



Portable Simula Revisited
Programmer's Reference

(as of mai 2023)

Table of Contents

1. INTRODUCTION

2. Download and Install the System

3. Activating the Simula Compiler

3.1 The Simula Editor

4. File Handling Conventions

4.1 File Naming

4.2 File Opening

4.3 File Initialization

4.4 File Closing

4.5 Access Mode Treatment

4.6 File Locking

4.7 Checkpointing a File

5. Separate Compilation

5.1 Attribute Files

5.2 Attribute Information Elements

5.3 Compatible Recompilation

6. Additional Facilities

6.1 Language Extensions

6.2 Additional Standard Procedures

6.3 Additional Standard Classes

6.3.1 Additional Standard Class Drawing

6.3.2 Additional Standard Class DEC_Lib

6.3.3 Additional Standard Class CatchingErrors

6.4 Directive lines

6.5 Conditional Compilation

7. Implementation Defined Aspects

7.1 Language Extensions

7.2 Allowed Implementation Restrictions

7.3 Implementation Dependent Characteristics

7.4 Implementation Defined Characteristics

1. INTRODUCTION

This is a new Simula System created by the Open Source Project 'Portable Simula Revisited'. The project was initiated as a response to the lecture held by James Gosling at the 50th anniversary of Simula in Oslo on 27th September, 2017. He mentioned that most programming Languages have been re-implemented in Java, but not Simula.

Therefore this Portable Simula System is written in Java and compiles to Java code with one exception; the Goto Statement need to be corrected in the byte code, which is done automatically.

Simula is considered the first object oriented programming language. It was originally designed for solving discrete event simulations as an extension to the Algol Language. But it quickly turned out to be a general purpose programming language. Among the main advantages of Simula are the introduction of the '**class**' declaration which lead to very cost-effective program development, and The standardisation of the language which means that Simula programs are easily transferred Between different Simula Systems.

One of the most important aspects of Simula is the possibility to define and compile a set of tools oriented towards some specific application. Three important examples of this possibility are included as standard facilities: The standard input-output package (chapter 10), list processing (chapter 11) and the package for discrete event simulation (chapter 12).

This Portable Simula System handles the full Simula language as defined by the Simula Standard adopted by the Simula Standards Group, except as noted in a later section of this chapter ("Deviations from Standard SIMULA").

THE PORTABLE SIMULA SYSTEM

The Portable SIMULA system is composed of three main entities: the Simula Editor, the Language Compiler, and the Execution Environment.

The program source is composed as described in the Simula Standard. It consists of statements which represent logical steps of a problem solution, and declarations which specify the representation of data operated upon by these statements.

The SIMULA Compiler

The compiler analyses the program and generates an equivalent set of Java Class Files, containing instructions corresponding to the SIMULA statements. The main output of the compilation consists of a single runnable .jar file. Also included in the .jar file is the runtime system which is a thin layer above Java's jdk. In this way, the .jar file can be executed also when only java runtime is available on the target computer.

In addition the compilation of a separately compiled module leads to output of so-called attribute elements. This is described in more detail in chapter 5.

Deviations from '*Simula Standard*'

The current version of this SIMULA system differs from the *Simula Standard* in some areas. Following the chapters of *Simula Standard*, this is a complete list:

- 6 External non-SIMULA procedures are NOT IMPLEMENTED.
- 9 The <real-type> parameter to the 'histsd' procedure must be type real.
- 10 The File procedure setaccess is not fully implemented

An extra access mode 'CHARSET:<charset name>' is available for image files to specify the character set used to encode/decode the file content.

Implementation Defined Aspects

Within the Simula Standard, certain aspects of the language are left open to be decided by the actual implementation. Chapter 8. gives a complete list of these, together with a specification of the decision taken on each subject for this Portable Simula System.

Information on how to install the system and about system limitations etc. is found in chapter 2.

About this Manual

This manual has repeated references to the Simula Standard. Note that the Standard was not intended as a text book on Simula, therefore it is rather formal in style in order to be as precise as possible. We refer to one of the text books on the market, for instance "An Introduction to programming in Simula" by Rob Pooley.

The remainder of the manual contains implementation dependent information of a practical nature, such as e.g. how to compile and execute a source program using the Simula Editor.

In addition information is given on system limitations, file search strategies etc.

2. Download and Install the System

To download and install this Simula System you may visit the web-page:

<https://portablesimula.github.io/github.io/>

Select 'Simula Download' and follow the instructions.

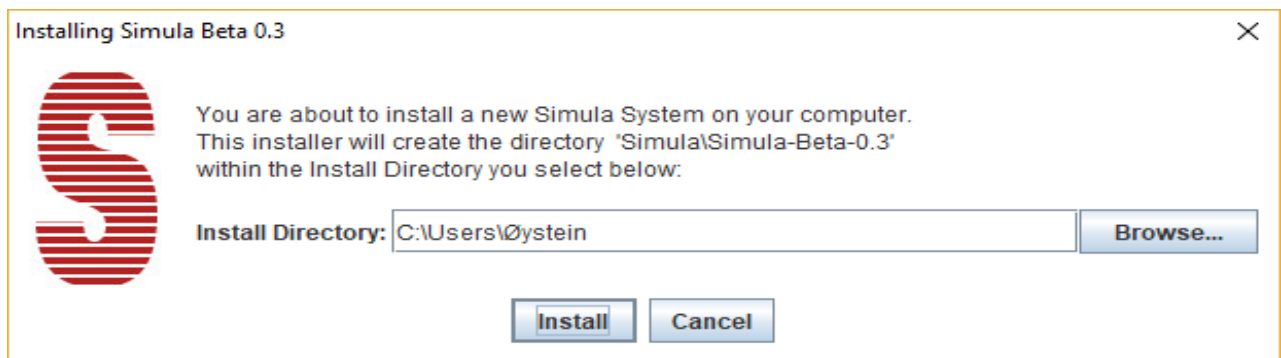
Just when your computer is about to download the setup file it probably gives a warning message. This message states that you are downloading from a unsafe source.

You must ignore this message.

When the 'setup.jar' is completely downloaded its time for executing it. This is done by clicking on it on Windows PC's.

< add description for Unix, MAC, Linux ... >

When setup.jar is running you are first asked to select the install directory:



In this case the Simula System will be installed in the following directory structure.

C:\Users\Øystein

\Simula	(This Directory is called 'Simula Home')
\Simula-Beta-03	(This Directory is called 'Release Home')
\rts ...	(Simula Runtime System)
\samples ...	(Sample Simula Programs)
simula.jar	(Executable Simula Compiler)
ReleaseNotes.txt	

3. Activating the Simula Compiler

The Simula Compiler is normally activated through a command-line of this form:

```
java -jar release-home\simula.jar
```

In this simple case the Simula Editor (see the following pages) is started.

On Windows an activation file `RunSimulaEditor.bat` is created during setup and placed in the Release Home directory together with the 'simula.jar' file. On Unix, Linux, MAC etc. a corresponding file `RunSimulaEditor.sh` is generated.

If an error message including a problem such as 'permission denied' appears, use `sudo` to run it as root (admin). Be careful, `sudo` allows you to make critical changes to your system. See: <https://en.wikipedia.org/wiki/sudo>

You can move these script files wherever you want, for example, to the desktop. Alternatively, on Windows you may create a link to 'simula.jar' and put it on the desktop. Icons are available in the \icon sub-directory if you want to decorate such 'buttons'.

General Case

In special rare situations you may use the general version of the command-line form:

```
java [java-options] -jar release-home\simula.jar [ simula-sourceFile ] [simula-options]
```

Java-options

are described in the relevant Java Technical Dokumentation.

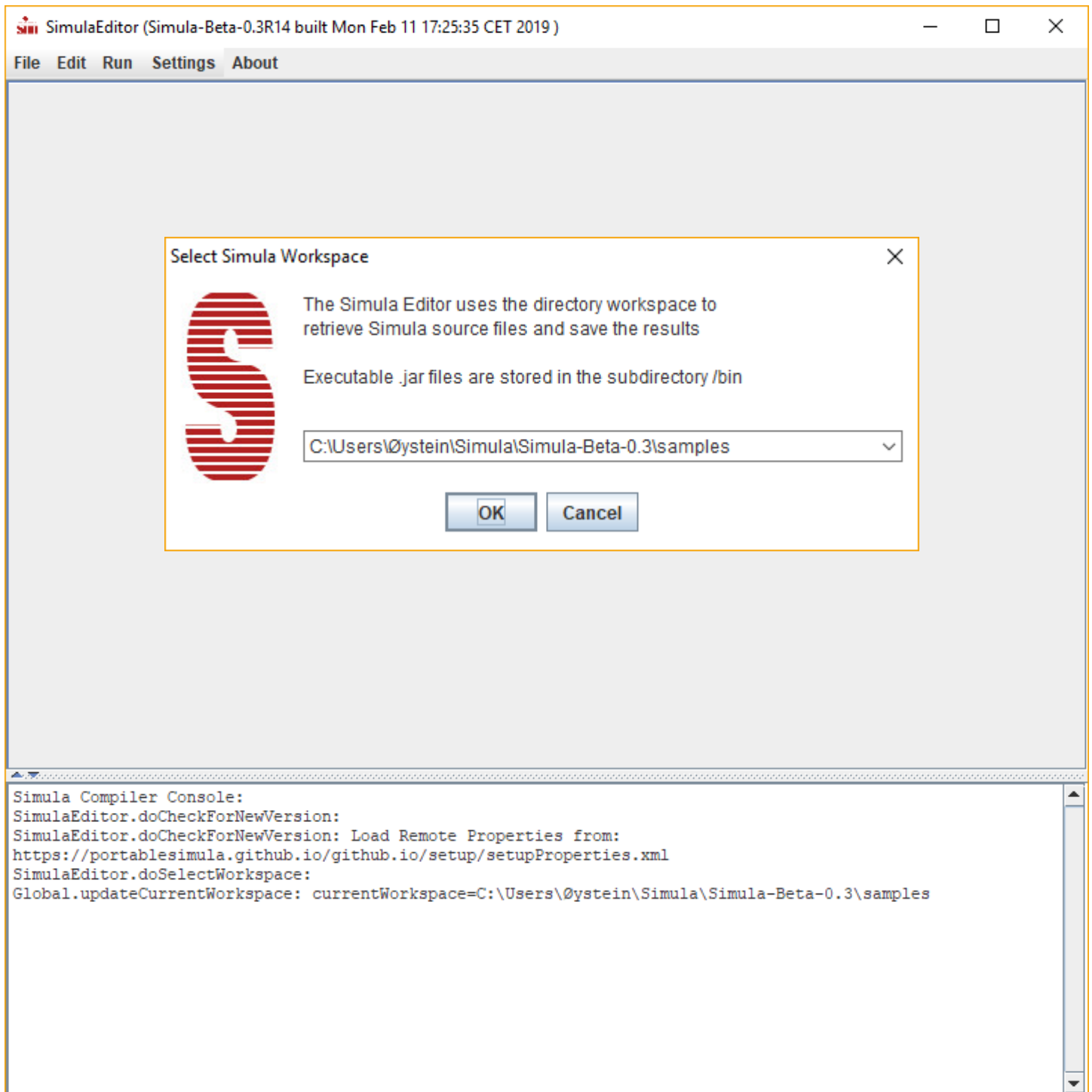
Simula-sourceFile

is the file containing the Simula text to be compiled and runed.

possible simula-options include:

-help	Print this synopsis of standard options
-noexec	Don't execute generated .jar file
-nowarn	Generate no warnings
-verbose	Output messages about what the compiler is doing
-keepJava <directory>	Specify where to place generated .java files Default: Temp directory which is deleted upon exit
-output <directory>	Specify where to place generated executable .jar file Default: Current workspace\bin

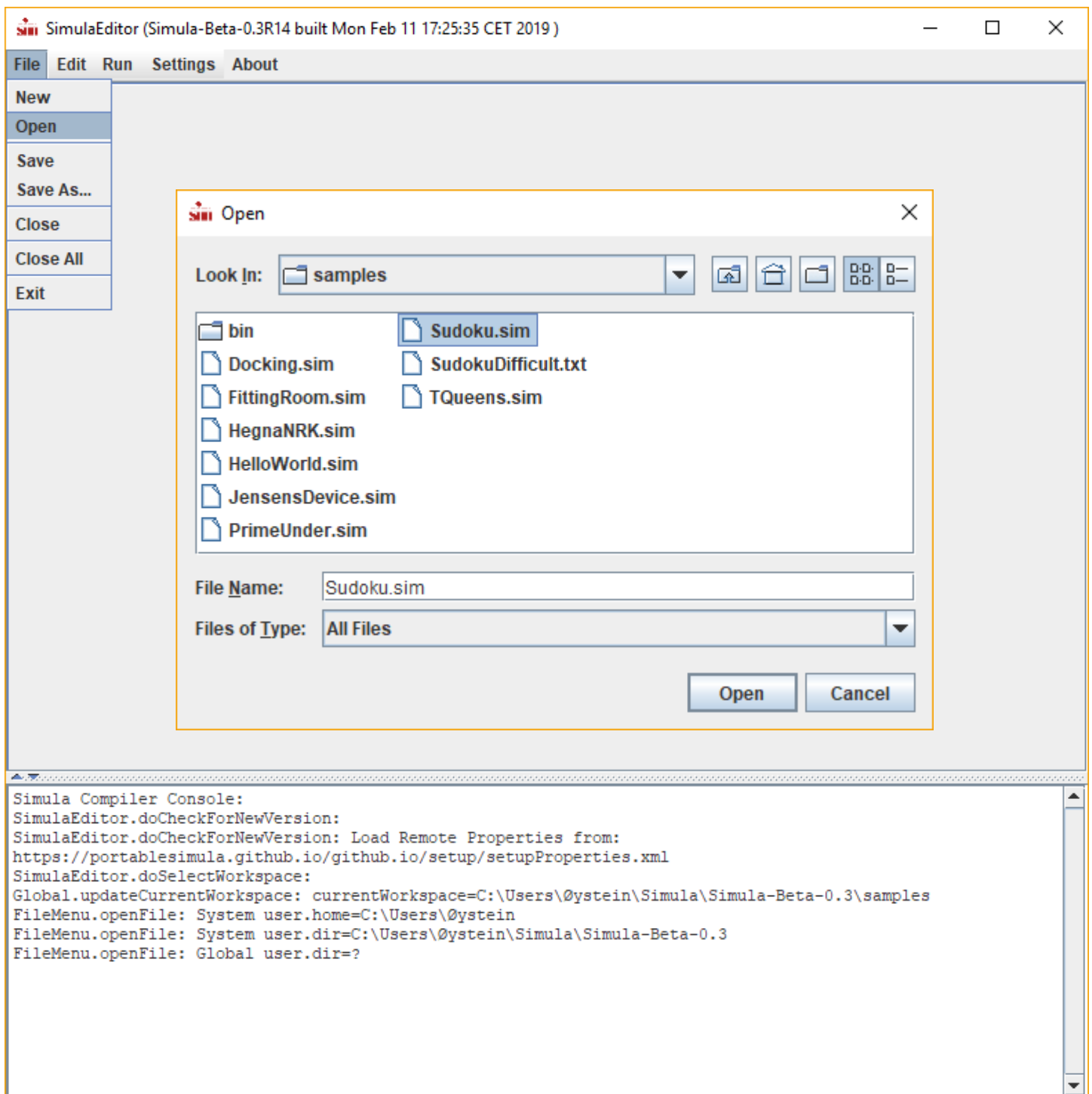
3.1 The Simula Editor



When the Simula Editor is started it will ask for a 'Workspace' to be used. A workspace is a directory for Simula Source files etc. The sub-directory `\bin` is used to store resulting `.jar` files.

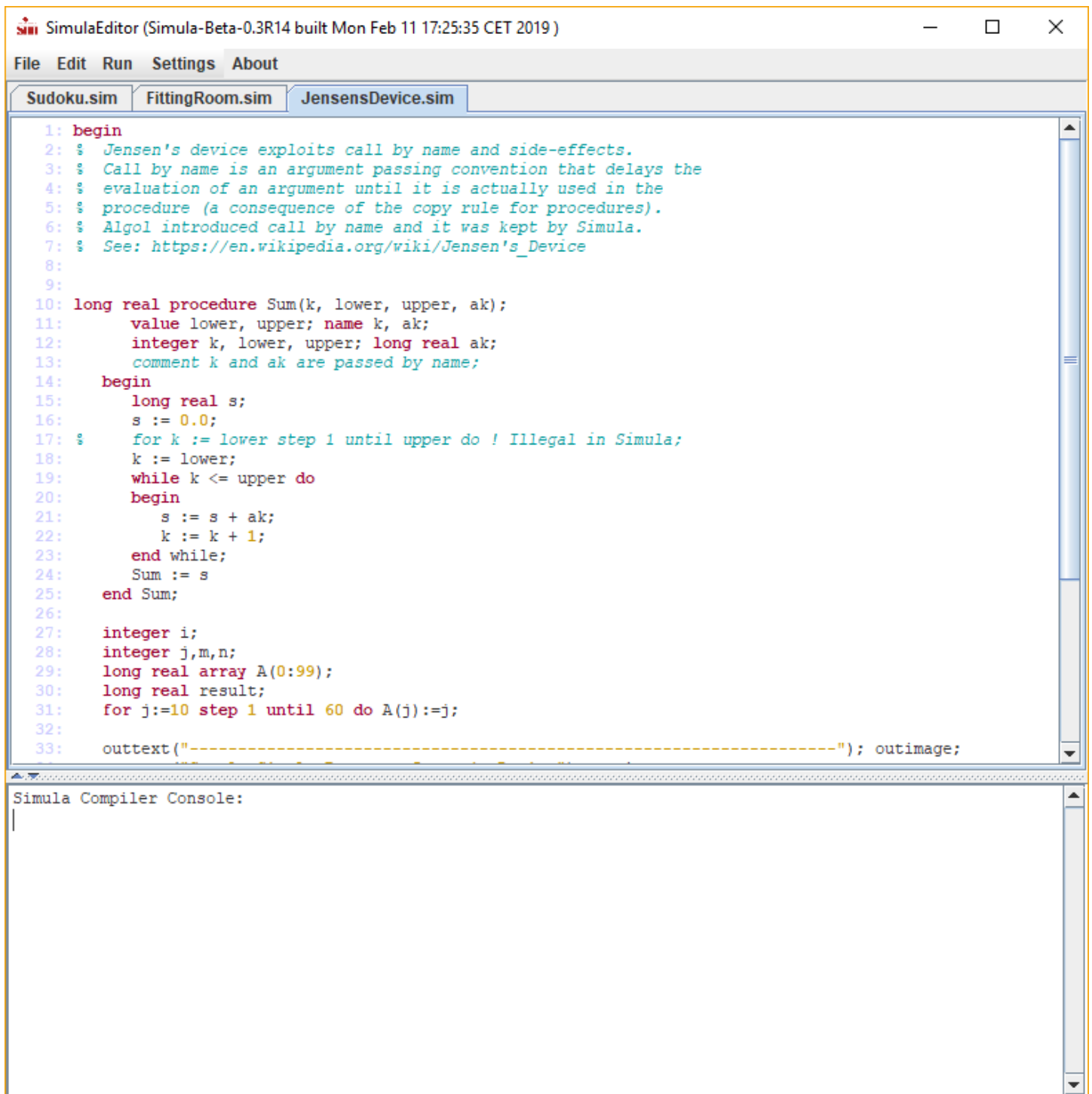
The default workspace `\samples` is created during setup and contains a set of sample Simula Programs. You may continue using this workspace for your own files or you may create a new one. This is done by pulling down and select the "Browse for another Workspace Directory".

Select [File] [Open] to open a sample Simula Source file.



You may ...

Syntax Sensitive Editing



The screenshot shows the SimulaEditor window with the title bar "SimulaEditor (Simula-Beta-0.3R14 built Mon Feb 11 17:25:35 CET 2019)". The menu bar includes "File", "Edit", "Run", "Settings", and "About". The tab bar shows three files: "Sudoku.sim", "FittingRoom.sim", and "JensensDevice.sim". The main editor area displays the following Simula code:

```
1: begin
2: % Jensen's device exploits call by name and side-effects.
3: % Call by name is an argument passing convention that delays the
4: % evaluation of an argument until it is actually used in the
5: % procedure (a consequence of the copy rule for procedures).
6: % Algol introduced call by name and it was kept by Simula.
7: % See: https://en.wikipedia.org/wiki/Jensen's\_Device
8:
9:
10: long real procedure Sum(k, lower, upper, ak);
11:     value lower, upper; name k, ak;
12:     integer k, lower, upper; long real ak;
13:     comment k and ak are passed by name;
14:     begin
15:         long real s;
16:         s := 0.0;
17: %     for k := lower step 1 until upper do ! Illegal in Simula;
18:         k := lower;
19:         while k <= upper do
20:             begin
21:                 s := s + ak;
22:                 k := k + 1;
23:             end while;
24:         Sum := s
25:     end Sum;
26:
27:     integer i;
28:     integer j,m,n;
29:     long real array A(0:99);
30:     long real result;
31:     for j:=10 step 1 until 60 do A(j):=j;
32:
33:     outtext("-----"); outimage;
```

Below the editor is the "Simula Compiler Console:" which is currently empty.

This is a preliminary Editor you should use with care.

To Compile and Run select [Run] [Run]

... enjoy

4. File Handling Conventions

4.1 File Naming

External file names or data set specifications follow the normal conventions for the OS under which the Simula System is running. Thus, a data set specification has the general form:

[path] file name [. extension]

where, the OS dependent name-separator character in the path may be '/' or '\' or any mix of them.

4.2 File Opening

When the attribute procedure "open" is called, the FILENAME of the file specifies the data set to be connected to the (Simula) file.

If a path is explicitly given in the FILENAME, the data set will be searched for in the specified directory only. If no path is given, the file is searched in user.dir

NOTE: The association between internal and external file is established at file open, not when the file object is created.

The search rules also applies for attribute elements.

If a data set named FILENAME is not found, then the action taken depends upon the type of file:

- For in(byte)files a popup file selector window is shown enabling the user to manually select another file. If the file selector is canceled the procedure "open" returns **false** and no further action is taken.
- For all other files, a new file FILENAME is created in the Current Workspace.

The identified data set is then opened, and "open" returns **true**.

The standard system files SYSIN and SYSOUT are treated in a special manner. These files are opened before the program proper starts execution (as described in chapter 10 of the Simula Standard), but may be closed and reopened in the Simula program. The data set association of these files can not, however, be changed from within an executing Simula program.

When using the runtime console, SYSIN and SYSOUT is connected to the console window. Otherwise; SYSIN is connected to Standard Input, SYSOUT to standard output.

4.3 File Initialization

When a data set created as a result of file "open", the underlying Java file stream is created. For a direct file unwritten images (bytes) below LASTLOC are zero.

4.4 File Closing

When a file is closed, the associated data set is released.

4.5 Access Mode Treatment

The standard attribute modes CREATE and PURGE are implemented.

The modes SHARED, APPEND, READWRITE, BYTESIZE and REWIND are not implemented in this release.

In addition, an extra access mode is available for image files to specify the character set used to encode/decode the file content. This mode has the form CHARSET:<charset name>

If the given charset name is illegal in the current Java virtual machine a runtime warning is given and the CHARSET mode remains unchanged.

4.6 File Locking

The procedure "lock" is implemented using Java's FileChannel.tryLock method.

The 'tryLock' method will attempt to acquire a lock on the given region of this channel's file.

This method does not block. An invocation always returns immediately, either having acquired a lock on the requested region or having failed to do so.

The procedure "unlock" simply release the underlying FileLock.

4.7 Checkpointing a File

The procedure "checkpoint" is implemented using Java's FileChannel.force method for direct access files and Java's flush method for all other file types.

5. Separate Compilation

5.1 Attribute Files

TBD

5.2 Attribute Information Elements

TBD

5.3 Compatible Recompilation

TBD

6. Additional Facilities

6.1 Language Extensions

S-Port Switch Statement

Extension inherited from S-Port Simula.

Switch-statement

= SWITCH (lowKey : hiKey) switchKey BEGIN { switch-case } [none-case] END

Switch-case

= WHEN <caseKey-list> do <statement> ;

None-case

= WHEN NONE do <statement> ;

<caseKey-list>

= caseKey { , caseKey }

CaseKey

= caseConstant | caseConstant : caseConstant

LowKey

= integer-or-character-expression

HiKey

= integer-or-character-expression

SwitchKey

= integer-or-character-expression

CaseConstant

= integer-or-character-constant

Translated into a Java Switch Statement with break after each <statement>

Occurs in Peter's S-Port compiler - but I haven't found a description of the semantics.

6.2 Additional Standard Procedures

- Procedure setSeed(seed); integer seed; ... ;
- Procedure waitSomeTime(millies); integer millies; ... ;
- Procedure printThreadList(withStackTrace); boolean withStackTrace; ... ;
- Text Procedure edit(x); <value-type> x; ... ;

where value-type ::= *integer* | *real* | *long real* | *boolean* | *character*

A suitable Text edit procedure is called behind the scene and the resulting item is trimmed and returned as result. I.e. leading and trailing blanks are removed

- Text Procedure edfix(x,n); <real-type> x; *integer* n; ... ;

where real-type ::= *real* | *long real*

The Text procedure putfix is called behind the scene and the resulting item is trimmed and returned as result. I.e. leading and trailing blanks are removed

The resulting numeric item is an INTEGER ITEM if n=0 or a DECIMAL ITEM with a FRACTION of n digits if n>0. It designates a number equal to the value of r or an approximation to the value of r, correctly rounded to n decimal places. If n<0, a run-time error is caused.

6.3 Additional Standard Classes

6.3.1 Additional Standard Class Drawing

The additional system class "Drawing" introduce basic graphical capabilities.
It has the class "simset" as prefix, so set-handling facilities are thus immediately available.

```
Simset class Drawing(title,width,height); Value title; Text title; Integer width,height;
begin
  ref(Head) Procedure renderingSet; ... ;
  Integer white,lightGray,gray,darkGray,black,red,pink,orange,yellow,green,magenta,cyan,blue;
  Integer Procedure color(r,g,b); Integer r,g,b; ...

  Procedure setBackgroundColor(color); Integer color; ... ;
  Procedure setDrawColor(color); Integer color; ... ;
  Procedure setFillColor(color); Integer color; ... ;
  Procedure setStroke(width); Real width; ... ;

  Procedure setFontStylePlain; ... ;
  Procedure setFontStyleBold; ... ;
  Procedure setFontStyleItalic; ... ;
  Procedure setFontStyleBoldItalic; ... ;

  Procedure setFontSize(size); Real size; ... ;
  Real Procedure getFontSize; ... ;

  Link class ShapeElement; ...
  Link class TextElement; ...

  ref(TextElement) Procedure drawText(t,x,y); Value t; Text t; ... ;

  ref(ShapeElement) Procedure drawLine(x1,y1,x2,y2); Long Real x1,y1,x2,y2; ... ;
  ref(ShapeElement) Procedure drawEllipse(x,y,width,height); Long Real x,y,width,height; ... ;

  ref(ShapeElement) Procedure drawRectangle(x,y,width,height);
  Long Real x,y,width,height; ... ;

  ref(ShapeElement) Procedure drawRoundRectangle(x,y,width,height,arcw,arch);
  Long Real x,y,width,height,arcw,arch; ... ;

  ref(ShapeElement) Procedure fillEllipse(x,y,width,height); Long Real x,y,width,height; ... ;

  ref(ShapeElement) Procedure fillRectangle(x,y,width,height); Long Real x,y,width,height; ... ;

  ref(ShapeElement) Procedure fillRoundRectangle(x,y,width,height,arcw,arch);
  Long Real x,y,width,height,arcw,arch; ... ;

  ...

end Drawing;
```

Additional Standard Class ShapeElement (local to Drawing)

```
Link class ShapeElement;
begin Integer COLOR,STROKE;
  ref(Shape) SHAPE; ! Java Class Shape is not visible in Simula ;

  Procedure setColor(rgb); Integer rgb; ... .. ;
  Procedure setStroke(size); Real size; ... .. ;

  Procedure drawLine(x1,y1,x2,y2); Long Real x1,y1,x2,y2; ... .. ;

  Procedure drawEllipse(x,y,width,height); Long Real x,y,width,height; ... .. ;

  Procedure drawRectangle(x,y,width,height); Long Real x,y,width,height; ... .. ;

  Procedure drawRoundRectangle(x,y,width,height,arcw,arch);
  Long Real x,y,width,height,arcw,arch; ... .. ;

  Procedure fillEllipse(x,y,width,height); Long Real x,y,width,height; ... .. ;

  Procedure fillRectangle(x,y,width,height); Long Real x,y,width,height; ... .. ;

  Procedure fillRoundRectangle(x,y,width,height,arcw,arch);
  Long Real x,y,width,height,arcw,arch; ... .. ;

  Procedure instantMoveTo(x,y); Long Real x,y; ... .. ;

  Procedure moveTo(x,y,speed); Long Real x,y,speed; ... .. ;

  Into( <rendering set> );
End;
```

The current color and stroke are maintained behind the scene and can be changed through the procedures 'setColor' and 'setStroke'.

Additional Standard Class TextElement (local to Drawing)

Link class TextElement(t,x,y); Value t; Text t; Long Real x,y;
begin

Procedure setColor(color); Integer color; ... ;
Procedure setStroke(width); Real width; ... ;

Procedure setFontStylePlain; ... ;
Procedure setFontStyleBold; ... ;
Procedure setFontStyleItalic; ... ;
Procedure setFontStyleBoldItalic; ... ;

Procedure setFontSize(size); Real size; ... ;
Real Procedure getFontSize; ... ;

Procedure setText(t); Value t; Text t; ... ;
Procedure moveTo(x,y,speed); Long Real x,y,speed; ... ;
Procedure instantMoveTo(x,y); Long Real x,y; ... ;

Into(<rendering set>);

end;

6.3.2 Additional Standard Class DEC_Lib

The additional system class "DEC_Lib" introduce a set of utility procedures from DEC Handbook III. The DEC Simula system was the very first Simula system created.

```
class DEC_Lib;
begin
  procedure abort(message);
  boolean procedure change(m,oldt,newt); name m; text m,oldt,newt;
  text procedure checkextension(fileName,defaulttextextension);
  integer procedure checkint(t); name t; text t;
  integer procedure checkfrac(t); name t; text t;
  long real procedure checkreal(t); name t; text t;
  text procedure compress(t,c); text t; character c;
  text procedure conc(t1,t2); text t1,t2;
  text procedure conc(t1,t2); text t1,t2;
  text procedure conc2(t1,t2); text t1,t2;
  text procedure conc3(t1,t2,t3); text t1,t2,t3;
  text procedure conc4(t1,t2,t3,t4); text t1,t2,t3,t4;
  text procedure conc5(t1,t2,t3,t4,t5); text t1,t2,t3,t4,t5;
  real procedure cptime;
  integer procedure dayno;
  text procedure daytime;
  procedure depchar(t,p,c); text t; integer p; character c;
  procedure enterdebug(maycontinue); boolean maycontinue;
  procedure exit(n); integer n;
  character procedure fetchchar(t,p); text t; integer p;
  character procedure findtrigger(m,tr); name m; text m,tr;
  text procedure from(t,p); text t; integer p;
  text procedure front(t); text t;
  boolean procedure frontcompare(string,config); text string,config;
  text procedure frontstrip(t); text t;
  text procedure getitem(tt); name tt; text tt;
  text procedure initem(fileref); ref(file) fileref;
  character procedure insinglechar;
  integer procedure linecount(pfil); ref(Printfile) pfil;
  character procedure lowc(c); character c;
  text procedure maketext(ch,n); character ch; integer n;
  boolean procedure puttext(oldt,newt); name oldt; text oldt,newt;
  text procedure rest(t); text t;
  character procedure scanchar(t); name t; text t;
  integer procedure scanfrac(t); name t; text t;
  integer procedure scanint(t); name t; text t;
  long real procedure scanreal(t); name t; text t;
  text procedure scantto(tt,c); name tt; text tt; character c;
  integer procedure search(t1,t2); text t1,t2;
  text procedure skip(tt,c); name tt; text tt; character c;
  integer procedure startpos(t); text t;
  text procedure today;
  text procedure tsub(t,p,l); text t; integer p,l;
  character procedure upc(c); character c;
  boolean procedure upcompare(master,test); text master,test;
  text procedure upto(t,p); text t; integer p;
end;
```

6.3.2.1 abort

procedure abort(message); **text** message;

Abort will print the message on the error device and then call terminate_program.

6.3.2.2 change

boolean procedure change(master,oldtext,newtext);
name master; **text** master,oldtext,newtext;

Starting at master.pos, the procedure change will search for the subtext oldt.

If not found, master remains unchanged and the procedure value is false.
The position indicator is set to length+1, i.e. master.setpos(0);

If found, oldt will be replaced by newtext and the procedure value is true.
If oldtext.length >= newtext.length then master will denote a subtext of the original master text, otherwise master will denote a new text object.
The position indicator is set to first character following newtext within the resulting master.

Changing all occurrences of oldtext to newtext may be done with the following procedure;

```
procedure changeAll(master,oldtext,newtext);  
name master; text masterr,oldtext,newtext;  
begin text local;  
    local:- master;  
    while local.more do change(local,oldt,newt);  
    master:- local  
end;
```

The position indicator 'pos' of master is set to first character after the substituted text. If not found, 'pos' is unaltered.

6.3.2.3 checkextension

text procedure checkextension(fileName,defaultextension);
text fileName,defaultextension;

The procedure checkextension may be used to add a default extension to file specifications not containing a dot ('.'). I.e.

```
fileName:- copy("myFile"); fileName:- checkextension(fileName,".txt");  
    will give fileName the value "myFile.txt", while  
fileName:- copy("myFile.doc"); fileName:- checkextension(fileName,".txt");  
    will leave fileName unaltered.
```

6.3.2.4 check numbers

integer procedure checkint(t); **name** t; **text** t;
integer procedure checkfrac(t); **name** t; **text** t;
long real procedure checkreal(t); **name** t; **text** t;

Each check number procedure analyses the text t from t.pos and on.
If a get<number> operation from this position is legal the returned value is +1.

If it would give an error - then
if the remaining text string is blank, the result is 0, else -1.

pos is placed after a legal item (+1),
after the first nonblank illegal character (-1)
or after the text if the rest is empty (0).

NOTE: The current implementation will place t.pos after a complete real item,
even if it overflows. Real underflow produces +0, overflow Infinity,
both of which are treated as legal results

6.3.2.5 compress

text procedure compress(t,c); **text** t; **character** c;

Removes all occurrences of the character c in the text t.

The procedure compress returns notext (if t contains no other characters
than a number of characters = c), or a reference to an initial subtext
of t (altered) which contains all characters of t not = c.
The part of t after this subtext is unchanged.

Example: t1:-copy("AxBxCxDx"); t2:-compress(t1,'x');
gives t1="ABCDxCxDx", t2==t1.sub(1,4), t2="ABCD"

6.3.2.6 Text concatenations

text procedure conc(t1,t2); **text** t1,t2;
text procedure conc2(t1,t2); **text** t1,t2;
text procedure conc3(t1,t2,t3); **text** t1,t2,t3;
text procedure conc4(t1,t2,t3,t4); **text** t1,t2,t3,t4;
text procedure conc5(t1,t2,t3,t4,t5); **text** t1,t2,t3,t4,t5;

Concatenate texts. Provided for compatibility only,
use the text concatenation operator & instead!

6.3.2.7 cptime

real procedure cptime;

Returns total CPU time spent since the beginning of the SIMULA program execution, expressed in seconds.

Provided for compatibility, use cputime instead!

6.3.2.8 dayno

integer procedure dayno;

Returns the day number in current month.

6.3.2.9 daytime

text procedure daytime;

Returns a reference to a new text of length 8 with contents: "hh:mm:ss"

where hh is hours
mm is minutes
ss is seconds.
at the time of the call.

6.3.2.10 depchar

procedure depchar(t,p,c); **text** t; **integer** p; **character** c;

Deposits the character c in the text t at position p.

If p is out of range, no action will be taken;

This is a safe version of S-port standard procedure StoreChar.

6.3.2.11 enterdebug

procedure enterdebug(maycontinue); **boolean** maycontinue;

Enterdebug will activate the debugging system (if present).

After a session, the execution will either continue (maycontinue true) or it will terminate (maycontinue false).

6.3.2.12 exit

procedure exit(n); **integer** n;

n=0: Terminate program immediately, with no further action.
No files closed.

n=1: Terminate program as if final end was reached, i.e. all
open files are closed (and a message given for each).

n=2: Equivalent to enterdebug(true).

Other values of n are reserved for extensions.

6.3.2.13 fetchchar

character procedure fetchchar(t,p); **text** t; **integer** p;

Returns the p'th character from t, if p is within range, otherwise '!0!'.

This is a safe version of S-port standard procedure LoadChar.

6.3.2.14 findtrigger

character procedure findtrigger(master,triggers);
name master; **text** master,triggers;

Starting from current master.pos, find first
occurrence of any of the characters in triggers.;

If a trigger character was found it will be returned and
master.pos will be placed after that character.

Otherwise when no trigger character was found '!0!' will be returned
and master.pos := master.length+1.

6.3.2.15 from

text procedure from(t,p); **text** t; **integer** p;

Returns a reference to the longest subtext of t starting at pos = p.

6.3.2.16 front

text procedure front(t); **text** t;

Returns a reference to the longest subtext of t before pos.

Invariant: front(t) & from(t,t.pos)) = t

6.3.2.17 frontcompare

boolean procedure frontcompare(string,config); **text** string,config;

Starting at current pos, does string begin with a substring equal to config ?

6.3.2.18 frontstrip

text procedure frontstrip(t); **text** t;

Returns a reference to the longest subtext of t starting with the first non-blank character.

6.3.2.19 getitem

text procedure getitem(tt); **name** tt; **text** tt;

Skips any blanks or tabs, starting at tt.pos. If tt.more holds then, an item is identified according to the following rules:

- a) If the first following character is a letter (a-z,A-Z), an identifier is found. The identifier consists of the initial letter and any following letters and/or decimal digits.
- b) If the first character is a digit, we have a numeric item, consisting of a string of digits with at most one decimal point "." included.
- c) Any other character except blank or tab forms an item on its own.

The value of getitem is a subtext reference to the item within tt, or notext if no item can be found starting at tt.pos. tt.pos will be placed after the item. When no item was found tt.pos=tt.length+1.

Example: "IF car.wheel_size > 13.5" will be split into the items "IF", "car", ".", "wheel", "_", "size", ">", "13.5" via successive calls to getitem.

6.3.2.20 hash

integer procedure hash(t,n); **text** t; **integer** n;

Procedure hash returns a calculated hash value, in the range (0:n-1), from a given text t. The result is taken modulo n before return.

It is recommended to choose n as a prime number.

6.3.2.21 initem

text procedure initem(fileref); **ref**(file) fileref;

The parameter is formally ref(file), but should refer an Infile, Directfile, Inbytefile or Directbytefile (otherwise notext is returned). Same as getitem for files.

On Infiles or Directfiles, lastitem is called first. Notext is returned, if (a) fileref is none, (b) fileref does not point to a legal file object (see above), or (c) the remainder of the file is spaces and/or TABs.

NOTE: Since no copying is done, a call on initem may destroy previously located items, if they are not copied by the program, since inimage may be invoked.

For bytefiles a single character item is returned.

6.3.2.22 insinglechar

character procedure insinglechar;

Returns next input character from sysin (after last inimage) without waiting for break character. A subsequent inimage will begin reading after the last character which has been input with insinglechar.

The use of this routine depends on runtime option USE_CONSOLE. If it is not set a runtime error will occur.

6.3.2.23 linecount

integer procedure linecount(pfil); **ref**(Printfile) pfil;

Return the current linesperpage setting for an open Printfile or -1 when pfil==none or -2 when pfil is not open.

6.3.2.24 lowc

character procedure lowc(c); **character** c;

Returns uppercase character as lowercase equivalent, other characters unchanged.

6.3.2.25 maketext

text procedure maketext(ch,n); **character** ch; **integer** n;

Return a text object of length n, filled with character ch.

6.3.2.26 puttext

boolean procedure puttext(oldstring,newstring);
name oldstring; **text** oldstring,newstring;

Puts the (short) text newstring into the (longer) text oldstring; newstring is put at pos in oldstring, and pos of oldstring is moved after the end of the new substring.

Returns false if there was not room for the new string, or if trying to modify a Constant text.

6.3.2.27 rest

text procedure rest(t); **text** t;

Returns a subtext reference of a text starting at pos.

6.3.2.28 scanchar

character procedure scanchar(t); **name** t; **text** t;

Safe version of getchar. Will not give runtime error, just returns char(0) if not t.more.

6.3.2.29 scanfrac

integer procedure scanfrac(t); **name** t; **text** t;

Scanfrac is similar to getfrac, but differs in two principal ways:

- 1) It looks for an item starting at t.pos
- 2) The handling of error conditions due to bad data is different.
Scanfrac returns the value of the next grouped item if any.
If no such item is found, 'minint' is returned.

t.pos will only be moved if de-editing was successful.

6.3.2.30 scanint

integer procedure scanint(t); **name** t; **text** t;

Scanint is similar to getint, but differs in two principal ways:

- 1) It looks for an item starting at t.pos
- 2) The handling of error conditions due to bad data is different.
Scanint returns the value of the next integer item if any.
If no such item is found, 'minint' is returned.

t.pos will only be moved if de-editing was successful.

6.3.2.31 scanreal

long real procedure scanreal(t); **name** t; **text** t;

Scanreal is similar to getreal, but differs in two principal ways:

- 1) It looks for an item starting at t.pos
- 2) The handling of error conditions due to bad data is different.
Scanreal returns the value of the next real item if any.
If no such item is found, 'minreal' is returned.

t.pos will only be moved if de-editing was successful.

6.3.2.32 scanto

text procedure scanto(tt,c); **name** tt; **text** tt; **character** c;

Scans from tt.pos until the next occurrence of the character c. Places tt.pos after the character found.

The function value of scanto is the text starting at the initial tt.pos and not including the character c found.

If no character c is found, tt.pos will be = tt.length+1, and scan denotes the rest of tt starting at the initial tt.pos.

6.3.2.33 search

integer procedure search(t1,t2); **text** t1,t2;

Search: find first subtext = t2 from t1.pos;

Return: The pos indicator of first character of subtext within t1.

6.3.2.34 skip

text procedure skip(tt,c); **name** tt; **text** tt; **character** c;

If, starting at tt.pos, a sequence of characters = c is found, then tt.pos is moved past these characters, otherwise tt.pos is unaltered.

The function value of skip is rest(tt) (after changing tt.pos).

6.3.2.35 startpos

integer procedure startpos(t); **text** t;

startpos returns the starting position of t within t.main.
Provided for compatibility only;

6.3.2.36 today

text procedure today;

Returns a reference to a text object of length 10 with contents: "yyyy-mm-nn" where yyyy is year, mm is month(in digits 01-12), nn is day (in digits 01-31).

This is the internationally standardized format for dates.

6.3.2.37 tsub

text procedure tsub(t,p,l); **text** t; **integer** p,l;

tsub is a safe variant of sub. Parameters which give an error in sub will return notext instead.

6.3.2.38 upc

character procedure upc(c); **character** c;

Returns lowercase character as uppercase equivalent, other characters unchanged.

6.3.2.39 upcompare

boolean procedure upcompare(master,test); **text** master,test;

Upcompare returns true if the contents of test is equal to the next test.length characters of master, counted from current master.pos.

The master characters will be converted to upper case before comparison (without changing the MASTER text).

Note that the TEST text will not be converted.
Thus

master	test	upcompare
BEGIN	BEG	true
BEGIN	beg	false
begin	BEG	true
begin	beg	false
xxxxx	BEG	false

assuming that master.pos = 1.

If test == notext the result will always be true.

6.3.2.40 upto

text procedure upto(t,p); **text** t; **integer** p;

Returns a reference to the longest subtext of t before pos = p.

6.3.3 Additional Standard Class CatchingErrors

The class CatchingErrors is used to implement runtime error catching. The basic idea is to enclose the 'inner' statement by a Java try-catch construction and, in the catch section to call a virtual procedure 'onError' to deal with the error situation.

To achieve this, the class CatchingErrors is hand-coded as a standard class.

```
class CatchingErrors;  
virtual: procedure onError;  
begin  
  -- code: try {  
    inner;  
  -- code: } catch(RuntimeException e) {  
  -- code:   onError$(e,onError);  
  -- code: }  
end;
```

Usage:

```
CatchingErrors begin  
  procedure onError(message); text message; begin  
    ... treating error  
  end;  
  ... any error here will cause calling onError  
end;
```

There are three ways to return from the procedure onError:

- 1) goto a non-local label.
- 2) calling terminate_program or error
- 3) falling through procedure end which has the same effect as ending the prefixed block.

Example: Safe version of getint returns maxint when it fails.

```
integer procedure safeGetint(t); text t; begin  
  CatchingErrors begin  
    procedure onError(message); text message; begin  
      safeGetint:=maxint;  
    end;  
    safeGetint:=t.getint;  
  end;  
end;
```

6.4 Directive Lines

The following compiler directives are defined according to the rules of Simula Standard sect.1.1.1:

If the first character of a source line is "%" (percent) the line as a whole is a directive line.

A directive line serves to communicate information to the compiler and consequently its meaning is entirely implementation-dependent, with the following single exception. If the second character is a space, the line has no significance; it may be used for annotation purposes. .

%BOUNDCHECK ON / OFF

Controls generation of array bound checking, if possible on the given implementation.

This S-PORT directive is ignored by this implementation.

%COPY file-name

Will cause the compiler to include the indicated file at this place in the source input file, as if the text was actually written in the module containing %COPY. COPY (or INSERT) may occur in the included file. The included lines are always counted, i.e. they will be numbered sequentially starting from the number of the line of the COPY-directive.

%DEFINITION

This directive is relevant only in a separately compiled procedure or class. The compiler will regard this compilation as a prototype definition only, i.e. the module is checked for syntactic and semantic correctness, and the appropriate attribute element is produced, but no code generation is performed.

This S-PORT directive is ignored by this implementation.

%EOF

When the compiler recognizes this directive it reacts as if the end of the source file was encountered, i.e. it terminates reading and, if this module was inserted (see INSERT / COPY), source scanning will continue after the INSERT (COPY) directive in the enclosing file.

This S-PORT directive is ignored by this implementation.

%INSERT file-name

Will cause the compiler to include the indicated file at this place in the source input file. INSERT may occur in the included file. In contrast to COPY, the included lines are not Counted (they will all be numbered with the line number of the line containing the outermost INSERT). Furthermore, if the source is being listed, listin is turned off before the inclusion and turned on again when reading continues after this directive.

This directive is treated as %COPY in this implementation.

%LIST ON / OFF
%NODUMMY ON / OFF
%NONECHECK ON / OFF
%NOSOURCE
%PAGE
%QUACHECK ON / OFF
%REUSE ON / OFF

These S-PORT directives are ignored by this implementation.

%SELECT string

Set or reset selectors for conditional compilation, see next section below.

%SETOPT string

Set or reset options and trace switches, see next section below.

%SETSWITCH name-string value-string

Will set compiler switch "name" to the value "value". The facility is intended for compiler maintenance, and is not explained further.

%SLENGTH last-column
%SOURCE
%SPORT ON / OFF
%TITLE page-heading

These S-PORT directives are ignored by this implementation.

%+string / %-string

Conditional compilation control, see next section below.

6.5 Conditional Compilation

This implementation makes it possible, on a line by line basis, to conditionally include or exclude source text from a compilation.

During input of the source text, the compiler maintains a set of boolean variables or selectors, corresponding to the letters of the (English) alphabet. Initially, these selectors are all reset. They may be set (reset) either by means of SELECT directives embedded in the source text or before compilation is activated by using compiler options.

Note: The case of the selector letters is significant.

If a source line contains "%+" or "%-" as its first two characters, the line is a conditional line. Such a line will be compiled conditionally, depending on the value of the selector(s) following immediately after. If no selector follows the line is treated as an illegal directive and ignored.

The SELECT Directive

The directive takes the form:

`%SELECT character-sequence`

First, all selectors are reset. Then, for each character in the sequence, the corresponding selector is set.

Conditional Lines

A conditional line takes the form:

`%selector-expression`

where " " represents the line to be conditionally included and the selector-expression has the form:

Selector-expression
= selector-group { selector-group }

Selector-group
= + letter_or_digit { letter_or_digit }
| - letter_or_digit { letter_or_digit }

i.e. a string of letters and signs, with the first character being a sign. The selector-expression is terminated by a SP.

Within each group, the values of the named selectors are tested. If the group is positive (prefixed by '+') and all tested selectors are set, or in the case of a negative group all tested selectors are reset, the group is deleted from the line, and the process is repeated for the next group (if any). If all selector groups have been deleted in this manner, the initial '%' and the terminating space is likewise deleted, and the resulting is treated as if it was read from source.

Note: The resulting line may well be a directive line.

If, on the other hand, a positive (negative) group contains a reset (set) selector, the line as whole is skipped.

Example:

<code>%+MD-f x:=x-1;</code>	the statement is compiled only when selectors 'M' and 'D' are set and 'f' is reset.
-----------------------------	--

7. Implementation Defined Aspects

In the Simula Standard, several issues of the language are left to be decided by the implementation. This section summarises all characteristics of the language which in some manner are dependent upon the particular implementation of the language, together with an indication of how the matter is treated in this implementation.

This section may be a useful guide when preparing programs which has been developed on other Simula Systems. Keeping these points in mind when preparing a program may spare you many a painful hour of debugging a program that behaves normally on some other Simula System, while it crashes on this system (or vice versa).

This chapter consists of four sections. Each of these contains a list of rules taken from the Simula Standard, each rule being followed by an explanation of decision taken for this implementation – marked THIS SYSTEM.

7.1 Language Extensions

This Simula System has added a 'Java type case statement' in the form of a switch-statement. See section 5.1

7.2 Allowed Implementation Restrictions

An implementation may restrict the value range of the 'isocode' construct, and the character set defined in table 1.1 in the Simula Standard, as long as the "basic" characters of the table are included.

THIS SYSTEM: Follows Java's character set; i.e. UNICODE internally and a OS-dependent character set when reading or writing files. The character set may be changed by the CHARSET access mode, see section 4.5.

An implementation may restrict the number of different block levels at which a system class may be used as a prefix.

THIS SYSTEM: No such restrictions exists.

An implementation may restrict, in any way, the use of "file" and its subclasses for prefixing or block prefixing.

THIS SYSTEM: No such restrictions exists.

The type **short integer** is allowed to be unsupported by an implementation in the sense that it must then be mapped onto **integer** (i.e. the key word **short** is ignored)

THIS SYSTEM: The type **short integer** is unsupported.

The type **long real** is allowed to be unsupported by an implementation in the sense that it must be mapped unto **real** (i.e. the key word **long** is ignored)

THIS SYSTEM: The type **long real** is fully implemented.

An implementation may restrict the use of the standard access modes and their possible standard values; in that case the procedure "setaccess" must treat such values as unrecognised, and return **false**.

THIS SYSTEM: No such restrictions exists.

An implementation may restrict the number of block levels at which an external class declaration may occur.

THIS SYSTEM: No such restrictions exists.

An implementation may restrict the possible values of access mode 'BYTESIZE' in any way.

THIS SYSTEM: The default byte size is 8 which can't be changed.

7.3 Implementation Dependent Characteristics

Whether or not the procedure "terminate_program" will close open external files (except those associated with 'sysin' and 'sysout'), is implementation defined.

THIS SYSTEM: All open files are closed (as if the appropriate "close" procedure was called).

The effect of a parameter to printfile.spacing with value zero, may be device and implementation dependent, if the standard effect of "overprint" cannot be achieved.

THIS SYSTEM: Printfile spacing is not implemented.

The interpretation of directive lines (apart from the " % space " convention) is implementation dependent.

THIS SYSTEM: Chapter 5.4 – 5.5 gives detailed information about which directives are supported.

The interpretation of "kind" and of the external identification string in an external procedure declaration is implementation dependent, as is the identification of a separately compiled module if no external identification is given.

THIS SYSTEM: See chapter 5. Separate Compilation.

It is implementation-dependent whether trailing blanks of image are actually transferred to the external file on outfile.outimage.

THIS SYSTEM: Trailing spaces are not recorded.

The size of the part of the file that is actually locked after a call to procedure "lock", is implementation dependent.

THIS SYSTEM: See description of procedure lock in section 4.6.

7.4 Implementation Defined Characteristics

The internal character set is implementation-defined. An implementation is required to document the translation between the internal character set and the standard character set (as defined by ISO 646).

THIS SYSTEM: Follows Java's character set; i.e. UNICODE internally.
See also section 4.5 access mode CHARSET to specify the character set used to encode/decode the file contents.

The values of "inlength" and "outlength" (see Simula Standard ch. 10) are implementation-defined.

THIS SYSTEM: Inlength=80 and outLength=132

The actual external files connected to SYSIN and SYSOUT are implementation-defined

THIS SYSTEM: When using the runtime console, SYSIN and SYSOUT is connected to the console window. Otherwise; SYSIN is connected to Standard Input, SYSOUT to standard output.

The relative value ranges of **real** and **long real** are implementation-defined.

THIS SYSTEM: The ranges are as defined by Java **float** and **double**.

Conversion from an integer type to a real type is exact within an implementation-defined range which includes zero. Conversion from **real** to long **real** is exact within an implementation-defined range which includes zero.

THIS SYSTEM: The ranges are as defined by the Java Language.

The range of a numeric item in a de-editing procedure is implementation-defined.

THIS SYSTEM: A runtime error occurs if the item exceeds the value range of the result.

The EXPONENT of the numeric item resulting from "putreal" has a fixed, implementation-defined number of characters.

THIS SYSTEM: Deviates from the Simula Standard since a **real** parameter will cause 4 characters to be used ('&', sign, two digits), and a **long real** will cause 6 characters to be used ('&&', sign, three digits).

The maximum length of a text frame is implementation-defined.

THIS SYSTEM: Text frames can hold Java's Integer.MAX_VALUE number of characters

The function values of "char" and "rank" are implementation defined.

THIS SYSTEM: According to UNICODE.

The exact definitions of the standard mathematical functions are implementation defined.

THIS SYSTEM: According to Java's Math.<func>

The association between a file object and an external file is implementation-defined.

THIS SYSTEM: The association is established at "open" and dissolved at "close".
See Chapter 4.

The effect of several file objects representing the same (external) file is implementation-defined.

THIS SYSTEM: The action taken depends upon the file access mode setting at "open".
See your OS manuals.

The details of procedures "open" and "close" are implementation-defined.

THIS SYSTEM: See chapter 4.

The interpretation of a function value of "lock" less than 1 is implementation-defined.

THIS SYSTEM: See description of procedure lock in section 4.6.

Outimage of outfile reacts in an implementation-defined way if the length of the internal image is incompatible with the format of the external file.

THIS SYSTEM: No incompatibilities exist.

Locate of directfile may invoke implementation-defined checks and possibly instructions to an external memory device.

THIS SYSTEM: No checks or I/O instructions are performed at "locate".

LINES_PER_PAGE of printfile: the value at object generation and after close is implementation-defined.

THIS SYSTEM: The default value is 66 lines.

The 'basic random drawing' algorithm is implementation-defined:

THIS SYSTEM: Random drawing is implemented using the recommended linear congruential sequence:

$$U(i+1) = \text{remainder} ((U(i) * 5^{**}(2*p+1)) // 2^{**}n)$$

with n=31 and p=6:

The effect of the conditions demanded for the parameters to "linear" not being fulfilled is implementation-defined.

THIS SYSTEM: If these conditions are not fulfilled, a runtime error occurs.

The number of decimals in the field for seconds of the function "datetime" is implementation-defined.

THIS SYSTEM: Three decimals are used.

The effect of the parameters to "histo" not fulfilling the condition: $\text{length of A} = \text{length of B} + 1$, is implementation-defined.

THIS SYSTEM: If this condition is not fulfilled, a runtime error occurs.

The default BYTESIZE for bytefiles is implementation-defined.

THIS SYSTEM: The default bytesize is 8 (bits).

Evaluation of arithmetic expression may give different results for different implementations.

THIS SYSTEM: Follows the rules of Java.