Portable Simula Revisited

Programmer's Reference

Working Draft

(as of xx februar 2019)

Table of Contents

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 dicrete 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:

```
5 Virtual labels and switches are not part of this
  implementation.

6 External non-SIMULA procedures are NOT IMPLEMENTED.

9 The name specified parameter to the random drawing procedures
  is not used.Instead, the default seed in the Java Library is
  used. This is not a real deviation since Simula Standard says
  BasicDrawing is implementation defined.

  The <real-type> parameter to the 'histd' 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 ot the language are left open to be decided by the actual implementation. Schapter 8. gives a complete list of these, together with a specification of the decision taken om 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 marked, 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.

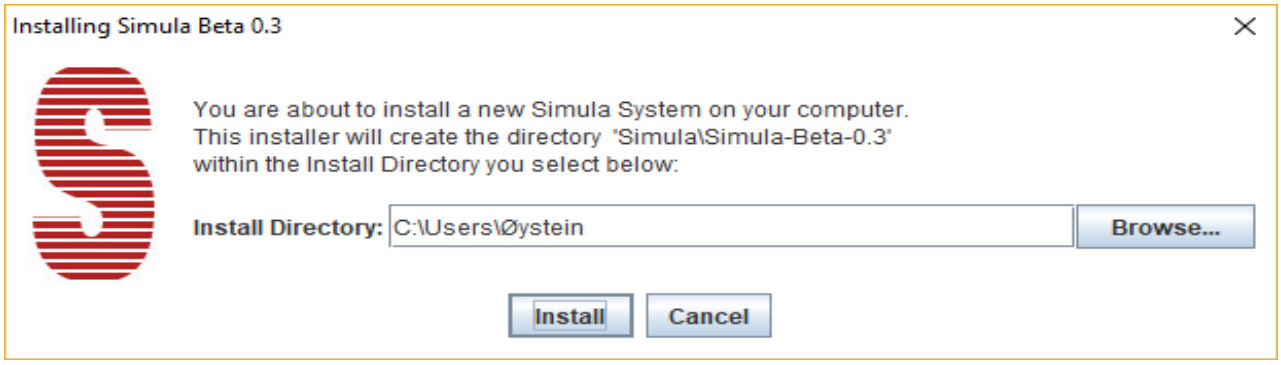
Just when your computer is about to download the setup file it probably gives a warning message. This message states that you are downloading from a unsafe source.

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 Compier

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 genereated.

You can move these script files wherever you want, for example, to the desktop. Alernatively, 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]
```

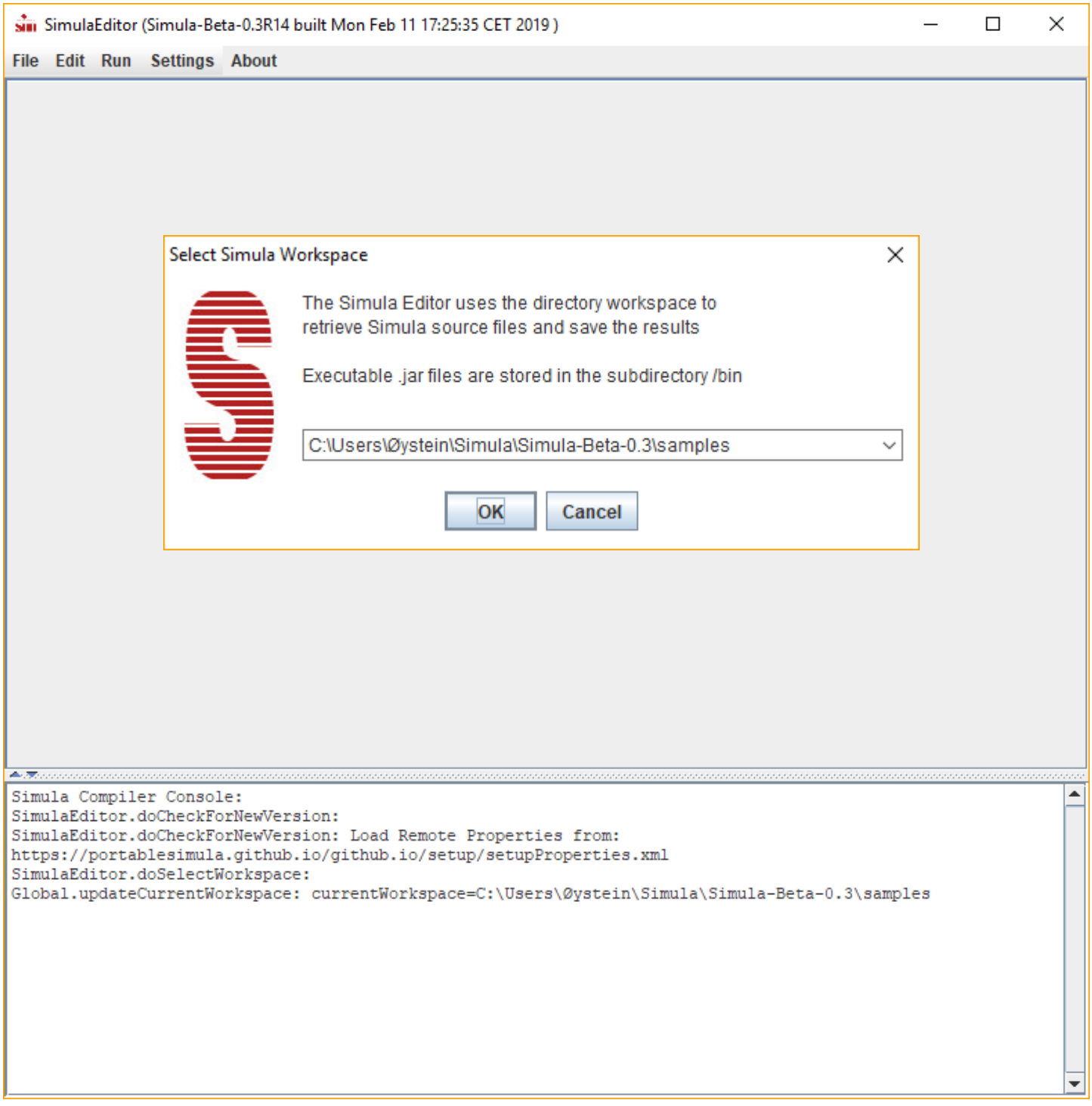
Java-options
: are described in the relevant Java Technical Dokumentation.

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

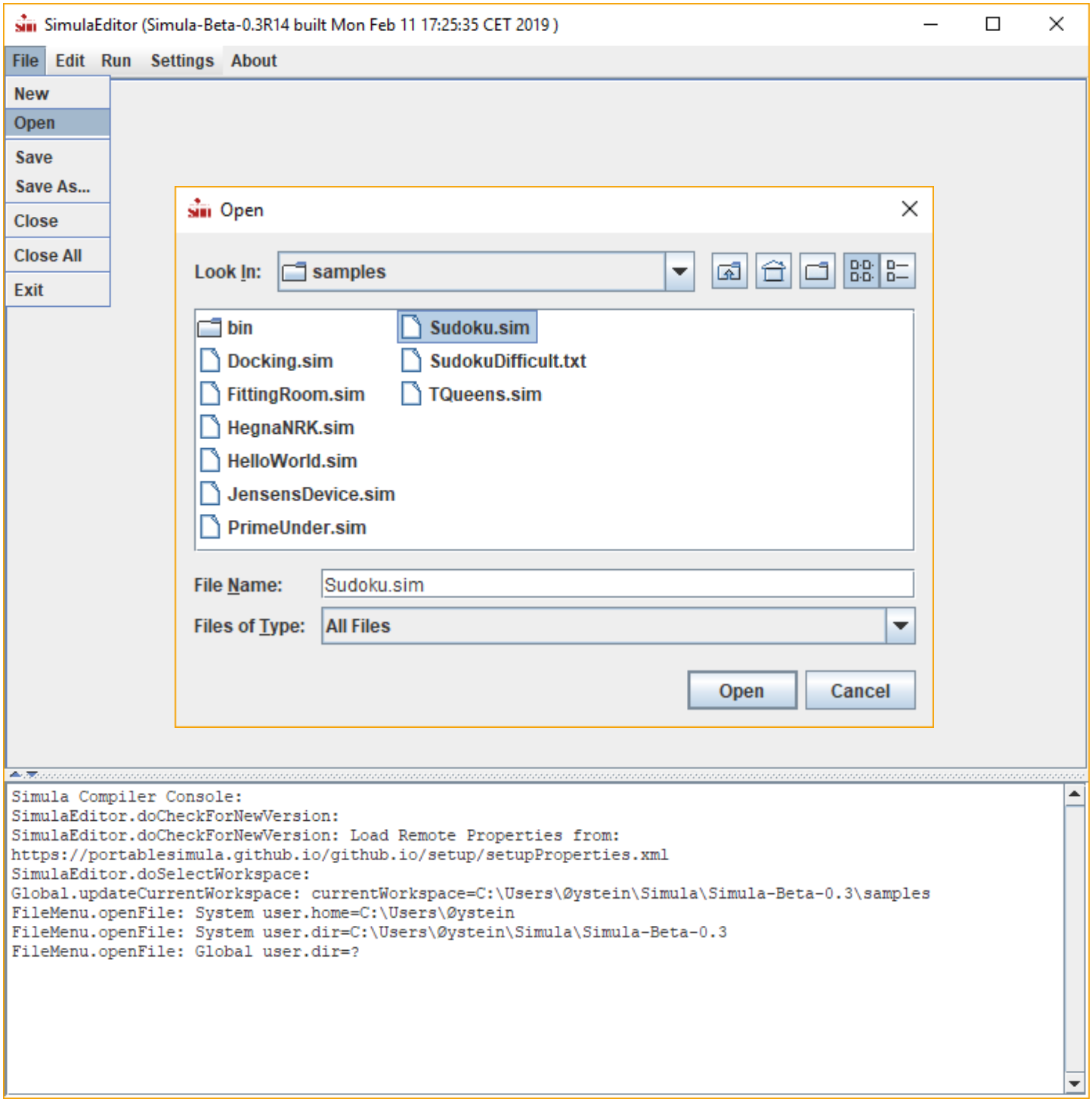
3.1 The Simula Editor



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.

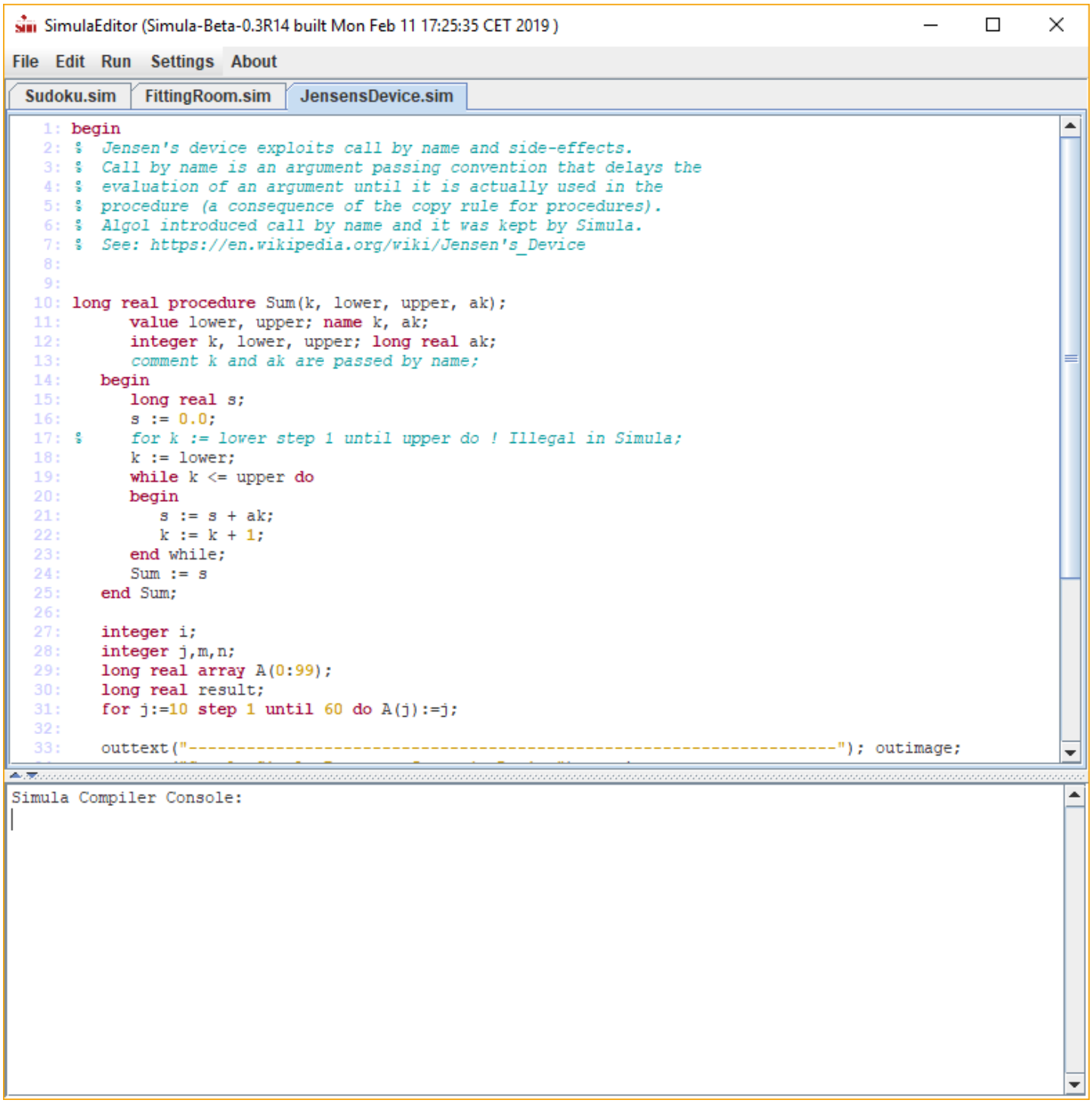
The default workspace \samples is created during setup and contains a set of sample Simula Programs. You may continue using this workspace for your own files or you may create a new one. This is done by pulling down and select the "Browse for another Workspace Directory".

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



You may ...

Syntax Sensitive Editing



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 following search rules apply:

- Search current directory
- Search Current Workspace.
- Search all the other Workspace's

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 Standad Input, SYSOUT to standard output.

4.3 File Initialization

When a data set created as a result of file "open", the underlaying 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 underlaying 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; ... ;

6.3 Additional Standard Classes

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; Long Real x1,y1,x2,y2; ... ;

    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;
```

6.4 Directive Lines

The following compiler directives are defined Acording to the rules of Simula Standard sect.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 occure in the included file. The included lines are always counted, i.e. they wil be numbered sequenctially 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 constrast 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 ot 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 { letter }
| - letter { letter }

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 I 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:

%+MD-f x:=x-1;	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. Keepig these points in mind when preparing a program may spare you many a painful hour of debugging a program that bahaves 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 unto **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 stadard 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 assosiated 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 diresctives 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 characterset is implementation-defined. An implementation is required to document the translation between the internal characterset and the standard characterset (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 Standad 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 Java's Math.random method. The name specified parameter to the random drawing procedures is not used. Instead, the default seed in the Java Library is used. Also, see chapter 5.2 procedure setSeed.

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: length of A = 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 atrithmetic expression may give different results for different implementations.

THIS SYSTEM: Follows the rules of Java.