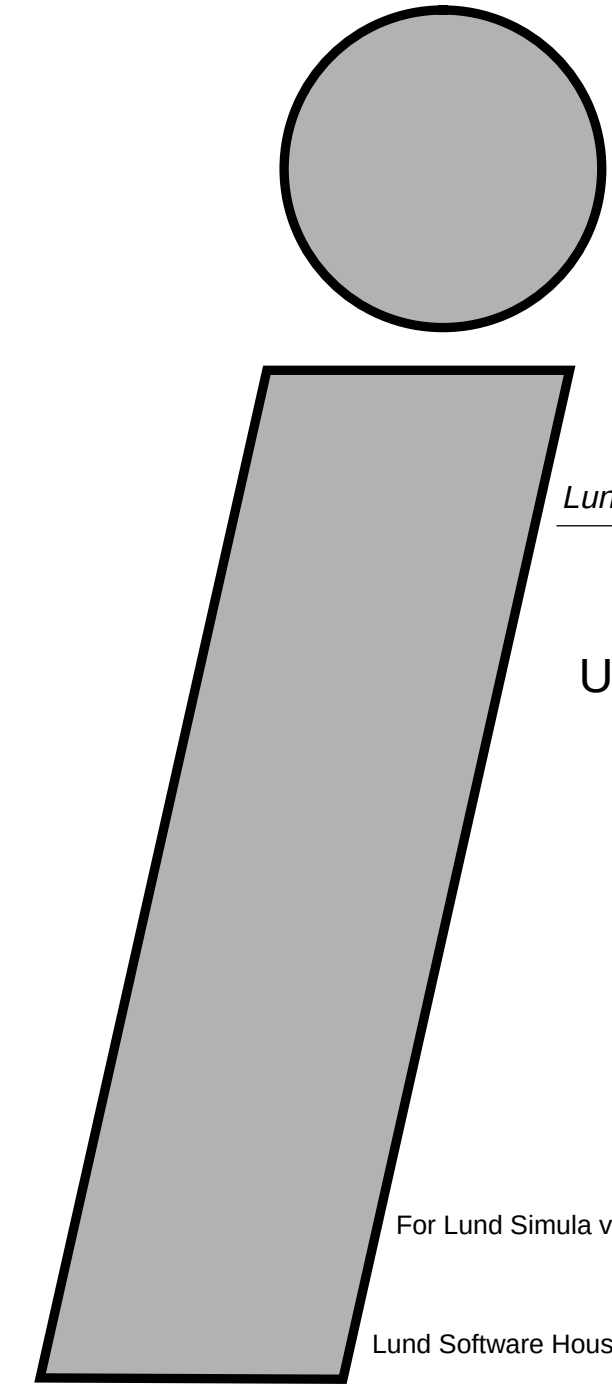


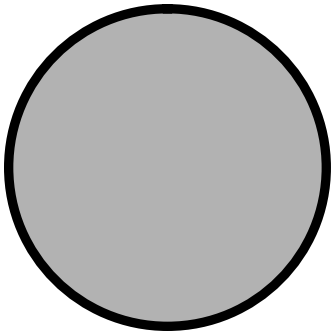
LUND

Lund Simula Documentation

Using the **libsim** library on Unix Systems

For Lund Simula version 4.15 or later

Lund Software House AB, Sweden



SIMULA

Lund Simula Documentation

Using the libsim libraray on Unix Systems
Version 4.15
by Boris Magnusson

Printed at: 5 December 1995 12:45 pm

Lund Software House AB
P.O.Box 7056
S-220 07 Lund, Sweden

Table of Contents

1 Introduction

SIMLIB is a collection of mainly small utility routines in SIMULA collected over the years at QZ and FOA in Stockholm, Sweden. Simula contains (compared to other languages) a fairly extensive library as part of its standard, but LIBSIM still offers many useful additions. LIBSIM was first offered with the Simula implementation for DEC-10, but have since then spread to also most other implementations. Although not part of the standard, LIBSIM can be regarded as a 'standard' library, being generaly available.

In the current version we have destilled the essential, portable parts of the original library. The parts that have been removed (or not yet included) are:

- Routines that are now part of the Simula standard library.
- DEC-10 specific utilities
- Terminal specific IO packages (SafeIO, Vista, DAHelp, SQHelp).

Routines
excluded from this version of SIMLIB are summarized in section 4.

2 Organisation and use

The routines are distributed as separately compiled Simula procedures and classes. The files are normally installed in '/usr/local/simulabin/libsim' (the -.atr files) and the object are reached trhrough the link: '/usr/local/simulabin/lib/ liblib-sim.a'.

Declaration in a Simula program:

external text procedure frontstrip;

Compilation:

% **simcomp <program> -L=/usr/local/simulabin -l=libsim**

(or just: **simcomp <program> -l**)

Linking:

% **simld <program> -llibsim**

(or just: **%simld <program> -l)**

3 Overview of functionallity

This part of the documentation describes the available routines and the back-ground needed to understand and use them. The description is kept short and intened to give an overview when reading through. It should be useful to get a summary of the routines available, and to select routines for potential use. When actually using a routine a fully detailed description is needed concerning parameters and results, please consult the Interface description in section 5.

3.1 Text utilities

This category of operations are operations that could have been attributes of text. They are writable in Simula, but convenient to have. In many cases the Pos indicator of the text passed as parameter is important (this is where the action is started, and it is updated as part of the result).

Front, Rest, UpTo, From can be described as combinations of parameters to 'Sub', introducing convenient names as outlined in figure 1.

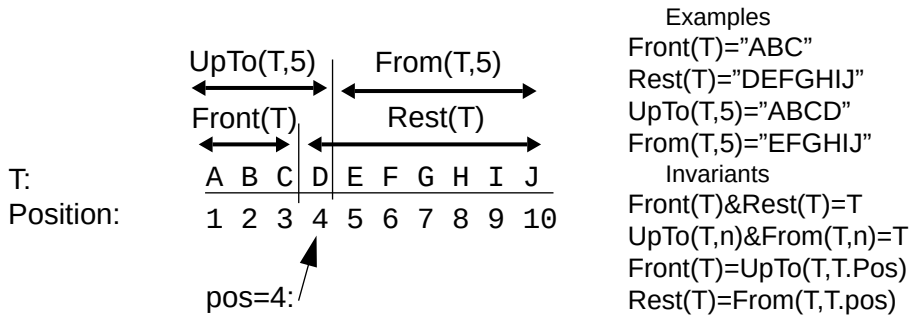


Figure 1 *Result of operations: Front, Rest, UpTo and From in an example*

TSub is equal to Sub except that it returns Notext when Sub returns an error.

FrontStrip is similar to Strip, but strips leading spaces and tabs (starts at 1).

ScanChar returns is similar to GetChar, except it returns Null at end of text.

ScanTo locates a character (starting from pos). Returns the subtext scanned over and updates pos to past the character.

Skip will scan past several occurances of a character, update pos to past the sequence and return 'Rest'.

Search locates a String in a (longer) Master text.

FindTrigger similar to ScanTo, but searches for any of a set of characters.

FrontCompare returns true if Rest of a text starts with a given string.

Change locates and replaces a string in a text (expanded when needed).

FetChar is similar to GetChar, but uses a parameter rather than pos.

DepChar is similar to PutChar, but uses a parameter rather than pos.

LowC returns lower case equivalent of a letter.

UpC returns upper case equivalent of a letter.

MakeText returns a new text with all characters equal to the parameter.

Compress returns a text where all occurances of a given character are removed.

3.2 Editing and de-editing of numbers, get-, put-, and in- routines

The following operations are offering facilities for safe reading of input numbers. There are also some operations that duplicate functionality of standard routines, but uses the current value of Pos of a text where the standard routines uses 1.

CheckReal returns true if GetReal can be called without runtime error.

CheckInt returns true if GetInt can be called without runtime error.

ScanReal similar to GetReal, but starts at pos and will not fail on bad data.

ScanInt similar to GetInt, but starts at pos and will not fail on bad data.

GetRadix similar to GetInt but with specific radix (not just 10) and starts at 1).

Radix edits a number into a text in a specific radix. Allocates a text string.

PutIntAtPos[1] similar to PutInt, but starts at pos rather than 1.

1. Added in Lund version of SIMLIB, not part of the original QZ set.

PutRealAtPos[1] similar to PutReal, but starts at pos rather that 1.

PutText analogus to the above (and PutChar), but deposit a string (at pos).

GetItem is similar to Getint etc. but returns a text containing the next item (i.e identifier, number or delimiter). It uses pos rather than start at 1.

InItem performs NextItem and GetItem on an Infile or Directfile image.

SimId locates the next item in a text and returns it if it is a Simula Identifer (including '_').

Today =DateTime.sub(1,10); result is todays date in form: 'yyyy-mm-dd'.

DayTime =DateTime.sub(11,8) result is time of day in format: 'hh:mm:ss'.

3.3 Support for Swedish national characters

The Swedish languages uses three letters (AA, AE, AO and aa, ae, oe) that are not present in English. In 7-bit character representations these six charcters are replacing six characters () which thus can not be used. The following routines are taking this convention into account and are useful instead of their English couterparts (when dealing with Swedish text). For simplictity they are named in Swedish. Note that these routines are useful only when Swedish characters in 7-bit representation is used. For 8-bit representations there is yet no widely used standard.

Bokstav - replaces standard routine Letter.

LitenBokstav - replaces standard routine LowCase.

StorBokstav - repalces standard routine UpCase.

JfrTkn - compares characters according to the alfabetic order in Swedish, ignoring case.

Meny - replaces Libsim routine Menu.

TagOrd - replaces Libsim routine GetItem.

InOrd - replaces Libsim routine InItem.

3.4 Support for character based, terminal I/O

The following operations offering some utilities for character based terminal I/O. They are compatible with and use sysin/sysout, but provides some additional functionality.

OutLine outputs a long string on several lines, chopped into image size pieces.

BreakOutLine as OutLine but no CR/LF at end of lines.

LookAhead locates next non-space charcter in a In- or DirectFile.

InLine issues a prompt and returns users (stripped) response.

Request like InLine, but validates the input.

3.5 Mathematical utilities

ILog = Entier(Log10(Abs(x)))+1 , number of digits in integer part.

ILog2I =ln(x)/ln(2), power of 2 for x, rounded to integer.

ISum, RSum, LSum returns sum of integer, real, long real array, respectively.

Sigma2 calculates Mean, Variance amd Sum of values in a real array.

SigMean calculates Mean and Variance of a statistcal material, iteratively needing only one observation at the time.

Scramble unsort, ramdomly permutes, the values in an integer array

PerGen class generating succesive permutations of an integer array.

3.6 Sorting and searching

There are stright forward routines for searching for characters and strings in texts (ScanTo, Skip, Search, FindTrigger) in section 3.1. Here we describe some more elaborate routines.

SortIA,SortRA,SortLA,SortTA sort an array in Ascedning (increasing) order.

SortID,SortRD,SortLD,SortTD sort an array in Descending order.

Variants: I=Integer, R=Real, L=Long Real, T=Text array

Hash calculate a hash-value based on a text.

Menu matches a key with a set of strings, allowing abreviation.

Lookup locates a string in a sorted text array using binary search.

FSrch, StSrch, StSrchB used for fast search of a pattern in many texts.

FastSearch[1] a class for text searching, equivalent to FSrch, but nicer packaged.

4 Routines excluded from Lund LIBSIM

4.1 Now directly available in Standard Simula library.

ClockTime - now part of standard library, just delete external declaration.

CPUTime - now part of standard library

MaxInt, MaxReal - now part of standard library (as is MinInt, MinReal, MinLongReal, MaxLongReal and MaxRank).

UpCase - now part of standard library

LowCase - now part of standrad library

4.2 Available in Standard Simula after minimal changes[2].

IMax, RMax LMax - change to standard routine Max

IMin, RMin LMin - change to standard routine Min

TxPtr - not needed any more since introduction of text constants. Replace TxPtr(T) with just T.

Conc2, Conc3, Conc4, Conc5 - replace Conc2(T1,T2) with T1 & T2 etc.

StartPos - now avilabale in Text, replace StartPos(T) with T.Start

InIsOpen, OutIsOpen, PrintIsOpen, DirectIsOpen - change to F.IsOpen

Exit - change to Terminate_Program

ScratchFile - Use sequence: ... F.SetAccess("purge"); F.Close;

ForceOut - replace with F.CheckPoint

4.3 Dependent on DEC-10 / VMS operating system.

These should be covered by Unix calls (or they are not yet understood ...):

OutChr - should go into Unix library?

OutString - should go into Unix Library?

---

1. Added in Lund version of SIMLIB, not part of the original QZ set.

2. For compatibility with existing programs, also these routines are avialable as external Simula procedures. They are, however, not meaningful to use in new programs.

GetCh - should go into Unix Library?
FindInfile, FindOutFile, FindDirectFile - Use FileUtil.Exist
Sleep - should go into Unix library (sleep)
CheckExtentsions - see simlib library: FileNamUtil
Echo - should go into Unix
TmpIn - See UnixCmdUitl.GetEnv
TmpOut - ?

5 Detailed Interfaces

5.1 Text Utilities

Front
text procedure Front(T); text T; ! Pos used, unchaged;
Returns a reference to the longest subtext of T before Pos.

Rest
text procedure Rest(T); text T; ! Pos used, unchaged;
Returns a subtext reference of a text starting at Pos.

UpTo
text procedure UpTo(T,I);
text T; ! pos not used, unchanged;
integer I; ! used as Pos;
Returns a reference to the longest subtext of T before Pos=I.

From
text procedure From(T,P);
text T; ! pos not used, unchanged;
integer P; ! used as Pos ;
Returns a reference to the longest subtext of T starting at Pos = P.

TSub
text procedure TSub(T,P,L);
text T; ! pos not used;
integer P,L; ! used for Pos, Length of sub-string;
Similar to attribute Sub. In cases where T.Sub(P,L) would have caused a run time error, TSub returns Notext.

FrontStrip
text procedure FrontStrip(T);
text T; ! Pos not used or changed, starts at 1. ;
Returns a reference to the longest sub-text of T starting with the first non-blank (space or tab) character.

ScanChar
character procedure ScanChar(T);
name T; text T; ! Pos used and incremented ;
Similar to GetChar, but returns Null when there is no more character in T.

ScanTo
text procedure ScanTo(T,C);
name T; text T; ! Pos used and updated;
character C; ! character to look for;
Starting from T.Pos, looks for the next occurrence of the character C. T.Pos will be placed after the character found and the subtext from previous Pos up to but not including the found character will be returned as result If no C is found, the rest of the text is returned. Example: for T="X,YY,ZZZ", ScanTo(T,',') will return the three strings: "X", "YY", "ZZZ" and then notext.

Skip
text procedure Skip(T,C);
name T; text T; ! Pos used and updated ;
character C; ! At return T.GetChar <> C;
Starting from T.Pos it finds the first character different from C and moves T.Pos to that character. Skip returns Rest(T). If the character at T.Pos is different from C, T.Pos will not be changed.

Search

```
integer procedure Search(Master, LookFor);
text Master; ! Pos used, but not changed;
text LookFor; ! string searched for ;
```

Locates the string 'LookFor' in 'Master'. Starts at Master.Pos and returns Master.Pos for 1st character of the first substring equal to LookFor (=Master.Sub(Search(Master,LookFor), LookFor.Length)) If no such subtext exists, then the result is Master.Length+1. Note that Master.Pos is not changed. In order to find succesive occurances of LookFor, the calling program must change Master.Pos by some other mean.

FindTrigger

```
character procedure FindTrigger(Master,Triggers);
name Master; text Master; ! text searched, from pos, and on (pos updated);
text Triggers; ! set of characters looked for. ;
```

Starting from current Master.pos, finds first occurrence of any of the characters in Triggers. The result is the character found and Master.Pos is set past the returned character. If no Trigger is found, the returned value is Null, and Master.Pos is set after the string.

FrontCompare

```
Boolean procedure FrontCompare(String, Beginning);
text String; ! text searched, from pos and on, unchanged;
text Beginning; ! string matching with;
```

Answeres: starting at current pos, does String begin with a substring equal to Beginning?

Change

```
Boolean procedure Change(Master,OldT,NewT);
name Master; text Master; ! Pos valid and updated, may denote new text;
text OldT,NewT; ! replace 'OldT' with 'NewT';
```

Change the subtext OldT in Master to NewT. Searches from Master.pos and on (see Search) for OldT, when found Change returns True and:
- if NewT.Length=OldT.Length then the string is simply replaced in Master,
- if NewT.Length<OldT.Length then the string is replaced and the rest of Master is compacted so Master will denote a subtext of the original Master text,
- if NewT.Length>OldT.Length then Master will be assigned a new (longer) string to hold the result.

In each case Master.pos is set to past the replaced string. if OldT is NOT found, Chnage returns False and, Master.pos is set to its length+1. Example: changing all occurrences of OldT to NewT can be done with the following procedure:

```
  procedure ChangeAll(Master,OldT,NewT); name Master; text Master,OldT,NewT;
 begin  text Local;
    Local:- master;
    while local.More do
      change(Local,oldt,newt);
    master:- Local
 end -- ChangeAll --;
```

FetChar

```
character procedure FetChar(T,P);
text T; ! pos not used or changed;
integer P; ! used as Pos in T.;
```

Returns the P:th character from T (or NULL if P is out of range).

DepChar

```
procedure DepChar(T,P,C);
text T; ! pos not used or changed, but content changed;
integer P; ! used for Pos in T;
character C; ! value stored in T.;
```

Deposits the character C in the text T at position P. If P is out of range, no action will be taken.

LowC

character procedure LowC(C);
character C; ! value to convert;

Returns lower case equivalent of letter, other characters unchanged.

UpC

character procedure UpC(C);
character C; ! value to convert ;

Returns lower case character as upper case equivalent, other characters unchanged.

MakeText

text procedure MakeText(C,N);
character C; ! character to fill with;
integer n; ! length of retruned, new text;

Returns a new text of length N filled with character C.

Compress

text procedure Compress(T,C);
text T; ! text altered, pos not used;
character C; ! character to remove;

Returns a subtext of T with only characters different than C. This first part of T is thus altered, but the part of T after this subtext is unchanged. Compress returns Notext if T contains no other characters than a number of characters = C.

Example: t1:-Copy("AxBxCxDx"); t2:-compress(t1,'x');
returns: t1="ABCDCxDx", t2==t1.Sub(1,4)="ABCD".

Today

text procedure Today;

Returns todays date in text form: "yyyy-mm-dd" =DateTime.sub(1,10)

DayTime

text procedure DayTime;

Returns todays date in text form: "hh:mm:ss" =DateTime.sub(11,8)

5.2 Editing and deediting of numbers: get, put and inroutines

CheckReal

integer procedure CheckReal(T);
name T; text T; ! Pos used and updated. ;

Considering the text from T.pos and on, if a GetReal operation 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, otherwise -1.
Pos is placed after a legal item (+1), after the first non-blank illegal character (-1) or after the text if the rest is empty (0).

CheckInt

integer procedure CheckInt(T);
name T; text T; ! Pos used and updated. ;

Considering the text from T.pos and on, if a GetInt operation 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, otherwise -1.
Pos is placed after a legal item (+1), after the first non-blank illegal character (-1) or after the text if the rest is empty (0).

ScanReal
long real procedure ScanReal(T);
name T; text T; ! Pos is used and updated. ;

Scanreal is similar to GetReal, but the handling of error conditions due to bad data is different. ScanReal returns the value of the next real item in the text T. T.Pos will only be moved if de-editing was successful. On Failure it returns min-longreal (= 'minus infinity').

ScanInt
integer procedure ScanInt(T);
name T; text T; ! Pos is used and updated. ;

Scanreal is similar to GetInt, but the handling of error conditions due to bad data is different. ScanInt returns the value of the next integer item in the text T. T.Pos will only be moved if de-editing was successful. On Failure it returns minint (= 'minus infinity').

GetRadix
integer procedure GetRadix(Base,T);
integer Base; ! Base used for representation of ..;
text T; ! ... the textual representation of an integer. Pos not used.;

As GetInt, converts a text T containing digits (i.e. an integer item) to an integer, but uses a selectable radix Base. If Base = 16 then the characters 0123456789ABCDEF may be used, otherwise, part of the sequence 0123456789(10)(11)(12)... may be used. Note that texts containing illegal characters will always be (somehow) interpreted.

Radix
text procedure Radix(Base,I);
integer Base; ! radix base to use, 8, 10, 16 are popular;
integer I; ! integer to represent in 'Base'. ;

Returns a text containing the representation of I in radix Base. Base may be negative but not -1 or zero. The resulting text will never contain blanks. If Base = 16 then the sequence 0123456789ABCDEF will be used, otherwise, part of the sequence 0123456789(10)(11)(12)... will be used.

PutIntAtPos
boolean procedure PutIntAtPos(T,I);
name T; text T; ! Pos used and updated, text changed. ;
integer I; ! Integer to convert to text. ;

Formats the integer I and deposits the result in the text T, left adjusted, starting at Pos. T.pos is increased past the stored digits. If T is not long enough, False is returned and pos unchaged.

PutRealAtPos
boolean procedure PutRealAtPos(T,L,N);
name T; text T; ! Pos used and updated, text changed. ;
long real L; ! Real to convert to text. ;
integer N; ! Number of digits in result. ;

Formats the real/long real L and deposits the result in the text T, left adjusted, starting at Pos. T.pos is increased past the stored digits. N is the digits used (see PutReal). If T is not long enough, False is returned and T.Pos is unchaged.

PutText
boolean procedure PutText(T, String);
name T; text T; ! Pos used and updated, text changed. ;
text String; ! characters to insert. ;

Puts the short word String into the long text T, starting at Pos. T.Pos is increased to after the stored characters. Returns False if there was not enough room in T.

GetItem
text procedure GetItem(T);
name T; text T; ! Pos is used and updated. Result is sub-text of T;

First any blanks or tabs after T.Pos in the text are skipped. Then the procedure reads an item. By an item is meant either an identifier (a letter followed by letters, digits) or a number (a series of digits which may contain one dot) or any other character except blank. The result is a reference to a subtext (not a copy) of T, or Notext if there are only blanks left or pos=length+1.
Example: T="IF CAR.WHEEL_SIZE > 13.5", GetItem will return: IF/CAR/./WHEEL/_/SIZE/>/13.5 and then Notext. Notes:

1. The position of the parameter starts from current pos.
2. Preceding blanks or tabs (if any) are skipped.
3. The resulting position indicator setting is that following the last character of the matched word;

GetSimItem

text procedure GetSimItem(T);
name T; text T; ! Pos is used and updated. Result is sub-text of T;

Simliar to getItem but accepts also underline in Identifiers.
First any blanks or tabs after T.Pos in the text are skipped. Then the procedure reads an item. By an item is meant either an identifier (a letter followed by letters, digits, '_') or a number (a series of digits which may contain one dot) or any other character except blank. The result is a reference to a subtext (not a copy) of T, or Notext if there are only blanks left or pos=length+1.
Example: T="IF CAR.WHEEL_SIZE > 13.5", GetSimItem will return: IF/CAR/./WHEEL_SIZE/>/13.5 and then Notext. Notes:

1. The position of the parameter starts from current pos.
2. Preceding blanks or tabs (if any) are skipped.
3. The resulting position indicator setting is that following the last character of the matched word;

InItem

text procedure InItem(FileRef);
ref(File) FileRef; ! ref to open Infile (or Directfile) object. ;

Same as getItem, but for Infiles and DirectFiles. LastItem is called first and then the result of GetItem(Image) is returned. Notext is returned if only blanks or tabs are left in the file or if Endfile is TRUE. Note that since the Item:s are not copied, a call on InItem may cause Inimage and thus overwrite of previously located items (in Image), if they have not been copied by the program.

SimId

text procedure SimId(T);
name T; text T; ! Pos is used and updated, returns sub-string of T.;

Locates and returns the subtext of the next SIMULA identifier in T. Starting at T.Pos it skips any blanks or tabs. If T.More still holds, an identifier is found if the next character is a valid identifier starter according to the Simula syntax. The value of SimId is a subtext reference to the identifier found within T (or Notext if no identifier was found). T.Pos is updated to past the identifier.

5.3 Support for Swedish national characters

Bokstav

Boolean procedure Bokstav(C);
character C; ! Character to test ;

Returns True for 7-bit ISO-characters correspondig to Letter and in additon it recognize upper and lower case of the three Swedish national characters. These replace the square brackets ([]), curley brackets ({}), back-slash () and pipe (|) in English ISO tabel. These six character codes are thus considered letters by Bokstav.

LitenBokstav

text procedure LitenBokstav(T);
text T; ! text to convert, Content changed but Pos unchanged.;

Convert a string of characters to all lower case as LowCase does. In addition it also converts the three Swedish national characters, represented in 7 bit ASCII, ([] -> {}| resp.).

StorBokstav

text procedure StorBokstav(T);
text T; ! text to convert, Content changed but Pos unchanged.;

Convert a string of characters to all upper case as UpCase does. In addition it also converts the three Swedish national characters, represented in 7 bit ASCII, ({}| -> [] resp.).

JfrTkn

integer procedure JfrTkn (c1, c2);
character c1, c2; ! charcters to compare ;

Returns a three-way result (+1,0,-1) depending on how the two characters C1 and C2 compare. The result is calculated taking the three Swedish characters into account (using 7-bit ISO representation). JfrTkn considers upper and lower case equal. The result is defined as:

-1 if C1<C2
0 if C1=C2
+1 if C1>C2

JfrTkn is supposed to be useful when sorting (Swedish) characters in alfabetical order.

Meny

Boolean procedure Meny(T,I,Tabell,N);
name I; integer I; ! Result index when found ;
text T; ! String to look for ;
text array Tabell; ! Table of valid menu commands ;
integer N; ! Length of the array Tabell (1:N) ;

This is a Swedish version of Menu, considering Swedish national characters. Meny is designed to be used for validity checks in menu-like command requests. When it returns true, I is returning the index in Tabell which matches the key T. For more details see Meny.

TagOrd

text procedure TagOrd(T);
name T; text T; ! Pos is used and updated. Result is sub-text of T;

This is a Swedish version of GetItem. The difference is that it allows also Swedish national characters (represented in 7-bit ISO) in an 'identifier'. Otherwise see GetItem for details

InOrd

text procedure InOrd(FileRef);
ref(File) FileRef; ! ref to open Infile (or Directfile) object. ;

This is a Swedish version of InItem, using TagOrd rather than GetItem. InOrd skips spaces and tabs, possibly calling inimage to locate and return an Item (identifer, number, special character). See TagOrd and InItem respectively for details.

5.4 Support for character based, terminal I/O

OutLine

procedure OutLine(T,Out_File);
text T; ! pos not used or chaged ;
ref (OutFile) Out_File; ! file to write on, e.g. SysOut;

Output the argument T as one or more lines of output on Out_File. Protected against errors when the parameter text is longer than the image, T is chopped into image size pices and output one by one on a line each.

BreakoutLine

```
procedure BreakoutLine (T, Out_File);
text T;  ! pos unused and unchaged ;
ref (OutFile) Out_File; ! file to write on, e.g. SysOut;
```

Similar to OutLine but does not output line-breaks (no CR/LF) after each line. It uses BreakOutimage, rather than Outimage as OutLine uses.

InLine

```
text procedure InLine(Question,In_File);
text Question; ! Prompt to print, pos not used;
ref(infile) In_File; ! File to read from. ;
```

Types 'Question' as a prompt and returns users response as a stripped sub-text of Sysin.Image. If In_file is None, SysIn is used for input (Sysout is always used).

LookAhead

```
character procedure LookAhead (fileref);
ref (File) fileref; ! Infile or Directfile object;
```

Locates the next non-white (space or tab) character. Calls LastItem to scan over white space. Returns the next non-white character (if any) but leaves pos so InChar will return the same charcter. When at end-of-file it returns Space!

Request

```
text procedure Request(Prompt,Default,Result,Valid,ErrMsg,Help);
name Prompt,Default,ErrMsg,Help, ! In parameters /may change;
Result, ! Out parameter;
Valid;  ! Jensens device, should match expression using 'Result';
text Prompt; ! Prompt text e.g. "X-value" ;
text Default;! Default value such as "2.5" ;
boolean Valid;! Should evaluate to 'True' if current value of Result is OK;
text Result; ! Result, user response assigned here;
text ErrMsg; ! Error text e.g. "Legal values are 1..10";
text Help;   ! Help text, e.g. "Enter valid horizontal coordinate" ;
```

Outputs a prompt to the user and returns a Validated reply. The Prompt and the Default value is displayed to the user. Users response is assigned to Result and then Valid (Boolean, Call-back, expression) is checked. If it evaluates to True, Request returns otherwise the user is show the error message ErrMsg and prompted for a better value. If the user enter a '?', the Help text will be displayed. If the user just hits Return, the Default is returned as Result. Note that all parameter are called by name so any of them can be matched with a expression or text procedure evaluating to different values each time they are called, as more and more verbose help texts.

5.5 Mathematical utilities

ILog

```
integer procedure ILog(X);
long real X;
```

Useful when editing numbers into texts. Returns Log10(Abs(X))+1, i.e. integer digits in X (excluding sign), if Abs(X)>=1, 0 if X=0 and number of leading zero decimals, if Abs(X)<1

ILog2i

```
integer procedure ILog2i (x);
long real x;
```

Returns ln(x)/ln(2), power of 2 for x, rounded to integer.

ISum

```
integer procedure ISum(IA,N);
integer array IA;
integer N; ! Size of array IA (1:N) ;
```

Return Sum of integer array IA (1:N).

RSum

```
long real procedure RSum(RA,N);
real array RA;
integer N; ! Size of array RA (1:N) ;
```

Return Sum of real array RA (1:N).

LSum

```
long real procedure LSum(LA,N);
long real array LA;
integer N; ! Size of array LA (1:N) ;
```

Return Sum of long real array LA (1:N).

Sigma2

```
real procedure Sigma2(Mean,Sum,RA,N);
name Mean,Sum; ! - output parameters;
real Mean; ! - mean value (Sigma2 returns Variance);
real Sum;  ! - sum of elements RA(1:N) ;
real array RA; integer N; ! input paramters, RA(1:N) ;
```

SIGMA2 calculates the variance, mean value and sum of a real array RA, from RA(1) to RA(N).

SigMean

```
procedure SigMean(Sigma2,Mean,K,XK1);
name Sigma2,Mean,K; ! In/Out parameters ;
long real Sigma2; ! In: previous variance, Out: updated taking Xk1 into account;
long real Mean;   ! In: previous mean,     Out: updated taking Xk1 into account;
integer K;        ! In: Number of observations, Out: incremented;
real XK1;         ! Input: The new observation of X, observation K+1;
```

Calculates Mean-value and Variance for a statistical material in an iterative fashion. Given mean and variance for k operations and the new observation k+1, mean and variance is updated. Example of use:

```
  begin
    external procedure SigMean;
    long real SigmaAccum,MeanAccum;integer Obs; ! Updated by SigMean;
    real X; ! the just found observation. ;
    long real StandDev; ! Calculated when all observations known;
    Obs:= 0; mean:= 0; sigma2:= 0; ! Initialization ;
    inimage; ! One value per input line;
    while enfile do
    begin
      X:=inreal; ! The new observation, X(Obs+1) ;
      SigMean(SigmaAccum,MeanAccum,Obs,x);
        ! Updates SigmaAccum,MeanAccum and Obs;
    end while;
    StandDev:= Sqrt(SigmaAccum*Obs/(Obs-1));
    Outtext("Mean, Variance, Std.dev. Observations:"); Outimage;
    Outreal(MeanAccum,5,20); Outreal(SigmaAccum,5,20);
    Outreal(StandDev,5,20); Outint(Obs,10); Outimage;
  end;
```

Note: compared to Sigma2, SigMean may lose some precision. The advantage of using SigMean over Sigma2 is that only one observation is need at the time rather than all observations (possibly many) needed by Sigma2.

Scramble
procedure Scramble(IntArray,Bottom,Top,U);
name U; ! Integer variable to be updated;
integer array IntArray; integer Bottom,Top;!IntArray(Bottom:Top) shuffled;
integer U; ! Random seed, as for other random drawing procedures. Init to odd no.;

Scramble performs a Random permutation of elements in the integer array IntArray in the interval (Bottom:Top).

class PerGen
class PerGen(A,N);
integer array A; integer N; ! Array to permute A(1:N);

Class PerGen generates all the possible permutations of the contents in integer array a(1:N), one at a time. PerGen objects works as iterators, 'Call' the object to generate next permutation in a. The permutations will be generated in such manner that the last elements of A will change most slowly. Example:

```
begin
  external class PerGen;
  integer array a(1:3);
  ref (PerGen) pg;
  a(1):=1; a(2):=2; a(3):=3;
  pg:- new PerGen(a,n);
  while pg.Cycles < 2 do ! consider first cycle only ;
  begin
    outint(a(1),3); outint(a(2),3); outint(a(3),3); outimage;
    Call(pg);  ! generate new permutation;
  end;
end - Result (one cycle of 6 permutations):
1  2  3 /  2  1  3 /  3  1  2 /  1  3  2 /  2  3  1 /  3  2  1
```

Note that permutations are generated systematically and that the order will be REVERSED when one N! cycle is completed. Thus for N = 2 then result will be (if A(1)=1, A(2)=2 initially): (1 2) (2 1) / (2 1) (1 2) / (1 2) (2 1) etc. (2 permutations per cycle).

Operations

Cycles
integer procedure Cycles;

Return the current permutation cycle number, 1,2,... The number is increased after N! permutations have been generated (including the inital one). Cycle is returning 2 when A contains the first permutation of the second cycle (which is equal to the last permutation of the first cycle, since the order of permutations are reversed with each cycle).

5.6 Sorting and searching

SortIA
procedure SortIA (Arr,N); integer array Arr; integer N;! Sort Arr(1:N). ;

Sort the Integer array Arr in Ascending order (Arr(i+1)>=Arr(i)). N is the size of A. Note that you can call: SortIA(Arr, UpBound(Arr,1)).

SortID
procedure SortID (Arr,N); integer array Arr; integer N;! Sort Ar(1:N). ;

Sort the Integer array Arr in Descending order (Arr(i+1)<=Arr(i)). See SortIA for details.

SortIA
procedure SortRA (Arr,N);real array Arr; integer N;! Sort Arr(1:N). ;

Sort the Real array Arr in Ascending order. See SortIA for details.

SortRD
procedure SortRD (Arr,N);real array Arr; integer N;! Sort Arr(1:N). ;

Sort the Real array Arr in Descending order. See SortIA for details.

SortLA
procedure SortLA (Arr,N);long real array Arr; integer N;! Sort Arr(1:N). ;
Sort the Long array Arr in Ascending order. See SortIA for details.

SortLD
procedure SortLD (Arr,N);long real array Arr; integer N;! Sort Arr(1:N). ;
Sort the Long array Arr in Descending order. See SortIA for details.

SortTA
procedure SortTA (Arr,N);text array Arr; integer N;! Sort Arr(1:N). ;
Sort the Text array Arr in Ascending order. See SortIA for details.

SortTD
procedure SortTD (Arr,N);text array Arr; integer N;! Sort Arr(1:N). ;
Sort the Text array Arr in Descending order. See SortIA for details.

Hash
integer procedure Hash(T,N);
text T; ! String to hash, pos not used, unchanged;
integer N; ! Result returned is modulus N.;
Returns a hash coded value of T in the interval (0:N-1). It is recommended to choose N as a prime number.

Menu
Boolean procedure Menu(T,I,Table,N);
name I; integer I; ! Result index when found (or 0, -1);
text T; ! String to look for ;
text array Table; ! Table of valid menu commands ;
integer N; ! Length of the array Table (1:N) ;
Identifies a command in a set of commands. It check for nonambiguous correspondence between T and an element from the text array Table. (Note that an exact match will always be accepted, even if it is a substring of another table element.) The table must contain upper case letters only with no trailing blanks. The input may have lower case letters. If a match is found Menu returns True and Index is the matching table entry. If no match is found Menu returns False then I will return 0. If T is ambiguous, I returns -1. Example:

```
begin text array Menutable(1:5);
  integer Index;  Boolean OK;  text T;
  Menutable(1):- Copy("STOP");  Menutable(2):- Copy("START");
  Menutable(3):- Copy("END");   Menutable(4):- Copy("ENDURE");
  Menutable(5):- Copy("EXIT");
  T:- <... input ...>;
  OK:= Menu(T,Index,Menutable,5);
```

OK will become True if T is equal to:
START,start,Sta (Index=2), sto (Index=1), enD (Index=3),
endure,endu (Index=4)
OK becomes False for T equal to:
ST,e,en (Index=-1), x (Index=0)

Lookup
Boolean procedure Lookup(T,TA,Low,High,I);
name I; integer I; ! The resulting index;
text T; ! Text-Key to look for;
text array TA; integer Low,High; ! Sorted array TA(Low:High) to look in;
Perform a binary search for T in the sorted text array TA(Low:High).
- If the text, T, is found in TA Lookup will return True, and the matching index I will be returned (for which TA(I)=T)
- If the text is Not found Lookup will return False and I will indicate where T should have been in TA. (I will be in the range (Low,High-1) in case TA(I)<T<TA(I+1) or else I = High if T>TA(High) or I = Low-1 if T<TA(Low)). TA

must be sorted in Ascending order, i.e. TA(j)<TA(j+1). (Use procedure SortTA for sorting if necessary.)

FSrch

integer Procedure FSrch (String, SData);
name String;
text String; ! String to search;
text SData; ! data as returned by a call to StSrcB/StSrcH ;

Performs a fast search in String for a pattern. The result is returned as the Pos in String of the first occrance of the Pattern. Before calling FSrch the pattern to search for must be initialized by a call to StSrcB (or StSrch). The result (SData) is used with succsive calls to FSrch to search for the pattern in various Strings.
Results, value returned by FSrch:
n (n is a non-negative integer) A match is found, n characters long String.sub(String.Pos,n) is the string found.
-1 No match
-2 String too long.

StSrcB

integer procedure StSrcB (Pattern, SData);
name Sdata; ! result parameter (automatically alloacted);
text Pattern; ! In: Patter specyfing what to search for.;
text SData; ! Out: Intermediate result used as param to FSrch. ;

StSrcB must be called before FSearch can be called. StSrcB analyzes Pattern (the patern so search for) and forms an intermedate representation in SData. The Pattern is in its simplest form the sequence of characters to search for (where upper and lower case letters match), but more complicated search crtieria can be specifed with control sequences in the following syntax:
?C Matches any single character.
?R Matches the beginning or end of "string".
?S Matches any character which is not a letter, digit or national character. Also matches the beginning or the end of a line.
?T Matches one or more spaces or horizontal tabs.
?D Matches any digit.
?A Matches any letter or national character.
?E (=Exact), do not match upper case character against lower case and vice versa.
?? Representation of '?' in patterns.
Example I: To find the next line beginning with "abcdef", search for "?Rabcdef".
Example II: To find the word "man" but not when it appears as part of a longer word like "command", search for "?Sman?S".
Example III: To find what is probably the number of a year, the two digits 19, followed by two more digits, followed by something which is neither a letter nor a digit, search for "19?D?D?S".
Resulting codes:
0 Initialization OK
-1 '?T' next to SPACE, HT or '?T'
-2 Empty pattern
-3 Illegal letter after '?'
-4 '?R' inside pattern
-5 Pattern too long
-6 SDATA too small.

StSrch

integer procedure StSrch (Pattern, SData);
name SData; ! result parameter (automatically alloacted);
text Pattern; ! In: Patter specyfing what to search for.;
text SData; ! Out: Intermediate result used as param to FSrch. ;

StSrch is a variant of StSrcB that orignally accepted only short Patterns. In the current implementation the gain is neglectable and StSrch simply calls StsrcB. Kept for compatibility only. See StSrcB for detailed documentation

6 Index to classes and procedures