtag
system outimage:newtag "outima"
import oti_key INT range 1 255
import oti_image STRING
endprofile
```

Key: The key associated with the data set.
Image: Output buffer.

If the file is of type 2 or 4, image is copied into the record at the current position of the data set, and the data set is positioned at the sequentially next record. On printfiles the image is printed from the current print position, without vertical spacing (i.e. the next print position is the next position on the same line); this is utilised by the Simula breakoutimage procedure.

Outimage is legal on outfiles, printfiles and directfiles only.

This routine may change the value of the global variable status to one of the values given in app. C.

```
profile breakoutp:newtag
system breakout:newtag "breako"
```

```
import br_key INT range 1 255
import br_img STRING
endprofile
```

key: The key associated with the data set

img: A string to be output

The routine Breako will output the string img to the current record of the data set, beginning at the current position. The record should not be "closed" i.e. the next Breako (or Outima) will output to the same record, beginning at the new position. If this is not possible, Breako shall perform as Outima.

On display terminals, Breako will output the string from the current cursor position, and leave the cursor positioned after the last charcter of img. Note that trailing blanks (of img) shall be output.

On other external data sets, it may be necessary for the environment to buffer the output internally.

The routine is legal for outfiles and printfiles only.

7.8 Locate record

```
profile locatep:newtag
system locate:newtag "locate"
import lo_key INT range 1 255
import lo_loc INT
endprofile
```

Key: The key associated with the data set.

Loc: Indicates the next record (directfile) or byte (directbytefile) position to be accessed.

The position of the data set is changed so that the next record read or written will be record number loc of the data set (the first record is numbered 1). Locate is legal on directfiles and directbytefiles only.

In case loc is out of range (i.e. greater than the maximum allowable record number), the implementation may choose to defer the error reporting to the first following in/outimage instead; in that case it is possible to position outside the data set without consequences as long as no actual I/O is requested.

This routine may change the value of the global variable status to one of the values given in app. C.

7.9 Deleteimage

```
profile deletep:newtag
system delete:newtag "delete"
import del_key INT range 1 255
export del_result BOOL
endprofile
```

Key: The key associated with the data set

Result: true if record has been succesfully deleted, false otherwise.

The record at the current position of the data set is deleted. Note that SIMULA does not distinguish between deleted records, and records that never were written. Consequently, the value of Lstloc may change after the deletion.

If Result is true, the data set is positioned at the sequentially next record, otherwise the position is unchanged (this can happen only if the current position of the data set is out of range).

Delete is legal on directfiles only.

This routine may change the value of the global variable status to one of the values given in app. C. Note, however, that end of file conditions (status 15 and 16) shall be reported through the value of Result.

7.10 Get data set name

```
profile get_dsetnamep:newtag
system get_dsetname:newtag "gdsnam"
import gdn_key INT range 1 255
import gdn_dsetname STRING
export gdn_filled INT
endprofile
```

Key: The key associated with the data set.

Dsetname: Filled by the routine, see below.

Filled: The number of characters in dsetname.

The name of the data set associated with the key is returned in the dsetname parameter, and filled is set according to the length. If an error occurs filled is set to zero. The generated name cannot be longer than 80 characters.

Note that the routine returns the actual data set name, which may be different from the data set specification under which the data set was opened.

This routine may change the value of the global variable status to one of the values given in app. C.

7.11 Get data set specification

```
profile get_dsetspecp:newtag
system get_dsetspec:newtag "gdspec"
import gds_code INT range 1 3
import gds_dsetpec STRING
export gds_filled INT
endprofile
```

Code: Indicates the kind of data set searched, see explanation below.

Dsetspec: A data set specification returned here.

Filled: The length of Dsetspec, or zero.

The following file names carry predefined meanings at the Simula level, thus it must be possible to connect to the corresponding data sets:

SYSIN identifies the standard input data set (type infile)
SYSOUT identifies the standard output data set (type printfile).

Furthermore some tracing output data set must be connected if needed; it will be identified by the name "SYSTRACE". The data set is a printfile, but it could well be organized with a circular buffer to avoid extensive output of unwanted information.

This routine is called in order to get a specification of one of these data sets. The resulting data set specification will later be used in calls to routine open.

The interpretation of what dsetspec represents for the different values of code is as follows:

Code	Dsetspec
1	What is the data set specification for SYSIN, the file will be opened as infile.
2	What is the data set specification for SYSOUT, the file will be opened as printfile.
3	What is the data set specification for SYSTRACE, the file will be opened as printfile.

The algorithm used to generate the specification will be highly implementation dependent. The generated data set specification cannot be longer than 72 characters.

This routine may change the value of the global variable status to one of the values given in app. C.

7.12 Get lines per page

```
profile get_lppp:newtag
system get_lpp:newtag "getlpp"
import gl_key INT range 0 255
export gl_lpp INT
endprofile
```

Key: The key associated with the data set.

Lpp: The number of lines per page defined for the indicated data set, or zero.

Printfiles have a (system or user defined) default number of printable lines per page. A call on this routine should result in this value being exported. Note that the number of lines skipped at page top and bottom must be excluded. If an error occurs the value returned should be zero. A negative value indicates no page formatting.

When the routine is called with parameter key=0, the result should be the standard 'lines per page' for this installation.

This routine may change the value of the global variable status to one of the values given in app. C.

7.13 New page

```
profile newpagep:newtag
system newpage:newtag "newpag"
import np_key INT range 1 255
endprofile
```

Key: The key associated with the data set.

The top-of-form action is performed on the data set, so that the next image will be printed on the first printable line of the following page. This is legal on printfiles only.

This routine may change the value of the global variable status to one of the values given in app. C.

7.14 Printoutimage

```
profile print_outimagep:newtag
system print_outimage:newtag "printo"
import po_key INT range 1 255
import po_image STRING
import po_spc INT
endprofile
```

Key: The key associated with the data set.

Image: The image to be printed.

Spc: Vertical spacing, see below.

The image is printed from the current line position, and the data set is positioned according to the value of spc as follows:

spc<0 The data set is backspaced -spc lines; if this is not possible the position is not changed (i.e. as if spc=0) and status 19 is returned.

spc=0 Indicates that the next image should overprint this; if this is not possible the action is as for spc=1 and status 19 is returned.

spc=1 The normal case: the next image is printed on the following line.

spc>1 spc-1 empty lines are printed before the next image.

This routine may change the value of the global variable status to one of the values given in app. C.

7.15 Byte input

```
profile inbytep:newtag
system inbyte:newtag "inbyte"
import ib_key INT range 1 255
export ib_byte INT range 0 255
endprofile
```

Key: The key associated with the data set.

Byte: Value input.

One byte is input from the current position of the data set, and the data set is positioned at the following byte.

```
profile in2bytep:newtag
known in2byte:newtag "in2byt"
import i2b_key INT range 1 255
export i2b_double INT range 0 65535
endprofile
```

Key: The key associated with the data set.
Double: Value input.

The body of the routine may be described as

```
double:= inbyte(key);
double:= double*256 + inbyte(key);
```

This routine may change the value of the global variable status to one of the values given in app. C.

Inbyte and In2byte are legal on inbytefiles and directbytefiles only.

7.16 Byte output

```
profile outbytep:newtag
system outbyte:newtag "outbyt"
import ob_key INT range 1 255
import ob_byte INT range 0 255
endprofile
```

Key: The key associated with the data set.

Byte: Value to be output.

Byte is output to the current data set position and the data set is positioned at the following byte.

```
profile out2bytep:newtag
known out2byte:newtag "out2by"
import o2b_key INT range 1 255
import o2b_double INT range 0 65535
endprofile
```

Key: The key associated with the data set.

Double: Value to be output.

Double is output to the data set as two bytes; the body of the routine may be described as

```
single := double//256;
outbyte(key, single);
outbyte(key, double-single*256);
```

This routine may change the value of the global variable status to one of the values given in app. C.

Outbyte and Out2byte are legal on outbytefiles and directbytefiles only.

7.17 Lastloc and Maxloc

```
profile lastlocp:newtag
system lastloc:newtag "lstloc"
import ll_key INT range 1 255
export ll_val INT
endprofile
```

```
profile maxlocp:newtag
system maxloc:newtag "maxloc"
import ml_key INT range 1 255
export ml_val INT
endprofile
```

Key: The key associated with the data set.

Val: The desired value.

The routine Lstloc will give access to the largest location of any written image in the file referenced by key.

The routine Maxloc will give acces to the maximum value that can be used as parameter to Locate on the file referenced by key.

Lstloc and Maxloc are legal on directfiles and directbytefiles only.

7.18 Checkpoint

```
profile chkpntp:newtag
system chkpnt:newtag "chkpnt"
import ll_key INT range 1 255
export ll_result BOOL
endprofile
```

Key: The key associated with the data set.

Result: Signals the result, see below.

When this routine is called, the environment shall attempt to secure the output produced so far. Depending on the nature of the associated external device, this causes completion of output transfer (i.e. intermediate buffer contents are transferred). If this is not possible, the result false is returned, otherwise the routine returns true.

The routine is legal for outfiles, directfiles, outbytefiles and directbytefiles only.

7.19 Direct file locking

```
profile lockfip:newtag
system lockfi:newtag "lockfi"
import lf_key INT range 1 255
import lf_lim REAL
import lf_loc1 INT
import lf_loc2 INT
export ll_result INT
endprofile
```

Key: The key associated with the data set.

Lim: Time limit (in seconds) for waiting for the file to be available for exclusive access. If the value is less than or equal to zero, the routine should return -1 without performing any file actions.

Loc1: First record or byte to be locked.

Loc2: Last record or byte to be locked.

Result: 0 - the file is locked

-1 - timelimit reached

<-1 - not possible to lock file

The routine Lockfi will lock the file for this application, i.e. it will give the program exclusive access to the dataset in question, to all or part of the file. See SIMULA Standard, 10.2.2.

```
profile unlockp:newtag
system unlock:newtag "unlock"
import ll_key INT range 1 255
endprofile
```

Key: The key associated with the data set.

The routine Unlock eliminates the effect Lockfi might have had on the data set.

Lockfi and Unlock are legal on directfiles and directbytefiles only.

7.20 Data Set Access Specification

The standard attribute procedure "setaccess" is defined in the SIMULA Standard, section 10.1.1. In order to implement the effect of this procedure, Opfile (section 7.4) and Clfile (section 7.5) may have a non-empty 'action' parameter, specifying the actions to be taken when a data set is opened, resp. closed.

The format of a non-empty action string, is w= ;<action> < ; <action> >* where <action> is an identifier specifying the wanted action. Standard S-port action identifiers are defined below, an implementation may add to this list as needed.

The column "at" in the table indicates when the action can occur: 'O' means as parameter to Opfile and 'C' means parameter to Clfile.

An action, if given, overrides an implementation defined default.

Action	at	Definition
SHARED	O	The data set may be shared with other programs.
NOSHARED	O	The data set cannot be shared, i.e. access must be exclusive.

APPEND O Open an existing data set for extension, i.e. position at highest written record + 1 (lastloc+1 on direct(byte)files).

NOAPPEND O If the data set exists, it is logically emptied. This is relevant for direct(byte)files only.

CREATE O If the data set exists, Opfile returns status 4 without further action.

NOCREATE O If the data set does not exist, Opfile returns status 3 without further action.

ANYCREATE O If the data set exists, it is used. If not, a new data set of the given Dsetspec is created.

READONLY O Relevant for direct(byte)files only. If output operations are attempted, status 20 should be returned.

WRITEONLY O Relevant for direct(byte)files only. If input operations are attempted, status 26 should be returned.

BYTESZ:n O 'n' is an integer. Relevant for bytefiles only. Specifies the size (in bits) of the bytes in the file. The interpretation is implementation-defined.

REWIND OC Rewind tape before opening or after close. No effect if data set is not on tape.

NOREWIND OC Do not rewind tape. No effect if data set is not on tape.

PURGE OC Make the data set inaccessible to the program after close.

NOPURGE OC Keep the data set accessible to the program after close.

RELEASE C If the data set is on tape, the tape is unloaded. Otherwise PURGE for scratch files, NOPURGE for other data sets.

NEXTFILE OC Only relevant for data sets on tape. At open, the file at which the tape is positioned is skipped. At close, the tape is positioned at the beginning of the following file (this is normally not possible for output type files).

PREVIOUS C Only relevant for data set on tape. After close, the tape is positioned at the beginning of the data set preceding the data set just closed (i.e. NEXTFILE at next open will position the tape at the data set just closed). If the current data set is the first on the tape, the tape is rewound.

REPEAT C Only relevant for data set on tape. After close, the tape is positioned at the beginning of the data set just closed.

MOVE:n OC 'n' is an integer. Relevant for data sets on tape only. If n=0, REPEAT. If n<0, perform PREVIOUS abs(n) times or until beginning-of-tape. If n>0, perform NEXTFILE n times.

8 TEXT PROCEDURES FROM SIMULA

The following routines are intended to implement the de-editing and editing procedures described in SIMULA Common Base. The numeric items read or produced by the routines are defined by the syntax of numeric items, SIMULA Common Base 10.8.1.

8.1 De-editing routines

The de-editing routines perform the functions described for the corresponding procedures in SIMULA Common Base 10.9.

The global variable itemsize shall be set by each routine to the number of characters in the numeric item read by the routine, including possible leading blanks.

8.1.1 Integers

```
profile get_intp:newtag
system get_int:newtag "getint"
import gi_item STRING
export gi_val INT
endprofile
```

The parameters to the routine have the following interpretation:

Item: This is the string that is subject for de-editing.

Val: This is the resulting integer from the routine.

This routine may change the value of the global variable status to one of the values given in app. C.

8.1.2 Reals

```
profile get_realp:newtag
system get_real:newtag "gtreal"
import gr_item STRING
export gr_val LREAL
endprofile
```

The parameters to the routine have the following interpretation:

Item: This is the string that is subject for de-editing.
Val: This is the resulting long real from the routine.

This routine may change the value of the global variable status to one of the values given in app. C.

8.1.3 Fractions

```
profile get_fracp:newtag
system get_frac:newtag "gtfrac"
import gf_item STRING
export gf_val INT
endprofile
```

The parameters to the routine have the following interpretation:

Item: This is the string that is subject for de-editing.
Val: This is the resulting integer from the routine.

This routine may change the value of the global variable status to one of the values given in app. C.

8.2 Editing routines

The editing routines perform the functions described for the corresponding procedures in SIMULA Common Base 10.10. Extensions to the set in SIMULA Common Base 10.10 are put_fix and put_real. They have the same function as putfix and putreal, but with a real parameter instead of a long real.

Note that long reals are to be edited into a real item.

The string import parameter will not be empty.

If the string is too short to contain the numeric item then this will be signalled through the global variable status, and the string is not filled with asterisks.

8.2.1 Integers

```
profile put_intp:newtag
system put_int:newtag "putint"
import pi_item STRING
import pi_val INT
endprofile
```

The parameters to the routine have the following interpretation:

Item: This is the string that is subject for editing.

Val: This is the integer that is to be edited into the string.

This routine may change the value of the global variable status to one of the values given in app. C.

8.2.2 Fixed format reals

```
profile put_fixp:newtag
system put_fix:newtag "putfix"
```

```
import pf_item STRING
import pf_val REAL
import pf_frac INT
endprofile
```

The parameters to the routine have the following interpretation:

Item: This is the string that is subject for editing.

Val: This is the real that is to be edited into the text.

Frac: This is the number of characters that is to be reserved for the fraction part of the decimal item.

This routine may change the value of the global variable status to one of the values given in app. C.

8.2.3 Fixed format long reals

```
profile put_lfixp:newtag
system put_lfix:newtag "ptlfix"
import pf_item STRING
import pf_val LREAL
import pf_frac INT
endprofile
```

The parameters to the routine have the following interpretation:

Item: This is the string that is subject for editing.

Val: This is the long real that is to be edited into the string.

Frac: This is the number of characters that is to be reserved for the fraction part of the decimal item.

This routine may change the value of the global variable status to one of the values given in app. C.

8.2.4 Floating format reals

```
profile put_realp:newtag
system put_real:newtag "ptreal"
import pr_item STRING
import pr_val REAL
import pr_frac INT
endprofile
```

The parameters to the routine have the following interpretation:

Item: This is the string that is subject for editing.

Val: This is the real to be edited into the string.

Frac: This gives the length of the fraction part of the resulting item.

If the text string is too short to contain the resulting numeric item or the value of the parameter frac is less than zero, this will be signalled through the global variable status when returning from the routine. The possible values of the global variable status are given in app. C.

8.2.5 Floating format long reals

```
profile put_lrealp:newtag
system put_lreal:newtag "plreal"
import pr_item STRING
import pr_val LREAL
import pr_frac INT
endprofile
```

The parameters to the routine have the following interpretation:

Item: This is the string that is subject for editing.

Val: This is the long real to be edited into the string.

Frac: This gives the length of the fraction part of the resulting item.

If the text string is too short to contain the resulting numeric item or the value of the parameter frac is less than zero, this will be signalled through the global variable status when returning from the routine. The possible values of the global variable status are given in app. C.

8.2.6 Fractions

```
profile put_fracp:newtag
system put_frac:newtag "ptfrac"
import pc_item STRING
import pc_val INT
import pc_n INT
endprofile
```

The parameters to the routine have the following interpretation:

Item: This is the string that is subject for editing.

Val: This is an integer to be edited as a grouped item.

N: This parameter specifies the number of digits to follow the decimal point in the grouped item.

If the string is too short to contain the resulting numeric item, this will be signalled through the global variable status when returning from the routine. The possible values of the global variable status are given in app. C.

8.2.7 Additional editing routines

For internal purposes it is necessary to introduce additional editing routines. Specially for debugging of the run-time system these routines are important.

The routines will give the possibility of outputing the simple types in the S-code. Since they are target dependently represented, they can only be output through the environment. We will describe the necessary routines collectively.

```
profile put_sizep:newtag
system put_size:newtag "ptsize"
import ps_item STRING
import ps_val SIZE
endprofile

profile put_oaddrp:newtag
system put_oaddr:newtag "ptoadr"
import po_item STRING
import po_val OADDR
endprofile

profile put_aaddrp:newtag
system put_aaddr:newtag "ptaadr"
import pa_item STRING
import pa_val AADDR
endprofile

profile put_paddrp:newtag
system put_paddr:newtag "ptpadr"
import pp_item STRING
import pp_val PADDR
endprofile
```

```
profile put_raddrp:newtag
system put_raddr:newtag "ptradr"
import pr_item STRING
import pr_val RADDR
endprofile
```

The import parameter val is the simple type which is to be edited into the string item. Editing of the type GADDR will be done through splitting and use of put_aaddr and put_oaddr.

This routine may change the value of the global variable status to one of the values given in app. C.

9 STANDARD PROCEDURES FROM SIMULA

9.1 Random drawing procedures

```
profile basic_drawp:newtag
system basic_draw:newtag "drawrp"
import d_U GADDR
export d_val LREAL
endprofile
```

U: The general address of an INT quantity which will be modified by the routine.
Val: A real number in the interval (0,1), such that 0 <= val < 1.

The routine shall replace the value of the specified integer variable U by a new value according to an implementation defined algorithm, which satisfy the condition stated in the Simula standard that

U(i+1) must be generated on the basis of U(i) alone, i.e. no other elements in the stream of pseudo random numbers should be used in the calculation.

For positive values of U a linear congruential sequence:

U(i+1) = (a*U(i) + c) modulo m i >= 0

may be adjusted to satisfy these conditions, and is recommended. The values of the constants "a", "c" and "m", and the value U(0), are crucial in order to generate "good" sequences; the target system representation of integers influences the choices. We refer to the litterature (e.g. D. Knuth: "The art of computer programming", chapter 3) for a discussion on how to obtain these values.

The exported real (val) is computed as U(i+1)/m.

If U is negative a socalled "antithetic drawing" should be obtained by computing U' and val' from -U in the same manner as for positive U. The new values will then be

```
U := - U';
val := if val' <> 0 then 1-val' else 0;
```

In this case the new value of U will also be negative, while the export real still lies in the interval (0,1).

This routine cannot change the value of status.

9.2 Utility procedures

9.2.1 Date and time

We need a routine to provide the function of date and time. This must be provided by the environment, because the information is only provided from the target system.

```
profile date_and_timep:newtag
system date_and_time:newtag "dattim"
import dat_result STRING
export dat_filled INT
endprofile
```

The result of a call on the routine will be filled into the string. The string should have the following syntax:

"yyyy-mm-dd hh:nn:ss.ppp"

This string is defined according to the ISO standard for providing the date and time. The ISO recommendation for the date specification is ISO 2014-1976, the recommendation for time is ISO 3307-1975, and the concatenation of the two is also in ISO 3307. The different parts of the string are:

yyyy	Four digits specifying the year
mm	Two digits specifying the month
dd	Two digits specifying the day in month
hh	Two digits specifying the hour

nn Two digits specifying the minute within the hour

ss Two digits specifying the second within the minute

ppp At least three digits specifying part of second.

The returned STRING will be filled with this string.

9.2.2 Redefine the lowten character

The character '&' represents the exponent sign in any numeric item to be edited or de-edited, (see the Common Base). This can be changed by the user. The following routine is defined to submit this information to the environment.

```
profile lowtenp:newtag
system lowten:newtag "lowten"
import lt_c CHAR
endprofile
```

The parameter c to the routine is the character that from this point on is used to represent the exponent sign in a numeric item.

If the lowten character is changed, this should also imply that the double lowten ("&&") applied for long real should also be changed.

9.2.3 Clock time

```
profile clocktp:newtag
known clockt:newtag "clockt"
export ct_val LREAL
endprofile
```

The routine returns a long real value specifying the elapsed time in seconds since midnight.

9.2.4 Change decimal mark

```
profile dcmarkp:newtag
system dcmark:newtag "dcmark"
import dc_chr CHAR
export dc_old CHAR
endprofile
```

Chr: The desired character to be applied as decimal mark from now on. Only legal values are '.' and ','.

Old: The previous decimal mark character.

The routine gives the possibility to redefine the value of the decimal mark character, and to get the default in the current system.

9.2.5 Text utility procedures

```
profile blankp:newtag
known blank:newtag "cblnk"
import br_str STRING
endprofile
```

The routine will space-fill the string Str.

This routine may not change the value of the global variable Status.

```
profile movep:newtag
known move:newtag "cmove"
import br_src STRING
import br_dst STRING
endprofile
```

The routine will move the contents of the string Src into the string Dst. The length of Dst must be greater than or equal to the length of Src, if not the value 24 will be assigned to the global variable status.

```
profile textrelp:newtag
known textrel:newtag "txtrel"
import tr_str1 STRING
import tr_str2 STRING
import tr_code INT
export tr_rel BOOL
endprofile
```

Str1: String to the left of the relation

Str2: String to the right of the relation

Code: Encodes relation to be tested (see below)

Rel: Result of evaluation, according to value of parameter Code:

code	result
1	rel := Str1 < Str2
2	rel := Str1 = Str2
3	rel := Str1 <= Str2
4	rel := Str1 > Str2
5	rel := Str1 <> Str2
6	rel := Str1 >= Str2

The routine will compare the two strings as indicated by "code" and return the result of the comparison.

This routine may change the value of the global variable Status.

```
profile stripp:newtag
known strip:newtag "stripp"
import str_Str STRING
export str_filled INT
endprofile
```

The routine will strip off trailing blanks in "Str" and return the position of the last non-blank character.

This routine will not change the value of the global variable Status.

9.2.6 Digit and Letter

```
profile digitp:newtag
known digit:newtag "digit"
import di_arg CHAR
export di_result BOOL
endprofile
```

Result: true if Arg is a decimal digit, else false

```
profile letterp:newtag
known letter:newtag "letter"
import le_arg CHAR
export le_result BOOL
endprofile
```

Result: true if Arg is a letter of the English alphabet, else false

The routines Digit and Letter cannot change the value of the global variable status.

9.3 Mathematical library procedures

The following mathematical library routines are defined to cover the same standard procedures in SIMULA.

These routines may change the value of the global variable status to one of the values given in appendix C.

The routines correspond to the similarly named SIMULA Standard functions.

The routine are defined following one of two possible schemes:

Scheme 1:

```
profile <name>p:newtag
system <name>:newtag "<external name>"
import arg <type>
export val <type>
endprofile
```

Scheme 2:

```
profile <name>p:newtag
system <name>:newtag "<external name>"
import arg1 <type>
import arg2 <type>
export val <type>
endprofile
```

The following routines are defined:

Routine:	Scheme:	Name:	External name:	Type:
Square root	1	sqrt	sqroot	LREAL
Square root	1	rsqrt	rsqroo	REAL
Logaritms	1	ln	logari	LREAL
Logaritms	1	rln	rlogar	REAL
Exponentiation	1	exp	expone	LREAL
Exponentiation	1	rexp	rexpon	REAL
Sinus	1	sinus	sinusr	LREAL
Sinus	1	rsinus	rsinus	REAL
Arctan	1	arctan	arctan	LREAL
Arctan	1	rarctan	rartan	REAL

Cosinus	1	cos	cosinu	LREAL
Cosinus	1	rcos	rcosin	REAL
Tangens	1	tan	tangen	LREAL
Tangens	1	rtan	rtangn	REAL
Arcsin	1	arcsin	arcsin	LREAL
Arcsin	1	rarcsin	rarsin	REAL
Arccos	1	arccos	arccos	LREAL
Arccos	1	rarccos	rarcos	REAL
Arctan2	2	atan2	atan2	LREAL
Arctan2	2	ratan2	ratan2	REAL
Sinh	1	sinh	sinh	LREAL
Sinh	1	rsinh	rsinh	REAL
Cosh	1	cosh	cosh	LREAL
Cosh	1	rcosh	rcosh	REAL
Tanh	1	tanh	tanh	LREAL
Tanh	1	rtanh	rtanh	REAL
Addepsilon	1	raddeps	raddep	REAL
Addepsilon	1	daddeps	daddep	LREAL
Subepsilon	1	rsubeps	rsubep	REAL
Subepsilon	1	dsubeps	dsubep	LREAL

Routine:	Scheme:	Name:	External name:	Type:
IntIntpower	2	iipower	iipowr	INT
RealIntpower *)	2	ripower	ripowr	REAL/INT
RealRealpower	2	rrpower	rrpowr	REAL
RealLrealpower *)	2	rdpower	rdpowr	REAL/LREAL
LrealIntpower *)	2	dipower	dipowr	LREAL/INT
LrealRealpower *)	2	drpower	drpowr	LREAL/REAL
LrealLrealpower	2	ddpower	ddpowr	LREAL
Log10	1	rlog10	rlog10	REAL
Log10	1	dlog10	dlog10	LREAL
Cotangens	1	rcotan	rcotan	REAL

Cotangens	1	dcotan	cotang	LREAL
Modulo	2	mod	modulo	INT

The following known routines are defined, following the same scheme:

Entier	*)	1	rentier	renti	REAL/INT
Entier	*)	1	dentier	denti	LREAL/INT
Min		2	rmin	rmin	REAL
Min		2	dmin	dmin	LREAL
Max		2	rmax	rmax	REAL
Max		2	dmax	dmax	LREAL
Abs		2	iabs	iabs	INT
Abs		2	rabs	rabs	REAL
Abs		2	dabs	dabs	LREAL
Sign	*)	1	rsign	rsign	REAL/INT
Sign	*)	1	dsign	dsign	LREAL/INT

*) The routines have different type of parameters and result as follows:

Routine:	1st param:	2nd param:	Result:
ripower	REAL	INT	REAL
rdpower	REAL	LREAL	LREAL
dipower	LREAL	INT	LREAL
drpower	LREAL	REAL	LREAL
rentier	REAL	-	INT
dentier	LREAL	-	INT
rsign	REAL	-	INT
dsign	LREAL	-	INT

10 EXCEPTION HANDLING

The normal progress in program execution may be interrupted by the detection of an exceptional condition in any part of the system:

- errors detected within RTE service routines (e.g. file opening).

- errors or other exception conditions detected by hardware (e.g. division by zero, CPU time limit).

- from the terminal it may be possible to interrupt the execution in several ways.

- user-specified breakpoints may be set by means of the environment-routine 'breakp' (see chapter 11).

10.1 Exceptions in the environment routines

Normally, exceptional conditions that occur within either an environment interface routine (as defined in this document) or within the target environment, are reported via the global variable "status".

The illegal address/instruction traps (code 8 or 9) should not be reported in this manner; if they occur they will in general indicate implementation errors thus they should be reported as such.

The time limit exceeded interrupt as well as the user interrupt (codes 12 and 11) may occur anywhere in the system; they should always be treated by calling the exception handler.

10.2 The exception handler

An exception detected by hardware will normally suspend program execution, and activate the interrupt monitor sequence in the operating system. It is however necessary that the S-port system gets control in such situations; how this might be done is of course highly target system dependent. In order to ensure an as uniform treatment of exceptions as possible across all S-port systems, the S-port exception handling is distributed between the environment interface and the run time system.

The exception monitor, resident in the environment interface, gains control from the operating system in all cases of exceptions which cannot be reported by a non-zero value of the global variable "status"; it is detailed in the following sections of this chapter. The main purposes of the exception monitor are:

- to activate the exception handler in the run time system.
- when the exception handler returns, switch control to a program address exported from the exception handler.

The exception handler is a routine embedded in the run time system; its main task is to registrate the exception and to provide the program address where continuation of the execution is wanted. This will be the address of a code sequence which leads to appropriate treatment of the exception, e.g. printing of error diagnostics or activation of a system for interactive observation of the execution.

The RADDR of the exception handler will be passed to the environment as a parameter to the routine initialise (see 3.1). The body of the exception handler is defined in the run time system; the profile is

```
profile exception_handlerp:newtag
interface "excpha"
import eh_code INT range 0 13
import eh_message STRING
import eh_addr PADDR
export eh_cont PADDR
endprofile
```

Code: Code for the actual exception condition.

Message: Further information about the exception (only given for code 0 or 11).

Addr: The program address where the exception condition was encountered.

Cont: The address to a point for continuation.

The exception handler will always return control to the exception monitor in the normal manner, explicitly giving an address (in parameter cont) at which execution should continue after exit from the exception monitor; this may or may not be equal to the address imported to the exception handler (in parameter addr).

The possible values for the import parameter code are:

0 Unspecified error condition.

Message contains an implementation dependent string explaining the trap/interrupt.

1 Invalid floating point operation trap.

2 Floating point division by zero trap.

3 Floating point overflow trap.

4 Floating point underflow trap.

5 Inexact result (floating point operation) trap.

The codes 1-5 signal one of the possible traps defined in the IEEE Floating Point Standard (cf. IEEE "Computer" March 1981).

6 Integer overflow trap.

7 Integer division by zero trap.

8 Illegal address trap.

9 Illegal instruction trap.

Codes 8 and 9 always signals an implementation error in either a front end compiler or an S-compiler; they should not occur otherwise.

10 Breakpoint trap.

This trap results from a user-specified breakpoint, see section 11.3.

11 User interrupt.

Message will detail the actual interrupt.

12 CPU time limit exceeded interrupt.

This interrupt must be given high priority; it will lead to a controlled termination of the program (by the run time system). Thus the time limit that provokes the interrupt must be sufficiently smaller than the absolute limit known to the operating system to allow for this.

13 Continuation is impossible.

This code shall be given when it is time to give up, it is further explained below.

14 Start of statement trap.

This trap results from a stmt instruction. See Definition of S-Code, chapter 16, and the definition of the routine 'stmt note' in this document.

15 Array index is out of bounds.

16 Attribute access through none.

Values outside this range will lead to an error termination of the execution.

When control is returned from the exception handler the export parameter cont will indicate what to do as follows:

Cont value: Exception monitor action summary:

<u>nowhere</u> Immediate termination of the execution.

legal Unconditional jump to the continuation address.
address

illegal The exception handler is called once again, with address code=13.

This routine cannot change the value of status.

10.3 Recovery actions

The possible recovery actions that can be performed are highly hardware dependent. It is therefore not possible to specify such actions. It is required that the environment takes the full responsibility for the handling of exceptions, for which recovery actions are implemented. For such exceptions, the exception handler should not be called.

10.4 Implementation of the exception monitor

The environment can be seen as being in one of two different modes, normal mode or exception mode. When execution is initiated the environment is in normal mode, and it continues in this mode until an exception occurs. The environment then enters exception mode and stays in that mode until the exception has been treated by the exception monitor and the exception handler.

While in exception mode the treatment of possible nested exceptions differ from the normal in that all except the interrupts corresponding to codes 11 and 12 signals an error in the exception treatment. In that situation it is very dangerous to continue execution (e.g. the system may enter an infinite interrupt loop) so the program should be terminated directly.

In the algorithmic description of an exception monitor given below it is assumed that it is possible to disable the exception detection mechanism in some target dependent manner, using the primitives "disable" and "enable".

Skeleton of an exception monitor:

```
Disable;
ADDR := the program address of the instruction that was to be executed next when
the exception occured, see note 1;
if in exception mode then begin
    if interrupt ! i.e. time-out or key-in, see note 2;
    then begin
        queue interrupt; ! see note 3;
        Enable, goto (ADDR) end
```

```
else call terminate(3,...);
end ;

Set exception mode;

CODE := the relevant value, see 10.2;
REPEAT:
MESSAGE :- suitable message or notext ;
Enable;
call exception handler( CODE, MESSAGE, ADDR, CONT );
Disable;

if CONT= nowhere then call terminate(3,...);

if interrupt queued then begin ! see note 3;
get the interrupt;
ADDR := CONT; ! see note 1;
CODE := the relevant value, i.e. 11 or 12 ;
goto REPEAT end ;

set normal mode;

if CONT is legal address then
begin Enable, goto (CONT) end
else
begin CODE := 13; goto REPEAT end ;
```

1. The value of ADDR should be such that goto (ADDR) leads to an uninterrupted continuation of the execution. This should be true also when the last instruction that has been executed was some kind of jump instruction (e.g. explicit jump or return from routine).

2. Only nested interrupts are allowed, traps may without exception be regarded as caused by implementation errors and therefore lead to immediate termination.

3. The following interrupts may occur: CPU time limit exceeded and User key-in. It is assumed that time-out has high precedence thus it should supercede any pending key-in interrupt. If several key-in interrupts occur it may be assumed either that the user lost patience with the system or that he/she regretted the previous key-in. In both cases only the latest key-in should be queued. As a consequence it may be seen that the queue employed is a single element queue, i.e. no more than one interrupt is pending.

User program

In Environment Interface:

exception (via OS)> Exception monitor

call exception handler(-)

> ...

RTS

.....

Exception handler <

..... < CONT? >

exit

= <u>nowhere</u>

.....V.....V.....

continue terminate

The figure shows the path of control flow in case of an exception within the user program. Such an exception will give control to some point in the error monitor routine in the environment. When control comes to the exception handler, it decides what will be the appropriate treatment of the exception. Its decision is signalled through the export parameter cont.

This routine cannot change the value of status.

11 TOOLS FOR OBSERVATION OF PROGRAM EXECUTION

The environment must support the needs at S-Code level to observe program execution integrated with an intelligible handling of errors and other exception conditions.

A set of tools for program developement and maintenance should comprise the features found in an interactive debugger combined with more extensive possibilities to trace the control and data flow and to measure time and storage requirements.

11.1 Reference to program point at outermost level

At any time during program execution there exists a set (possibly empty) of nested routine activations. The routine activation which first (in time) entered this set is called the outermost one.

The program point at which the currently outermost routine was activated is of particular interest for the user. At any time this gives a reference to the statement at user program level giving rise to the action taking place.

The routine get outermost is defined to satisfy this need:

```
profile get_outermostp:newtag
system get_outermost:newtag "gtoutm"
export gom_result PADDR
endprofile
```

The result of a call on this routine will be the program address at which the currently outermost routine activation was performed, seen from the body of the get_outermost routine.

Note that, seen from the body of this routine, the set of nested routine activations is never empty. A nonzero status should never occur.

Also note that in this sense the handling of any exception condition (e.g. errors detected by hardware, user interrupts) should be considered as a routine activation. In these cases 'get_outermost' will be used to get the PADDR of the program point at user program level to be referenced in error diagnostics etc.

11.2 Correspondence between line numbers and PADDR values

Error diagnostics and other messages in a dialogue with the user depends on the possibility to identify program points. This is prepared for in the S-Code by the definition of the line, decl or stmt instructions, giving the possibility to establish the connection between PADDR values and the line numbering found in source-listings produced by the S-Code generator (e.g. the Simula front end compiler).

In the communication with the user a program point is identified by a line number, together with an identification of the corresponding source program module. The program system may consist of several modules with line-numberings resulting from separate compilations (see the Definition of S-Code, chapter 1.3).

In the communication with the run time environment a program point is identified by a PADDR value.

The routines 'get_line_no' and 'get_paddr' will satisfy the need of the observation system to get the correspondence.

The environment is assumed to establish this correspondence as if a table of the numbers given in line, decl or stmt instructions and corresponding PADDR values were set up in the following way:

By the generation of target code a PADDR-counter exists, giving the location of the next target instruction to be produced. When a line, decl or stmt instruction is encountered the given line number is associated with the PADDR value of that counter. The value of the counter by the end of code generation will be the upper limit for PADDR values associated with the present module.

The line numbers given in succesive line, decl or stmt instructions will occur in increasing order. The first and last line, decl or stmt instruction encountered in a module defines the interval of valid line-numbers for that module. Note that there may be line-numbers in that interval for which no corresponding line, decl or stmt instruction appears, and thus will not appear in the line-number table.

In this manner each module will be associated with an interval of corresponding PAADR values. For two different modules the two corresponding PADDR intervals will never overlap.

When a module name and a line number n is given in a call on get_paddr, the part of the table corresponding to that module is searched for the smallest line number m greater or equal to n. The PADDR associated with m is returned as result. If no such m is found this is reported to the caller by a nonzero status.

When a PADDR is given in a call on get_line_no, the table is searched for the greatest PADDR q such that q <= p . As result the line number with which q is associated is returned. If there is no module in the table for which p is in the interval of associated PADDR values, this is reported through a nonzero status.

Routine definition:

```
profile get_paddrp:newtag
system get_paddr:newtag "gtpadr"
import gpa_module STRING
import gpa_line INT
export gpa_addr PADDR
endprofile
```

Module: The module name. If the empty string is given the main program is assumed.

Line: The line number in the given module for which a corresponding PADDR value is wanted.

Addr: The PADDR value resulting according to the rules above.

This routine may change the value of the global variable status to one of the values given in app. C.

Routine definition:

```
profile get_line_nop:newtag
system get_line_no:newtag "gtlno"
import gln_addr PADDR
```

export gln_lineno INT
endprofile

Addr: A program address for which we want the corresponding source line number.

If this parameter has a value which is not associated with a source line, then status 27 should result.

Lineno: Source line number corresponding to Addr.

The routine will return the line number corresponding to the given program address, as it originally was in the source text.

This routine may change the value of the global variable status to one of the values given in app. C.

profile get_line_identp:newtag
system get_line_ident:newtag "gtlnid"
import gli_adr PADDR
import gli_result STRING
export gli_filled INT
endprofile

Adr: A program address for which we want the line identification.

Result: The resulting identification of the source line.

Filled: The number of characters filled in Result.

The S-code instruction line establish a connection between a program address and a source line number.

This routine will convert an address into a complete identification of the corresponding source line. This identification must include the name of the module (the "module id") in which the line occurs.

If the program address specified does not identify a line number in any module, the global varibale status is given the value 19. If the string "result" is too short to contain the identification, the string is filled and status is given the value 24.

11.3 Implementation of static breakpoints

In order to implement the interactive debugging system it must be possible to set or reset a static breakpoint. The routine 'breakpoint' is defined to enable an efficient implementation of this.

```
profile breakpointp:newtag
system breakpoint:newtag "brkpnt"
import brk_addr PADDR
import brk_sw BOOL
endprofile
```

The parameters to the routine have the following interpretation:

Addr: The program point at which a breakpoint is asked to be set or reset. At the breakpoint the exception handler (see chapter 10) should be called with code = 11 (key-in interrupt) and Addr as import parameters. If the PADDR value does not correspond to a program point within the executing program status 27 should result. The addr should always be an address that is achieved through an earlier call on get_paddr.

Sw: On/Off

If Sw=true (On) and a breakpoint is already set at Addr, status 18 should result. Accordingly, if Sw=false (Off) and no breakpoint is set at Addr, status 37 should result.

When a breakpoint is set at Addr, the environment is asked to force an exception each time that program point is reached by calling the exception handler immediately before the instruction at Addr is executed.

Some implementations may feel it necesary to restrict the number of breakpoints that can be in effect simultaneously.

This routine may change the value of the global variable status to one of the values given in app. C.

11.4 <u>Statement Start Exceptions</u>

In order to efficiently and safely implement attention interrupts and stepwise execution of Simula statements, an observation tool must be able to get control each time a Simula statement is about to be executed.

The stmt instruction may be used to conditionally notify the processor executing the code, that a Simula statement is about to be executed, by causing a start of statement exception.

The stmt instructions may be part of the S-Code, even if the stmt instructions will not be used to generate start of statement exceptions. At compile-time the S-Code compiler may optionally be instructed that start of statement exceptions should never be generated, see section 4.1.

If not told otherwise, the S-Code compiler should generate code which conditionally generates an exception, at a point in the code corresponding to the stmt instruction. This point will correspond to the start of a Simula statement. At run-time a flag controls whether the exception should be generated or not. This flag is set on and off by an environment routine "stmt note". It will be called with argument value TRUE if the flag should be set on, and with the argument value FALSE if the flag should be set off. If the flag is on, then the exception handler should be called with a code indicating a start of statement exception. See section 10.2.

The profile for the routine is:

<u>profile</u> stmt_notep:newtag

<u>system</u> stmt_note:newtag "stmnot"

<u>import</u> stmt_on_off INT <u>range</u> 0 1

<u>endprofile</u>

11.5 <u>Communication with the user</u>

<u>profile</u> sysprip:newtag

<u>known</u> syspri:newtag "syspri"

<u>import</u> sp_img STRING

```
endprofile
```

Img: A string to be directly output to the debugging device.

The routine Syspri is used to give messages to the user, from the run time system, e.g. SIMOB. It is important that this output always reaches the user, since it may be an error message. Consequently, this routine is not allowed to change the value of the global variable status, and it should not itself generate any exceptions.

The "debugging device" is an i/o device defined by the environment. It is normally the combination of SYSIN and SYSOUT (e.g. the user's terminal).

```
profile sysprop:newtag
known syspro:newtag "syspro"
import sp_msg STRING
import sp_img STRING
export sp_filled INT
endprofile
```

Msg: A string to be directly output to the debugging device.

Img: A string to be filled by the user at the debugging device.

Filled: The number of characters filled in Img by the user.

The routine Syspro is used to prompt for input from the user.

Note: The supplied body for these routines will not work fully according to this description; only the environment inclusion can do that. The routines will connect to sysin/sysout.

11.6 Dump routine for debugging of RTS

This routine is only meant to help the debugging of the rts, and may of course be implemented with an empty body if the rts debugging is not relevant:

```
profile dmpobjp:newtag
system dmpobj:newtag "dmpobj"
import dmp_filekey INT range 1 255
import dmp_obj OADDR
import dmp_lng SIZE
endprofile
```

Key: This is the internal identification of a data set. It must refer to an open printfile, otherwise the routine will have no effect.

Obj: This parameter specifies where the dump starts.

Lng: This parameter specifies the size of the dump.

The routine will give a dump of the area beginning at `obj` and ending at `obj+lng-1`.

This routine may change the value of the global variable status to one of the values given in app. C.

11.7 Processor Time Usage

In order to measure the usage of cpu-time efficiently, the following routine is defined:

```
profile cpu_timep:newtag
system cpu_time:newtag "cputim"
export time_used LREAL
endprofile
```

It returns the elapsed cpu-time measured in seconds. The difference between two calls will be interpreted as the elapsed time between the two calls.

Status 19 will result if it is not possible to measure the processor time usage.

11.8 Hashing routine

```
profile hashp:newtag
known hash:newtag "hash"
import h_str STRING
export h_val INT range 0 255
endprofile
```

Str: A string to be "hashed"

Result: Integer computed from "str"

The routine will from a string produce an integer value in the given range. Hash is e.g. used by the compiler to compute unique values for each symbol of the source text, and it is provided to enable an implementation to take full advantage of the machine architecture.

The string str may contain any character, not only letters. It should be tested that the routine has sufficient quality, for instance by studying possible collisions between SIMULA keywords and standard identifiers.

Listing of the Environment Module

The following is a "complete" listing of the environment module.

```
global module "comn" "update-no"
global status INT range 0 36 system "status"
global itemsize INT range 0 72 system "itsize"
const maxlen SIZE nosize system "maxlen"
const inptlinelng INT c-int "0" system "inplth"
const ouptlinelng INT c-int "0" system "outlth"
```

```
record STRING
attr chradr GADDR
attr nchr INT
endrecord
profile initialisep system initialise "initia"
profile terminatep system terminate "termin"
profile get_intinfop system get_intinfo "gintin"
profile get_textinfop system get_textinfo "gtexin"
profile get_sizeinfop system get_sizeinfo "sizein"
profile give_intinfop system give_intinfo "gviinf"
profile give_textinfop system give_textinfo "givinf"
profile def_work_areap system def_work_area "dwarea"
profile movep system move "movein"
profile look_upp system look_up "lookup"
profile open_dsp system open_ds "opfile"
profile close_dsetp system close_dset "clfile"
profile inimagep system inimage "inimag"
profile outimagep system outimage "outima"
profile locatep system locate "locate"
profile get_dsetnamep system get_dsetname "gdsnam"
profile get_dsetspecp system get_dsetspec "gdspec"
profile get_lppp system get_lpp "getlpp"
profile newpagep system newpage "newpag"
profile print outimagep system print outimage "printo"
profile inbytep system inbyte "inbyte"
profile in2bytep known in2byte "in2byt"
profile outbytep system outbyte "outbyt"
profile out2bytep known out2byte "out2by"
profile get_intp system get_int "getint"
profile get_realp system get_real "gtreal"
profile get_fracp system get_frac "gtfrac"
profile put_intp system put_int "putint"
profile put_fixp system put_fix "putfix"
profile put_lfixp system put_lfix "ptlfix"
profile put_realp system put_real "ptreal"
profile put_lrealp system put_lreal "plreal"
profile put_fracp system put_frac "ptfrac"
profile put_sizep system put_size "ptsize"
profile put_oaddrp system put_oaddr "ptoadr"
profile put_aaddrp system put_aaddr "ptaadr"
profile put_paddrp system put_paddr "ptpadr"
profile put_raddrp system put_raddr "ptradr"
profile basic_drawp system basic_draw "drawrp"
profile date_and_timep system date_and_time "dattim"
profile lowtenp system lowten "lowten"
profile sqrtp system sqrt "sqroot"
profile rsqrtp system rsqrt "rsqroo"
profile lnp system ln "logari"
profile rlnp system rln "rlogar"
profile expp system exp "expone"
profile rexpp system rexp "rexpon"
profile sinusp system sinus "sinusr"
profile rsinusp system rsinus "rsinus"
profile arctanp system arctan "arctan"
profile rarctanp system rarctan "rartan"
profile cosp known cos "cosinu"
profile rcosp known rcos "rcosin"
profile tanp known tan "tangen"
profile rtanp known rtan "rtangn"
profile arcsinp known arcsin "arcsin"
profile rarcsinp known rarcsin "rarsin"
profile arccosp known arccos "arccos"
profile rarccosp known rarccos "rarcos"
profile exception_handlerp interface "excpha"
import eh_code INT range 0 13
import eh_message STRING
import eh_addr PADDR
export eh_cont PADDR
endprofile
profile get_outermostp system get_outermost "gtoutm"
profile get_paddrp system get_paddr "gtpadr"
profile get_line_nop system get_line_no "gtlno"
profile breakpointp system breakpoint "brkpnt"
profile stmt_notep system stmt_note "stmnot"
profile dmpobjp system dmpobj "dmpobj"
profile cpu_timep system cpu_time "cputim"
global encdrv OADDR
global curdrv OADDR
......
tag Status 0
tag Itemsize 1
......

body
......
endmodule
```

Errors reported by status code

Errors detected within RTE service routines (e.g. file opening) are normally reported as a non-zero status of the routine. The actions taken on the basis of this status will be defined at the S-code level. The meaning of the status codes that may be generated from the I/O subsystem of the environment interface, is in most cases evident from the short explanatory text given.

Note that in some cases more than one type of error may occur; no assumptions are made about the precedence of the status codes in general.

The types of errors that can occur, and give control back in this way, are illustrated by the following list of status codes:

0 (not used)

Zero can never be returned from a routine. In case everything is OK the value of status is not changed.

1 Invalid filekey

The key is within the interval 1..255, but no data set is associated with the key. The file may have been closed, and consequently the filekey is again undefined.

2 File not defined

No real file associated with local name.

The data set specification does not correspond to either a descriptor name or a data set name.

3 File does not exist

The file association has been given, but the fysical file specified does not exist. The data set specification refers a descriptor but this descriptor does not identify a data set.

4 File already exists

An attempt has been made to create a file which already exists. Some systems could allow you to define several files with the same name, e.g. scratch files. This should not occur.

5 File not open

An operation on a file is asked for, but the file is not open.

6 File already open

A request for file opening has been made on a file which is already open.

7 File already closed

For some exterior reason the data set has been closed outside the control of the Simula system, (e.g. a tape has been dismounted by the operator).

8 Illegal use of file

The data set organisation is incompatible with the wanted usage as given in filetype, e.g. an attempt to read from an outfile.

9 Illegal record format for directfile

The external record format is not compatible with the directfile definition.

10 Illegal filename

The string specified does not follow the syntax of a file name in this system.

11 Output image too long

The image length is longer than the file record on an attempt to write on the file.

12 Input image too short

When reading from a file, the image is not large enough to hold the complete record to be read.

13 End of file on input

When reading from a file, the end of file record was read.

14 Not enough space available

When work space is asked for, and the specified amount of storage cannot be allocated.

15 File full on output

When writing to a file, the space allocated to the file is exhausted, and no more space can be furnished.

16 Location out of range

When reading from a file, the specified record in the directfile has never been written. When writing or positioning in a file, the specified location will bring us outside the area reserved for the file.

17 I/O error, e.g. hardware fault

Any hardware detected error which does not refer to an error done by the user.

18 Specified action cannot be performed

The specified action for open file or close file has not been implemented, and consequently cannot be performed.

19 Impossible

This will mean that it is impossible to implement the the specified effect, or that the request has been defined illegal.

This status is returned as a signal to the run time system that it need not bother to try recovery, the program should be aborted.

There will normally be a separate specification of the interpretation of this code under each routine that can give this return value.

20 No write access to this file

Writing has been requested to a file that has been protected against writing.

21 Non-numeric item as first character

The de-editing of a string to a numeric item has been requested, but the string does not start according to the syntax of a numeric item.

22 Value out of range

The de-editing of a string to a numeric item has been requested, but the result is a numeric item that is to large to be represented in the specified type.

23 Incomplete syntax

The de-editing of a string to a real item has been requested, but the string does not complete a real item according to the syntax of a real item.

24 Text string too short

The editing of a numeric item into a string has been requested, but the string is too short to contain the result of the editing operation.

25 Fraction part less than zero

The editing of a real as a floating point or fixed point real has been requested, but the fraction part has been specified with a negative length.

26 No read access to this file.

Reading has been attempted on a read-protected file.

27 Argument out of range for system routine

The code refers mainly to the matematical library routines, and indicate that one of the arguments were out of range.

28 Key previously defined

This specifies that the generation of a key has been made for a file which already has a key referencing it.

29 Maximum number of keys exceeded

The S-port system restricts the number of files that may be open simultaneously to 255, it is however expected that the target system's limit is lower. If any of these limits are exceeded this status is returned from open.

30 This service function is not implemented

One of the give_ or get_ routines have been called with an index which is not known in this implementation, or which has not been implemented. Some default value will be assumed.

31 Syntax error in dsetspec

32 No read access

33 Illegal action

34 Partial record read.

35 Undefined record (on directfile).

36 Maximum number of breakpoints set

STATUTES
for the
STANDARDS GROUP
for the Portabls SIMULA System.

Article 1. Definitions

The Portable SIMULA System (PSS) consists of a language dependent part and a target dependent part. The interface between these two parts is at any time defined in the documents:

S-PORT: Definition of S-code

S-PORT: The Environment Interface

The two documents comprise the PSS interface definition.

An S-compiler system is a system which is able to translate and execute programs represented according to the PSS interface definition.

Article 2. Objectives

The Standards Group for the Portable SIMULA System (SG/PSS) is an organisation which at all times shall:

- be the final arbiter in the interpretation of the PSS interface definition and be a center for custody of this formal definition.

- provide a forum for discussion and exchange of information relating to the PSS interface definition and its support.

- standardise the PSS interface definition and modify the definition when this is found necessary.

Article 3. Membership

Membership is open to organisations and firms responsible for the maintenance and support of an S-compiler system in active use. Any organisation may apply and be voted a member of the SG/PSS.

The Norwegian Computing Center (NCC), Oslo, Norway and the Edinburgh Regional Computing Center (ERCC), Edinburgh, Scotland are ex officio members of the SG/PSS. The NCC will also act as the secretariat of the SG/PSS.

The SG/PSS can offer membership to individuals, in recognition of their contribution to the SG/PSS work.

Once granted, an SG/PSS membership lasts until:

- it is resigned by the member, or

- it is revoked by the SG/PSS because the conditions under which it was granted cease to exist or the member acts against the objectives of the SG/PSS.

There is no membership fee for the SG/PSS. Members must cover their own expences.

Article 4. Representation, Voting and Meetings

Each member shall appoint one person to be his/her representative in the SG/PSS. The duration of appointment is determined by the members.

The SG/PSS shall meet once every year for an Annual Meeting. This meeting will, in addition to possible administrative matters, handle proposals related to the PSS interface definition. The Annual Meeting shall also elect one of the member's representatives as Chairman for the Standards Group.

Decisions can only be taken regarding matters on the agenda presented to the members at least 3 weeks before the meeting, unless all members present agree otherwise.

In addition to the Annual Meeting SG/PSS may have extraordinary meetings. Such meetings are held when the Chairman finds it necessary, or when this is approved by a majority of the members.

To constitute a quorum, all members of the SG/PSS shall be notified of the meeting and a majority of the members representatives shall be present or give their votes by mail, including the NCC and ERCC representatives.

Decisions by SG/PSS are made by a majority vote among the representatives taking part in the vote. Changes to the statutes for the SG/PSS or a decision to dissolve the SG/PSS require 4/5 majority. Any decision requires the consent of the NCC.

The SG/PSS meetings are open to non-representatives or non-members. Observers have no voting rights, and must apply to the Chairman of the SG/PSS for each meeting they wish to attend.

Article 5. Effectuation and revisions

These statutes were adopted at the Foundation Meeting of the Standards Group for the Portable SIMULA System in Edinburgh the 3rd March 1981 and come into effect immediately.

Formal rules of the SG/PSS operation

1. The main task of the SG/PSS is the maintenance of the PSS interface definition. Its work consists of:

a) a clarification of obscure parts of the definition.

b) removal of eventual conflicts in the definition.

c) alteration of the definition when necessary.

2. The following types of changes in the definition can be directly considered by SG/PSS:

a) obvious oversights that have occurred in the text of the definition.

b) removal of language restrictions that are proved obsolete for consistency and implementation.

c) trivial extensions to the existing concepts that are felt relevant for continued use of the definition in changing environments.

d) any other changes as long as none of the members are against it.

3. Any other changes than those mentioned in point 2 above can also be considered. These changes must, however, be submitted to the secretariat two (2) months prior to the meeting. Submitted proposals will be distributed to the members without delay, to facilitate inclusion of relevant comments in the material presented with the meeting agenda at least three (3) weeks before said meeting.

4. All proposals for changes in the PSS interface definition conforming with the above rules must be formulated in writing in a concise manner and submitted to the secretariat. Anyone may submit such a proposal.

5. The Chairman of the SG/PSS is responsible for confirming receipt of each proposal, registering it and scheduling its processing at one of the meetings of the SG/PSS. Alternatively the Chairman may point out any inadequacies in a proposal to its submitter.

The proposal will be announced at the subsequent SG/PSS meeting which may approve or revoke the Chairman's decision related to this proposal.

Complete material related to a proposal will be submitted to the members when the proposal is processed at the next meeting or by specific request.

6. In its final form every proposal will be an updating text to one of the documents comprising the PSS interface definition. It will further indicate the original submitter, date of submission and its motivation. Alternative forms of the proposal and reasons for their rejection are a valuable part of the document. An example of a suitable form is attached to these rules.

The logical consistency of the text, its clarity and conciseness are of utmost importance. To this end the SG/PSS or its Chairman may allocate one particular member to bring the proposal into the required shape if this is deemed necessary.

7. It is the responsability of the Chairman to notify the submitter of a proposal about the result of its processing by the SG/PSS if this is not otherwise obvious. It is also his/her responsibility to minimize the time taken over each proposal.

8. The final text of the proposals will be available from the secratariat as a supplement to the PSS interface definition until they are incorporated into the definition at the next revision of the documents. To facilitate this process, at its approval a proposal will be assigned a number reflecting which of the documents it refers to and its chronological order.

Proposal for changing the Portable SIMULA System interface definition.

Document affected:

Submitter:

Date:

Title of proposal:

Affected section:

Proposal:

Motivation:

SG/PSS decision on the above proposal

For:
Against:
Abstained: