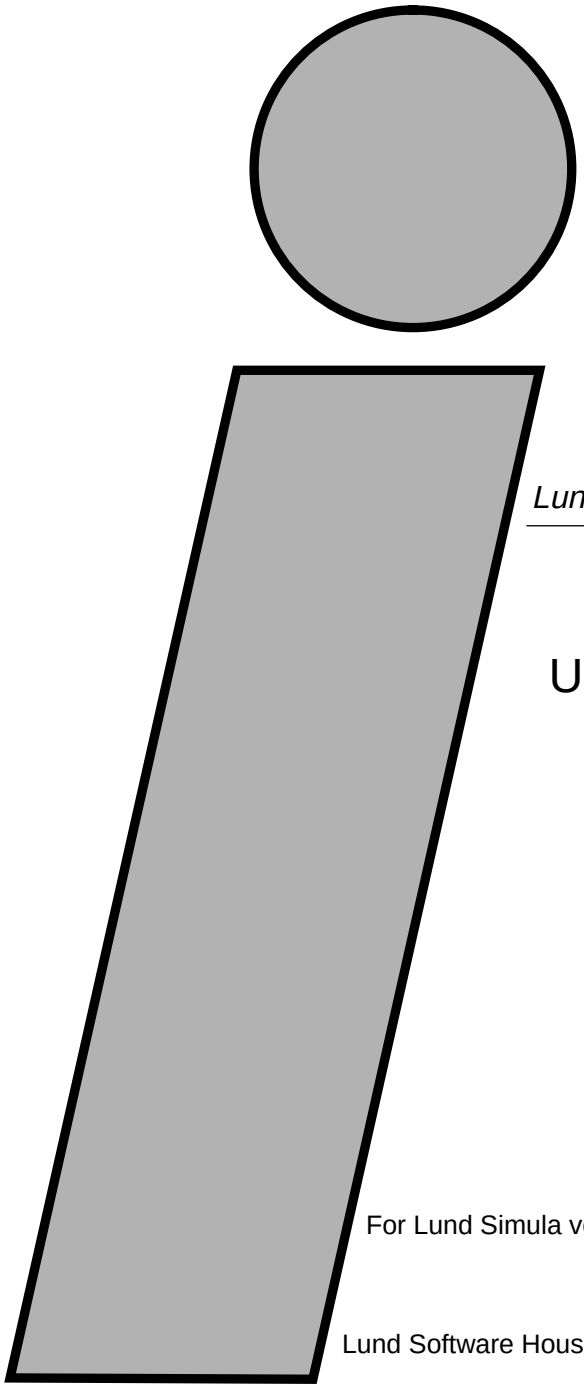


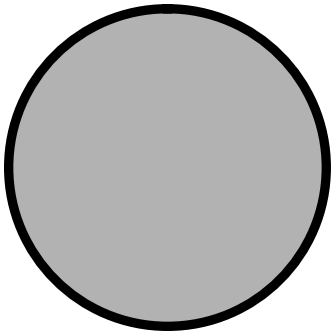
LUND

Lund Simula Documentation

Using the **simlib** library on Unix Systems

For Lund Simula version 4.15 or later

Lund Software House AB, Sweden



SIMULA

Lund Simula Documentation

Using the SimLib library on Unix Systems
Version 4.15

by Boris Magnusson

Printed at: 5 December 1995 12:23 pm

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

Table of Contents

1 Introduction

Simlib is a library with externally compiled classes and procedures. The routines contain various usefull routines that can not easily be written in Simula such as bitmanipulation and other assembly level routines, and also some operating system interface routines. The first group of routines are portable across Lund Simula implementations while the second group are portable across Unix implementations only.

Portable classes

BitFiddleClass - Bitwise Boolean operations on Integers
BitPackClass - routines to pack Numbers in binary form
BitSetClass - Operations on Set of Numbers
CallDebugger - Enter Simula debugger from program
CmdLineClass - contains routines to read commandline
MemInfoClass - access info about Simula objects
MemManagerClass - contains routines to control Simula memory management
MemoryAccess - Routines to access raw memory
MemStatistics - statistics of created Simula objects
UnsafeConversion - Routines to override types for simple values

Unix dependent classes

Character_IO - Perform raw charcter I/O to the terminal
DirectoryFile - Read the file-names of a filesystem Directory
FilenameClass - contains routines to manipulate filenames
FileStatus - routines to access Unix filesystem info about a file
FileUtil - delete/rename/check if exist on files
UnixCmdLineClass - read command line and environment variables in raw form
UnixUtil - error codes from errno.h and related routines

2 Organisation and use

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

Declaration in a Simula program:

external class BitFiddleClass;

Compilation:

% **simcomp <program> -L=/usr/local/simulabin -l=simlib**
(or just: % **simcomp <program> -l**)

Linking:

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

3 Detailed Interfaces for Portable classes

3.1 AnyObject

class AnyObject;

Can be used as outermost abstract superclass to simulate systems with a single inheritance tree. Note that such sub-classes must be separately compiled since sub-classes otherwise will be rooted in their enviroment of surrounding blocks.
Supers: -
Kind: Abstract
Init: none
Sequencing: -

3.2 Linkage

This is an alternative implementation of the functionality provided by the class Simset included in the Simula standard. The difference is in the packaging. Here it has the form of three classes in the same 'module'. It can thus be used much more freely than the standard package, which must be used as a prefix class (at the outermost superclass).

DECLARATION AND USE

With Standard Simset:

```
Simset class PackageA; ......  <no use of simset in PackageA>;
PackageA class PackageB;
begin
   Link class myNotice; .....;
end - PackageB -;
```

With this implementation

```
class PackageA; .....;
PackageA class PackageB;
begin
   external class Linkage; ! This also brings in Link and Head ;
   Link class myNotice; ..... ;
end - PackageB -;
```

Compilation: simcomp -L=/usr/local/simulabin -I=simlib
Linking: simld ... -lsimlib

DETAILED INTERFACE

class LINKAGE

external class AnyObject;
AnyObject class Linkage;

Common abstract super class for Link and Head. Implements fundamental linking capabilites. Should not be used directly in application programs, neither as objects nor as direct super class.
Supers: AnyObject
Kind: Abstract
Init: none
Sequencing: -

Suc

ref(Link) procedure Suc;

Return the successor of this element in its list, or none if it is last in the list or not in a list at all.

Pred
ref(Link) procedure Pred;

Return the predecessor of this element in its list, or none if it is first in the list or not in a list at all.

Prev
ref(Linkage) procedure Prev;

Return the predecessor of this element in its list, seen as a cyclic list. The Head object is returned if this element is first in the list or none is returned if it is not in a list at all.

class HEAD

Linkage class Head;

Head objects represents a list by itself. An empty list consists of just a Head object. Head objects gives also empty lists identity. It also makes traversals of lists simpler since the head object act as end-markers.
Subclasses of Head are used to implement operations that are applicable for entire lists or 'sets', such as traversals.
Supers: Linkage, AnyObject
Kind: Subclassable, Instantiable
Init: none
Sequencing: (Suc/Pred/Prev/First/Last/Empty/Cardinal/Clear)*

First
ref(Link) procedure First;

Return the first element in the list, or none if the list is empty.

Last
ref(Link) procedure Last;

Return the last element in the list, or none if the list is empty.

Empty
boolean procedure Empty;

Return true if there are no elements in the list.

Cardinal
integer procedure Cardinal;

Return the number of elements in the list, =0 if it is empty

Clear
procedure Clear;

Remove all the elements of this list (it becomes empty).

class LINK

Linkage class Link;

Link objects represent individual elements that can be part of a list. A Link object can be part of at most one list at a given time, but can be moved between lists. Subclasses of Link are used for operations applicable for single elements, and attributes needed for each object.
Supers: Linkage, AnyObject
Kind: Subclassable, Instantiable
Init: none
Sequencing: (Suc/Pred/Prev/Out/Follow/Precede/Into)*

Out
procedure Out;

If this element is part of a list, remove it from the list.

Follow
procedure Follow(ptr); ref(Linkage) ptr;

Make this element follow the object 'ptr' in a list. If this element is part of a list, it is first removed from that list. If ptr is none, or not member of any list, there is no other effect than the possible removal. Notice: calling Follow on a Head object will insert this element as the first element of the list.

Precede

procedure Precede(ptr); ref(Linkage) ptr;

Make this element precede the object 'ptr' in a list. If this element is part of a list, it is first removed from that list. If ptr is none, or not member of any list, there is no other effect than the possible removal. Notice: see Into for inserting an object last in a list.

Into

procedure Into(H); ref(Head) H;

Make this element the last element of a list. If this element is part of a list, it is first removed from that list. If H is none, there is no other effect than the possible removal.

3.3 BitFiddleClass

class BitFiddleClass;

Operations for logical bitwise manipulations on 32-bit integers.
This class only specifies operations and have no state varaiables. All data to the operations are given as paramaters.
The class might serve useful to use this class as a super class for more specialized abstractions, such as BitSetClass.
Supers: -
Kind: Instantiable
Init: none (Defualt Bit0Low)
Sequencing: (Bit0Low / Bit0High / BitClear / BitSet / BitGet / BitNot / BitAnd BitIor / BitXor / BitFirst / BitShift)*

Operations

Bit0Low

procedure Bit0Low;

Set bitordering so bit number 0 is the least significant bit.
bitnumbering: 31,30,29,...,0 (bit 0 lowest significans, bit 31 is sign).
This is default.

Bit0High

procedure Bit0High;

Set bitordering so bit number 0 is the most significant bit.
bitnumbering: 0,1,2,..,31 (bit 0 is signbit, bit 31 is least significant.)

BitClear

integer procedure BitClear(i,bno);
integer i; ! Input bitpattern;
integer bno; ! Affected bit number;

Return the value of i with bit numbered bno cleared (=0).

BitSet

integer procedure BitSet(i,bno);
integer i; ! Input bitpattern;
integer bno; ! Affected bit number;

Return the value of i with bit numbered bno set (=1).

BitGet

Boolean procedure BitGet(i,bno);
integer i; ! Input bitpattern;
integer bno; ! Affected bit number;

Return Value of bit bno of i.

BitNot
integer procedure BitNot (i);
integer i; ! Input bitpattern;
Return the bitwise inverted value of i.

BitAnd
integer procedure BitAnd(i,j);
integer i,j; ! Input bitpatterns;
Return the result of i AND j calculated bitwise.

BitIor
integer procedure BitIor(i,j);
integer i,j; ! Input bitpatterns;
Return the result of i IOR j calculated bitwise.

BitXor
integer procedure BitXor(i,j);
integer i,j; ! Input bitpatterns;
Return the bit-number of the first set bit in i. If there is no set bit, the result is -1.

BitFirst
integer procedure BitFirst(i);
integer i; ! Input bitpattern;
Return the bit-number of the first set bit in i. If there is no set bit, the result is -1.

BitShift
integer procedure BitShift(i,s);
integer i; ! Input bitpattern;
integer s; ! multiplication with 2**s;
Return the integer result of shifting i s steps. If Bit0low this means shifting left (s>0) or right (s<0). If Bit0High the shift direction is reversed so shift left corresponds to multiplication and right to division in both cases. This is a logical shift, so all bits shifted in are zeroes.

3.4 BitSetClass

BitFiddleClass class BitSetClass(MaxSet);
integer MaxSet;
BitSetClass is set of integers, compatcly represented. The members are numbers 0,1,...,MaxMember. MaxMember is given when an object of this class is created.
Supers: BitFiddleClass
Kind: Instantiable
Init: none
Sequencing: (MaxMember AddMember RemoveMember isMember FirstMember SetClear SetNot SetCopy SetUnion SetCut SetEqual LoadFromArray StoreToArray)*

Operations

MaxMember
Returns the highest member number in this set (=MaxSet).

AddMember
procedure AddMember(number);
integer Number;
Include Number in the set. If Number is outside 0..MaxMember the call is ignored.

RemoveMember
procedure RemoveMember(Number);
integer Number;

Remove Number from the set if a member. If Numer is outside 0..MaxMember the call is ignored.

isMember
Boolean procedure isMember(number);
integer Number;

Return True if Number is member of the set, False otherwise. If Number is outside 0..MaxMember it also returns False.

FirstMember
integer procedure FirstMember;

Return the lowest Number of a member in the set. Returns -1 for an empty set.

SetClear
procedure SetClear;

Make the set contain no members.

SetNot
procedure SetNot;

Make the set contain exactly those Numbers that where not members before.

SetCopy
ref(bitsetClass) procedure SetCopy;

Return a copy of the set that initally contains the same Members.

SetUnion
procedure SetUnion(otherSet);
ref(bitSetClass) otherSet;

Make the set conatain also those Numbers that are members of another Set. Numbers of otherSet, outside 0..MaxMembers of this Set are not considered.

SetCut
procedure SetCut(otherSet);
ref(bitSetClass) otherSet;

Make this set contain only those Numbers that initally are members of both sets.

SetEqual
Boolean procedure SetEqual(otherSet);
ref(bitSetClass) otherSet;

Return true if both sets contain the same members. If the two sets are of different size they are considered equal if the corresponding parts are equal and the extension part of the larger set is empty.

LoadFromArray
procedure LoadFromArray(ar);
integer array ar;

Regard an integer array as a set member representation where the most significant bit of the first element corresponds to member O and so on. Make the set contain exactly those members where there is a bit=1 in the integer array. Bits outside the range 0..MaxMembers are not included.

StoreToArray
procedure StoreToArray(ar);
integer array ar;

Regard an integer array as a set member representation where the most significant bit of the first element corresponds to member O and so on. Set the bits in the integer array corresponding to the members of the set. The set will be truncated if the array is too short.

3.5 BitPackClass

class bitpackclass;

This class is used to store Numbers as bytes of bitpatterns in a text string. It is usefull for storing or communicating numbers in a compact form. The number of bytes a number occupies is: short integer 2, integer4, real 4 and long real 8. The bytes are placed from pos and onwards, the pos is advanced 2/4/8 positions as required. If the text is not long enough a runtimne Error is reported.
This class contains operations only. Data is stored (or fetched) from a text parameter to each operation.
Supers: -
Kind: Instantiable
Init: none
Sequencing: (PackShort / UnPackShort / PackInt / UnPackInt / PackReal / UnPackReal / PackLong / UnPackLong)*

Operations

PackShort

procedure PackShort(t,S);
name T; text T; ! Output buffer, pos incremented with 2.;
short integer S; ! Input value to pack into T.;

Put S into T, starting from pos, and increment T.pos with 2 If the rest of T is less than 2 characters long the routine will make a runtime error.

UnPackShort

Short Integer procedure UnPackShort(T);
name T; text T; ! Input, next 2 bytes, T.pos incremented.;

Extract 2 bytes from T, starting at pos, and increment T.pos with 2. The 2 bytes are made into a short (16-bit) integer which is returned.

PackInt

procedure PackInt(T,Int);
name T; text T; ! Output buffer, pos incremented with 4.;
integer Int; ! Input value to pack into T.;

Put Int into T, starting from pos, and increment T.pos with 4 If the rest of T is less than 4 characters long the routine will make a runtime error.

UnPackInt

Integer procedure UnPackInt(T);
name T; text T; ! Input, next 4 bytes, T.pos incremented.;

Extract 4 bytes from T, starting at pos, and increment T.pos with 4. The 4 bytes are made into a 32-bit integer which is returned.

PackReal

procedure PackReal(T,R);
name T; text T; ! Output buffer, pos incremented with 4.;
real R; ! Input value to pack into T.;

Put R into T, starting from pos, and increment T.pos with 4 If the rest of T is less than 4 characters long the routine will make a runtime error.

UnPackReal

Real procedure UnPackReal(T);
name T; text T; ! Input, next 4 bytes, T.pos incremented.;

Extract 4 bytes from T, starting at pos, and increment T.pos with 4. The 4 bytes are made into a 32-bit Real which is returned.

PackLong

procedure PackLong(T,L);
name T; text T; ! Output buffer, pos incremented with 8.;
long real L; ! Input value to pack into T.;

Put R into T, starting from pos, and increment T.pos with 8 If the rest of T is less than 8 characters long the routine will make a runtime error.

UnPackLong
Long real procedure UnPackLong(T);
name T; text T; ! Input, next 8 bytes, T.pos incremented.;

Extract 8 bytes from T, starting at pos, and increment T.pos with 8. The 8 bytes are made into a 64-bit Long real which is returned.

3.6 CallDebugger

procedure CallDebugger;

Start the Simula debugger as if the program was interrupted with a ctrl-\. The program may be continued after the debugging session. Notice that this procedure acts as a no-op if the program is not linked with the Simula debugger.

3.7 CmdLineClass

class CmdLineClass;

Class for reading information from the command line using the Simula 'standard' way of interpreting options, called 'parameters' and 'arguments'.
A parameter is a named parameter to the program. Parameters come in three types, Boolean, integer and text. Parameters are either present or not on the command line.
An argument is a string, not starting with a '-' that appear on the command line. Arguments are numbered and many programs can take a variable number of arguments.
In UNIX a parameter is syntactically distinguished with an initial '-' and values are separted with a '=' from the parameter name.
Using this class means your program will show a command-line interface similar to other Simula programs. As an alternative, the class UnixCommandLine offers a more primitive interface to the command line, making it possible to write Simula programs mimicing all the different versions of command line interfaces used by Unix utilities.
Supers: -
Kind: Instantiable
Init: none
Sequencing: (ProgramName ParBool ParInt ParInt ParText ArgCount ArgText)*

Operations:

ProgramName
procedure ProgramName(b);
name b;text b; ! Buffer for result;

Return name of the executing program. Result: b:=<program name>, b.setpos(string.length+1).
NOTE: that b must be long enough to store the result.

ParBool
Boolean procedure ParBool(T);
text t; ! Name of parameter;

Return booolean value of named parameter.
Result: true if t present, false if not.

ParInt
Boolean procedure ParInt(T,V);
name V; integer V; ! Returned valeu - if any;
text t; ! Name of parameter;

Return Integer value of named parameter.
Result: True and V=number if T=<number> present, False if not.

ParText
Boolean procedure ParText(t,b);
name b; text b; ! Buffer for result (in B.sub(1,B.pos-1));
text t; ! Name of the parameter asked for.;

Return text value of named parameter in T. Result: if t=<string> present return True and b:=string, b.setpos(string.length+1), return False if not.
NOTE: that b must be long enough to store the result.

ArgCount
integer procedure ArgCount;

Return number of arguments on the command line. Result: number of arguments.

ArgText
Boolean procedure ArgText(ArgN,Buf);
name Buf;text Buf; ! Buffer to hold the result.;
integer ArgN; ! Argument number to get. ;

Return text value of numbered argument. Result: if there is a ArgN:th argument, True is returned and Buf:=the argument, Buf.setpos(string.length+1), otherwise it return False.
NOTE: that Buf must be long enough to store the result.

3.8 MemInfoClass

class Meminfoclass;

The routines in this class can scan the heap and access information about the Simula objects. This class is interfacing to low level facilities and not intended for general use. See also MemStatistics.
Supers: -
Kind: Instantiable
Init: MemInit
Sequencing: (MemInit (NextTemp (TemplateType/BlockNr/Module)*)*)*

Operations

MemInit
procedure MemInit;

Initiates the module. Call this routine once before calling any of the other routines.

NextTemp
integer procedure NextTemp;

Returns the "type" of the next object in the heap. This is in the form of a unique integer for each class or procedure, its "template" Returns zero when no more objects exist.

TemplateType
integer procedure TemplateType(T);
integer T; ! Template identifer, returned by NextTemp.;

Returns the type of the template T (class procedure etc.). These are coded as integers, all blocks have Type>= 8R200.

BlockNr
integer procedure BlockNr(T);
integer T; ! Template identifer, returned by NextTemp.;

Returns the unique block identifier (the index in -.atr file) matching the template T which must be block (TemplateType>=BlockType.

Module
procedure Module(T,Buf);
name Buf; text Buf; ! Buffer to contain the result.;
integer T; ! Template identifer, returned by NextTemp.;

Returns the name of the source file where the block corresponding to the template, T, is defined. T must be a block, (TemplateType>=BlockType).

Constants

BlockType
integer BlockType=8R200; ! Lowest block-type value.;

3.9 MemoryAccess

class MemoryAccess;

The operations in this class facilitates access to values by machine addresses, typically obtained through calls to C routines. One situation where it is interesting to use these operations is when interfacing to a C-routine that return the address of a C struct (or a C array) as the result. The operations to fetch the values can be used to copy the entier struct (or array) to a corresponding Simula array. In this case one has to know the length of the array, often defined in C headerfiles.
The operations can also be used to fetch values from a C-struct one by one. In this case one has to know also the offsets in the C-strucure for each value.
The second set of operations can be used to fill in a C-structure that is earlier allocated by C. This might be useful when calling C-routines, but often it is more convenient to call such a routine with a Simula integer array, emulating the C-struct.
Supers: -
Kind: Instantiable
Init: none
Sequencing:
(GetCIntAt/GetCShortAt/GetCRealAt/GetCLongAt/
CArLength/CIntArray/CStringToText/CStringArToTextAr)*
(PutCIntAt/PutCShortAt/PutCRealAt/PutCLongAt/TextToCString/free)*

Operations to fetch values from C address-space

GetCIntAt
integer procedure GetCIntAt(CAddress,COffset);
integer CAddress; ! Base address ;
integer COffset; ! Offset from base in bytes, added to CAddress.;

Return the integer located at 'CAddress'+'COffset'. If CAddress is not a legal address, this operation will most likely crash the Simula program.

GetCShortAt
integer procedure GetCShortAt(CAddress,COffset);
integer CAddress; ! Base address ;
integer COffset; ! Offset from base in bytes, added to CAddress.;

Return the 16-bit integer located at 'CAddress'+'COffset'. The result is sign-extended to a propper integer value. If CAddress is not a legal address, this operation will most likely crash the Simula program.

GetCRealAt
real procedure GetCRealAt(CAddress,COffset);
integer CAddress; ! Base address ;
integer COffset; ! Offset from base in bytes, added to CAddress.;

Return the floating point number located at 'CAddress'+'COffset'. If CAddress is not a legal address, this operation will most likely crash the Simula program.

GetCLongAt
long real procedure GetCLongAt(CAddress,COffset);
integer CAddress; ! Base address ;
integer COffset; ! Offset from base in bytes, added to CAddress.;

Return the double precision floating point number located at 'CAddress'+'COffset'. If CAddress is not a legal address, this operation will most likely crash the Simula program.

CArLength

```
integer procedure CArLength(CArray);
integer CArray;
```

Returns length of NULL-terminated array of Addresses in C. 'CArray' is the location of the first address, CArLength reads successive words until a zero word is found and returns the count of the non-zero words.

CIntArray

```
procedure CIntArray(CArray,CSize,IArray);
name IArray; ! result parameter;
integer CArray; ! Start of C array to copy from;
integer CSize; ! Length of C array to copy;
integer array IArray; ! Simula array to copy to;
```

Copy 'CSize' integers from the location 'CArray' to the Simula array IArray, starting at its first element. If the number of elements in IArray is smaller than CSize, only the first part of CArray is copied.

CStringToText

```
procedure CStringToText(CString,TReturn);
name TReturn; ! Result parameter, becomes new Simula text object.;
Integer CString; ! Address of first byte in string to copy.;
Text TReturn; ! Returned as result;
```

Copy the content of memory at location 'CString' until a NULL byte is found. Place the copied bytes in a new Simula text returned as 'TReturn'. This procedure creates a new Simula text and initializes it to the contet of the CString parameter.

CStringArToTextAr

```
procedure  CStringArToTextAr(CArray,CSize,TArray);
name TArray; ! result parameter;
integer CArray; ! Start of C array, to copy from;
integer CSize;  ! Size of C array ;
text array TArray; ! Simula array to copy to;
```

Copy the C strings denoted by 'CArray' to new Simula texts and assign to Simula text array 'TArray'. Lowest index of each array match. Copy at most CSize strings or, if the Simula array is shorter, only as many as to fill 'TArray'. For each string bytes are copied up to the first NULL character. Each Simula text is allocated to match this length.

Operations to store values in C address-space

PutCIntAt

```
procedure PutCIntAt(CInt,CAddress,COffset);
integer CInt; ! Integer to store;
integer CAddress; ! Base address ;
integer COffset;  ! Offset from base in bytes, added to CAddress.;
```

Store the integer 'CInt' at location 'CAddress'+'COffset'. If CAddress is not a legal address, this operation will most likely crash the Simula program.

PutCShortAt

```
procedure PutCShortAt(CShort,CAddress,COffset);
integer CShort; ! A 16-bit integer to store;
integer CAddress; ! Base address ;
integer COffset;  ! Offset from base in bytes, added to CAddress.;
```

Store the 16-bit integer 'CShort' at location 'CAddress'+'COffset'. If CAddress is not a legal address, this operation will most likely crash the Simula program.

PutCRealAt
procedure PutCRealAt(CReal,CAddress,COffset);
real CReal; ! real (C-float) value to store;
integer CAddress; ! Base address ;
integer COffset; ! Offset from base in bytes, added to CAddress.;
Store the floating point number 'CReal' at location 'CAddress'+'COffset'. If CAddress is not a legal address, this operation will most likely crash the Simula program.

PutCLongAt
procedure PutCLongAt(CLong,CAddress,COffset);
long real CLong; ! Long real (C-double) value to store;
integer CAddress; ! Base address ;
integer COffset; ! Offset from base in bytes, added to CAddress.;
Store the double precision floating point number 'CLong' at location 'CAddress'+'COffset'. If CAddress is not a legal address, this operation will most likely crash the Simula program.

TextToCString
integer procedure TextToCString(SText);
name SText; ! (Adr of text-box);
text SText; ! Simula text to copy;
Copy the content of 'SText', NULL terminated, to an new memory area allocated in C address spece. Return the address of the C string copy as an integer. Note that this string needs to be freed when no longer in use by a call to 'free' either by the Simula program, or by the C program (but not both).

free
procedure free(CAdr);
integer CAdr; ! memory area to free.;
Call the C utility 'free' to release a memory area earler obtained by a call to 'malloc'.

3.10 MemStatistics

MemInfoClass class MemStatistics(MaxObjectTypes);
integer MaxObjectTypes; ! Largest number of class/procedure/array etc types.;
This module collects and prints memory statistics of heap usage. The parameter defines the limit of how many different object types that can be used for statistics collection, also the number of lines printed. The information can be dumped to a text file for inspection. This class needs more facilities in order to be of general use as a memory inspection tool.
Supers: MemInfoClass
Kind: Instantiable
Init: call Clear.
Sequencing: (Clear Scan Print)*

Operations

Clear
procedure Clear;
Reset the module to init state. Call to get a fresh sample.

Scan
procedure Scan;
Collect statistics over the objects in the entier heap.

Print
procedure Print(F);
ref(outfile) F; ! The parameter must be an open outfile.;
Print the collected statistics onto the file in form of a table:

count:module-name/block-id - or-
count:NON-BLOCK-TYPE

3.11 MemManagerClass

class MemManagerClass;

The routines in this class can be used to control the actions of the garbage collector. One can measure the memory usage and request Garbage Collection from the user program. This might be interresting in order to avoid GC during sensitive periods of program execution.
Supers: -
Kind: Instantiable
Init: none
Sequencing: (GetFreeMemory / CallGC / NeedsMemory)*

Operations

GetFreeMemory
integer procedure GetFreeMemory;

Return the amount of free memory in bytes, e.g. what can be allocated before next garbage collection.

CallGC
procedure CallGC;

Call the garbage collector to free memory.

NeedsMemory
procedure NeedsMemory(Amount);
integer Amount; ! Low water mark.;

Check if Amount bytes of memory can be allocated without calling the GC, if not the GC is called now. If after the GC there is not enough memory available the execution is interrupted.

3.12 UnsafeConversion

class UnsafeConversion;

This class contains routines that change type on bitpatterns. They are useful when interfacing to external routines or doing I/O to unformatted interfaces. If for example a bitpattern has been obtained as an integer, but it is know that the bitpattern should really be interpreted as a real. Accordingly, a program can use this class to output a real number as an integer bitpattern.
The routines converting to/from 16-bit integers can be useful when interfacing to C-programs that use this packed form in e. g. structs or in arrays, since Simula is using 32bit integers also for 16-bit values.
Supers: -
Kind: Instantiable
Init: none
Sequencing:
(ItoSS / SStoI / ItoReal /RealtoI / IItoLong / LongtoII)*

Operations

ItoSS
procedure ItoSS(I,s1_result,s2_result);
name s1_result,s2_result;
integer i;
short integer s1_result;!:=I//2**16, sign extended;
short integer s2_result;!:=mod(I,2**16), sign extended;

Split an 32-bit integer into two 16-bit intergers containing the most significant(s1) and least significant(s2) bits. Both results are sign extended.

SStoI

```
integer procedure SStoI(s1,s2);
short integer s1,s2;
```

Combine two 16-bit integers into one 32-bit integer. Returns s1*2**16+s2, except that the numbers are masked logically AND together.

ItoReal

```
real procedure ItoReal(Bits);
integer Bits; ! Formally an integer but bitpattern of a real;
```

This procedure returns the same bitpattern as given as parameter, but formally as a real. Can be used to circumvent Simula typechecking of simple values.

RealtoI

```
integer procedure RealtoI(Bits);
real Bits; ! Formally a real but bitpattern of an integer;
```

This procedure returns the same bitpattern as given as parameter, but formally as an integer. Can be used to circumvent Simula typechecking of simple values.

ItoLong

```
long real procedure ItoLong(I1,I2);
integer I1,I2;
```

Combine two 32-bit integers into one 64-bit long real. This procedure returns the bitpatterns given to it, changing the type to long real. I1 is the high order bits of the representation of a long real (including the sign bit and exponent) while I2 is the low order bits of the mantissa.

LongtoI

```
procedure LongtoI(Bits,I1,I2);
name I1,I2;
Long real Bits;
integer I1,I2;
```

Splits one 64-bit long real into two 32-bit integer bit-patterns. This procedure returns the bitpattern given to it. I1 is the high order bits of the representation of a long real (including the sign bit and exponent) while I2 is the low order bits of the mantissa.

4 Detailed Interfaces for Unix related classes

4.1 Character_IO

class Character_IO;

Perform character by charcter I/O to the terminal, bypassing sysin/sysout. The characters are read without echo and written directly, no expanssion of tabs etc are done. The routins are used to write programs making interactive use of Vt100-type terminals. When these routines are used normal I/O should not be used.
Supers: -
Kind: Instantiable
Init: -
Sequencing: (Open (Input/Output/OutString)* Close)*

Operations

Open
procedure Open;

Must be called before any I/O operation. Will terminate the program if it can not be performed.

Close
procedure Close;

Called to reset the terminal to normal operation. Should be called before Sysin/sysout I/O is preformed or the program is terminated.

Input
character procedure Input;

Wait until the user hits a key, then return the ASCII equivelent. There is no automatic echoing on the terminal.

Output
procedure Output(Ch); character Ch;

Send the character Ch the the terminal as unprocessed as possible.

OutString
procedure OutString(T,Length);
text T; integer Length;

Send the first Length characters of T to the terminal. This operation is equivalent to, but more efficient than, repeated calls to Output.

4.2 DirectoryFile

class DirectoryFile(FileName);
text FileName;

After succefully opened, the filename of a directory can be read sequentially by calling NextEntry to advance and EntryName to access the name of the file. This class only reports the name of the files in a directory. See the class FileStatus for getting information about the indiviual files.
Supers: -
Kind: Instantiable
Init: DF :- new DirectoryFile("aDirectoryFilename") ;
Sequencing: (Open (NextEntry EntryName)* Close)*

Operations

Open
Boolean procedure Open;

Return True if the file is a directory and can be opened for reading.

Close
Boolean procedure Close;
Returns true if the file could be closed succesfully.

NextEntry
Boolean procedure NextEntry;
Advances to the next entry in the directory. Returns True if there was yet another entry and false when there are no more entries.

Entryname
text procedure Entryname;
Returns the "filename" of the current Entry in the directory.

4.3 FileNameClass

class FilenameClass;
Class for manipulation of filenames, extracting parts and combining directory paths and single file names. All operations return NEW simula texts.
These operations are the simliar to the "modifiers" h, t, r, e in UNIX csh. The parts of a filename are named as follows:

```
  head    - the part before the last slash
  tail    - the part after the last slash
  basename- the part before the last dot
  suffix  - the part after the last dot
  slash   - separator between directories, in Unix a '/'
  dot     - separator between basename and suffix, in Unix '.'
```

If there is no slash in the filename, then head=tail=filename
If there is no dot in the filename, then basename=filename, suffix=NOTEXT
Example

```
 Argument:  a/b/c.d  c    .d   c.   /    a/  /a  -
 Results:
  head      a/b     c    .d   c.   -    a   -   -
  tail      c.d     c    .d   c.   -    -   a   -
  basename  a/b/c   c    -    c    /    a/  /a  -
  suffix    d       -    d    -    -    -   -   -
```

(Special cases: '-' means NOTEXT)

Supers: -
Kind: Instantiable
Init: none
Sequencing: (Head/Tail/Basename/Suffix/HeadAndTail/DefineSeparators)*

Operations:

Head
text procedure Head(F);
text F; ! full file name.;
Returns the head of filename F.

Tail
text procedure Tail(F);
text F; ! full file name.;
Returns the tail of F.

Basename
text procedure Basename(F);
text F; ! full file name.;
Returns the basename of F.

Suffix
text procedure Suffix(F);
text F; ! full file name.;
Returns the suffix of F.

HeadAndTail
text procedure HeadAndTail(H,T);
text H,T; ! Head and Tail part of constructed filename. ;
Returns H & '/' & T (if default value on separator 'Slash').

BasenameAndSuffix
text procedure BasenameAndSuffix(B,S);
text B,S; ! Basename and Suffix of the constructed filename. ;
Returns B & '.' & S (if default value on separator 'Dot')

DefineSeparators
procedure DefineSeparators(aSlash,aDot);
character aSlash, aDot;
Re-defines the separators, default vaules are '/' and '.' respectively.

4.4 FileStatus

BitFiddleClass class FileStatus;
Access file system info of a file, such as type and creation date.
Supers: BitFiddleClass
Kind: Instantiable
Init: call SetFile before other methods.
Sequencing: (SetFile
(Device / Inode / Protection / FileType / HardLinks/ OwnerUserId /
OwnerGroupId / DeviceType / FileSize / LastAccessTime / LastModifyTime /
LastStatusChange / OptimalBlockSize / BlockAllocated)*)*

Operations

SetFile
boolean procedure SetFile(fname);
text fname; ! Full name of file to look up.;
Fill this object with nformation regarding 'fname'. Future calls to other routiens of this class will return values regarding this file until SetFile is called again.
Returns false if file 'fname' is not found.

Device
integer procedure Device;
Return the 'device' attribute.

Inode
integer procedure Inode;
Return the 'Inode' attribute of the file.

Protection
integer procedure Protection;
Return the 12 file 'protection' bits

FileType
integer procedure FileType;
returning an integer, with one of the following values (pre-defined constants):
FIFO= fifo ;
FCHR= character special ;
FDIR= directory ;
FBLK= block special ;
FREG= regular ;
FLNK= symbolic link ;
FSOCK= socket ;

These constants are avialable as attributes of this class. Soft links are reported as links (rather than as the file they denote).

HardLinks
integer procedure HardLinks;
How many hard links to this file.

OwnerUserId
integer procedure OwnerUserId;
User Id of file owner.

OwnerGroupId
integer procedure OwnerGroupId;
Group Id of file owner.

DeviceType
integer procedure DeviceType;
Device Type of filesystem hosting the file (types=?).

FileSize
integer procedure FileSize;
Size of the file in bytes.

LastAccessTime
integer procedure LastAccessTime;
in seconds since year 0, 1970, Jan 1st.

LastModifyTime
integer procedure LastModifyTime;
in seconds since year 0, 1970, Jan 1st.

LastStatusChange
integer procedure LastStatusChange;
in seconds since year 0, 1970, Jan 1st.

OptimalBlockSize
integer procedure OptimalBlockSize;

BlockAllocated
integer procedure BlockAllocated;
Size of file in disk blocks.

4.5 FileUtil

class FileUtil;
OS related operations on Simula files that can not, or is not convenient to do through the Simula File 'Access' function.
Operations placed in this class are intended to be 'portable' and implementable in most operating systems.
Supers: -
Kind: Instantiable
Init: -
Sequencing: (Exist/Delete/Rename/FileNumber)*

Operations

Exist
Boolean procedure Exist(T);
text T; ! Full name of the file.;
Return true if a file exist. The parameter is the filename of the file. Note: This routine returns true even if the file can not be opened because of access priviliges.

Delete
Boolean procedure Delete(t);
text t; ! Full name of the file.;

This procedure deletes a file if it exists. Returns false if the file did not exist or couldn't be deleted (access priviliges).

Rename
Boolean procedure Rename(told,tnew);
text told; ! Full filename as the file is currently known.;
text tnew; ! Full, new, name the file will be known as.;

This procedure renames a file. Returns false if it couldn't be done. Notice that files can not be moved between filesystems through renaming.

FileNumber
integer procedure FileNumber(f);
ref(file) f; ! Open Simula file.;

This procedure returns the Unix filenumber (0,1,2,..) of an open Simula file. Returns -1 if it is not open. Useful when interfacing to Unix system software.

4.6 UnixCmdLineClass

class UnixCmdlineClass;

Basic interface to Unix program 'parameters' (from the command line and from environment variables). Command line arguments are returned in uniterpreted Unix form.
Use this class if you want to write Simula programs which look like standard Unix applications. If you want to write portable Simula programs use the routines in CmdLineClass instead which exists for several non Unix Simula implementations.
Supers: -
Kind: Instantiable
Init: -
Sequencing: (MaxCmd/CmdArg/MaxEnv/EnvArg/GetEnv)*

Operations

MaxCmd
integer procedure MaxCmd;

Return the number of arguments on the command line, numbered 1,2 ...

CmdArg
text procedure CmdArg(ix);
integer ix;

Return the command line argument number index as a Simula text. If index is out of range (1..MaxCmd) it returns notext. With ix=1 it returns the name of the program/script used to start this process.

MaxEnv
integer procedure MaxEnv;

Return the number of environment variables numbered 1,2 ...

EnvArg
text procedure EnvArg(ix);
integer ix;

Return the environment variable number index as a Simula text in the same format as when listed with 'printenv'. If index is out of range (1..MaxEnv) it returns notext.

GetEnv
Text procedure GetEnv(Arg,Defined);
name Defined; text Arg; Boolean Defined;

Returns True and the 'value' of Arg if defined as env-variable. Ex: given: 'EDITOR=vi' will make GetEnv("EDITOR") return: "vi" and Defined=True. Returns Defined=False (and Notext) if 'Arg' is not defined. Notice that the combination Notext and True means that the 'Arg' is defined as the empty string.

4.7 UnixUtil

class UnixUtil;

This class is intended to give support Unix systems programming in Simula. Currently it contains a routine to execute sub-processes, to get the latest system call status ("errno") and the predefined Unix system-error constants.
Supers: -
Kind: Instantiable
Init: -
Sequencing: (System/GetErrNo/PError/E-constants)*

Operations

System

integer procedure System(Cmd);
text cmd; ! Shell command to execute inculding options like "ls -ls";

Start a shell and perform the command cmd. Wait until the command is completed (if system call is interrupted). Return the exit code from the command.

GetErrNo

integer procedure GetErrNo;

Returns the status (errno) from last issued unix call. See constants for the meaning of the error codes.

PError

procedure PError(msg);
value msg; text msg; ! Text, often identifing the complaining program.;

Print the message followed by a decription of the last generated error.

Unix Error constants

Basic errors

```
integer
EPERM=1,!- Not owner -;
ENOENT=2,!- No such file or directory -;
ESRCH=3,!- No such process -;
EINTR=4,!- Interrupted system call -;
EIO=5,!- I/O error -;
ENXIO=6,!- No such device or address -;
E2BIG=7,!- Arg list too long -;
ENOEXEC=8,!- Exec format error -;
EBADF=9,!- Bad file number -;
ECHILD=10,!- No children -;
EAGAIN=11,!- No more processes -;
ENOMEM=12,!- Not enough core -;
EACCES=13,!- Permission denied -;
EFAULT=14,!- Bad address -;
ENOTBLK=15,!- Block device required -;
EBUSY=16,!- Mount device busy -;
EEXIST=17,!- File exists -;
EXDEV=18,!- Cross-device link -;
ENODEV=19,!- No such device -;
ENOTDIR=20,!- Not a directory-;
EISDIR=21,!- Is a directory -;
EINVAL=22,!- Invalid argument -;
ENFILE=23,!- File table overflow -;
EMFILE=24,!- Too many open files -;
ENOTTY=25,!- Not a typewriter -;
ETXTBSY=26,!- Text file busy -;
EFBIG=27,!- File too large -;
ENOSPC=28,!- No space left on device -;
ESPIPE=29,!- Illegal seek -;
EROFS=30,!- Read-only file system -;
EMLINK=31,!- Too many links -;
EPIPE=32,!- Broken pipe -;
```

Math software errors

```
EDOM=33,!- Argument too large -;
ERANGE=34,!- Result too large -;
```

Non-blocking and interrupt i/o errors

```
EWOULDBLOCK=35,!- Operation would block -;
EINPROGRESS=36,!- Operation now in progress -;
EALREADY=37,!- Operation already in progress -;
```

IPC/network software errors

- Argument errors -;
ENOTSOCK=38,!- Socket operation on non-socket -;
EDESTADDRREQ=39,!- Destination address required -;
EMSGSIZE=40,!- Message too long -;
EPROTOTYPE=41,!- Protocol wrong type for socket -;
ENOPROTOOPT=42,!- Protocol not available -;
EPROTONOSUPPORT=43,!- Protocol not supported -;
ESOCKTNOSUPPORT=44,!- Socket type not supported -;
EOPNOTSUPP=45,!- Operation not supported on socket -;
EPFNOSUPPORT=46,!- Protocol family not supported -;
EAFNOSUPPORT=47,!- Address family not supported by protocol family -;
EADDRINUSE=48,!- Address already in use -;
EADDRNOTAVAIL=49,!- Can't assign requested address -
- Operational errors -;
ENETDOWN=50,!- Network is down -;
ENETUNREACH=51,!- Network is unreachable -;
ENETRESET=52,!- Network dropped connection on reset -;
ECONNABORTED=53,!- Software caused connection abort -;
ECONNRESET=54,!- Connection reset by peer -;
ENOBUFS=55,!- No buffer space available -;
EISCONN=56,!- Socket is already connected -;
ENOTCONN=57,!- Socket is not connected -;
ESHUTDOWN=58,!- Can't send after socket shutdown -;
ETOOMANYREFS=59,!- Too many references: can't splice -;
ETIMEDOUT=60,!- Connection timed out -;
ECONNREFUSED=61,!- Connection refused -;

Mixed errors

```
- Others;
ELOOP=62,!- Too many levels of symbolic links -;
ENAMETOOLONG=63,!- File name too long -
- Should be rearranged -;
EHOSTDOWN=64,!- Host is down -;
EHOSTUNREACH=65,!- No route to host -;
ENOTEMPTY=66,!- Directory not empty -
- Quotas & mush -;
EPROCLIM=67,!- Too many processes -;
EUSERS=68,!- Too many users -;
EDQUOT=69,!- Disc quota exceeded -
- Network File System -;
ESTALE=70,!- Stale NFS file handle -;
EREMOTE=71,!- Too many levels of remote in path -
- Streams -;
ENOSTR=72,!- Device is not a stream -;
ETIME=73,!- Timer expired -;
ENOSR=74,!- Out of streams resources -;
ENOMSG=75,!- No message of desired type -;
EBADMSG=76,!- Trying to read unreadable message -
- SystemV IPC -;
EIDRM=77,!- Identifier removed -
- SystemV Record Locking -;
EDEADLK=78,!- Deadlock condition. -;
ENOLCK=79,!- No record locks available. -
- RFS -;
ENONET=80,!- Machine is not on the network -;
ERREMOTE=81,!- Object is remote -;
ENOLINK=82,!- the link has been severed -;
EADV=83,!- advertise error -;
ESRMNT=84,!- srmount error -;
ECOMM=85,!- Communication error on send -;
EPROTO=86,!- Protocol error -;
EMULTIHOP=87,!- multihop attempted -;
EDOTDOT=88,!- Cross mount point (not an error) -;
EREMCHG=89,!- Remote address changed -
- POSIX -;
ENOSYS=90;!- function not implemented -;
```

5 Index to classes and procedures