The Mjølner System
Basic Libraries

Reference Manual

Mjølner Informatics Report

MIA 90-08(1.4)

November 1997

Contents

1. The Basic Libraries

A programmer guide

This document contains documentation on version 1.6 of the basic libraries of the Mjølner System. It is a programmer's guide. The purpose is to provide the necessary information to the users of the libraries. The facilities are documented by means of the BETA interfaces and examples of how to use the facilities.

This document includes the documentation on the following libraries of the Mjølner System:

- **`betaenv.bet`** contains the most basic library of the Mjølner System. It contains patterns describing character, integer, streams, exceptions, etc. as well as control patterns and input/output patterns. Most (if not all) BETA programs will use `betaenv`, either directly or indirectly.
- **`math.bet`** contains an interface to standard real functions in BETA. The library contains patterns for mathematical functions: trigonometric, hyperbolic, exponential and logarithmic, floating point manipulation, power, miscellaneous constants.
- **`numberio.bet`** contains patterns to be used for reading numbers from any input stream, and for writing any number to any output stream. The patterns are able to read and write all numeric types of the BETA language.
- **`formatio.bet`** contains two patterns for making formatted input and output on any stream.
- **`random.bet`** contains an elaborate random generator system, containing random generators with many different statistical properties.
- **`regexp.bet`** contains facilities for working with regular expressions in text strings.
- **`file.bet`** contains the general interface into files, residing on some file system.
- **`directory.bet`** contains the general interface into directories on hierarchical file system.
- **`unixFile.bet`** contains the UNIX specific file system interface.
- **`unixDirectory.bet`** contains the interface into the hierarchical UNIX file system directories.
- **`systemEnv`** defines the experimental concurrency system for BETA. SystemEnv contain three closely related libraries: `basicsystemenv.bet`, `systemenv.bet`, and `timehandler.bet`.
- **`repStream.bet`** contains the definition of a special type of repetitions that has stream-like operations.

- **`external.bet`** contains the various facilities for enabling BETA programs to interface to external languages, like C and Pascal.

2. The betaenv Library

When programming in BETA, the basic BETA environment `betaenv` is utilized. This chapter describes how to use the facilities in `betaenv`.

`Betaenv` contains several attributes that are used in any BETA program. That is, each BETA program must have `betaenv` in its origin path.

The patterns in `betaenv` are divided into several different categories such as character patterns, integer patterns, boolean patterns, control patterns, input/output patterns, stream and exception patterns.

The first section of this chapter describes how `betaenv` is used in general, while the subsequent sections concern the individual patterns.

2.1.1 Backward compatability

This version of betaenv contains a few important changes, which are not backward compatible. The most important are:

- The text attribute `findCh` has been renamed to `findAll`.
- The text attributes `copyAppend` and `copyPrepend` have been removed. To gain the same effect, you can write:

  ```
  '...'->(t.copy).append
  ```

  or

  ```
  '...'->(t.copy).prepend
  ```

To ease the process of porting your code, we have included a small fragment called `betaenvold.bet` containing these obsolite facilities.

2.2 Using the betaenv Library

The basic structure of `betaenv` is realized by means of the Mjølner fragment system. It is as follows:

```
BODY 'betaenvbody'
--- betaenv: descriptor ---
(# ...
   (* A lot of useful patterns *)
   ...
   theProgram: @<<SLOT program: descriptor>>;
   <<SLOT lib: attributes>>
   ...
do theProgram
#)
```

program slot

The `program` slot must be filled by the user program and can have the following form:

```
ORIGIN '~beta/basiclib/v1.6/betaenv'
--- program: descriptor ---
(#
do 'This is a small BETA program' -> puttext
#)
```

lib slot

The `lib` slot makes it possible to add attributes to `betaenv`. An example of this is the following which might be in a file called `stack.bet`:

```
ORIGIN '~beta/basiclib/v1.6/betaenv'
--- lib: attributes ---
stack:
  (# push: (# e: @integer enter e do ... #);
     pop: (# e: @integer do ... exit e #);
     empty: ...
     ...
  #)
```

The stack can then be used as follows:

```
ORIGIN '~beta/basiclib/v1.6/betaenv';
INCLUDE 'stack'
--- program: descriptor ---
(# s: @stack
do 7 -> s.push;
   s.pop -> putint;
#)
```

2.3 Basic Patterns

Simple types

`Betaenv` contains definitions of the basic patterns `char`, `integer`, `real`, `boolean`, `true` and `false` that are predefined in the BETA language (i.e. built-in data types). These patterns are self-assignable. This means an `integer` object can be assigned to another `integer` object, a `char` object can be assigned to another `char` object etc.

For efficiency reasons, the usage of the basic patterns `char`, `integer`, `real`, `boolean`, `true` and `false` is somewhat restricted, compared with all other patterns in the system. These restrictions are:

Restrictions

- They cannot be used as *superpattern:*s to other patterns. E.g. `subInteger: integer(# ... #)` is illegal.
- Dynamic references to instances of these basic types cannot be obtained. E.g. `var[] -> ...` is illegal if `var` is an instance of one of these basic patterns.
- Dynamic references to basic patterns cannot be declared. E.g. `var: ^integer` is illegal.

True object oriented patterns for integers, characters, reals and booleans are also part of the system (see later). However, using those patterns impose an execution overhead compared with the basic patterns.

2.3.1 The Integer Pattern

Arithmetical operators

Besides the arithmetical operations: `+`, `-`, `*`, `div`, and `mod` and the relational operations: =, <>, >, >=, < and <=, the `Min`, `Max` and `Abs` patterns are defined for integers. The patterns `MaxInt` and `MinInt` returns the largest (respectively smallest) integer on the machine.

```
(# i,j,k: @integer
do 3 -> i;
   '? '->puttext; getint->j; (* read an integer from keyboard *)
   ((i,j) -> Max, j+5)  -> Min -> k;
   'k is '-> puttext; k -> putint; newline;
   (if k-1
    // 5 then ...
    // 17 then ...
   if)
#)
```

2.3.2 The Boolean Pattern

Logical operators

In the current implementation, `true` and `false` return respectively the values 1 and 0. The unary operator `not` and the binary operators `and`, `xor`, and `or` can be applied to `boolean`s.

Booleans are used in the traditional way:

```
(# aBoolean, anotherBoolean: @boolean;
   a: @integer
do ...
   (if aBoolean then
       ...
   else
    (a>7) and (a<17) -> anotherBoolean
   if)
#)
```

2.3.3 The Real Pattern

Arithmetical operations

The arithmetical operations: `+`, `-`, `*`, `div` (or `/`), and the relational operations: `=`, `<>`, `>`, `>=`, `<` and `<=` are defined for reals. The patterns `MaxReal` and `MinReal` (defined in `math.bet`) returns the largest (respectively smallest) real on the machine.

The following example shows how to use reals. The pattern `putreal` is described later.

```
   (# x,y: @real;
   do ...
      1.23 -> x;
      'Print the number 1.23:' -> putline;
      y -> putreal;
      newline;
      x -> y;
      (if x=y then 'x and y are equal!' -> putline if);
      '3.0*7.0 = ' -> puttext;
      3.0 * 7.0 -> putreal;
      newline;
      '(-4.0)*(-3.0) = ' -> puttext;
      -4.0 * (-3.0) -> putreal;
      newline;
      '(-4.0)/8.0 = ' -> puttext;
      -4.0 div 8.0 -> putreal;
      newline;
   #)
```

2.3.4 The Char Pattern

ASCII

The `char` pattern enables the manipulation of characters. Characters can be expressed as literals or as the corresponding ASCII values. The pattern `Ascii` defines all non-printable characters as constants (such as `null`, `nl`, `cr`, `esc`, `del`). `Newline` however, is a variable containing either `nl` or `cr` depending on the computer.

`Ascii` also contains local patterns for various conversions and testings of characters. E.g. the patterns `IsDigit`, `IsLetter`, `IsUpper` and `IsLower` are provided for determining the kind of a character. `IsSpace` testes whether the character is `sp`, `cr`, `nl`, `np`, `ht` or `vt`. Conversion is available through the `upCase` and `lowCase` patterns.

The following example shows how to use `char` and `upCase`:

```
   (# a,b: @char
   do 'a' -> a;
      98 -> b;
      (if a -> Ascii.upCase
       // 'A' then ...
       // 'B' then ...
       ...
      if)
   #)
```

2.3.5 The Repetition Pattern

The BETA compiler also implements repetitions. Betaenv contains the repetition pattern, defining the available operations on repetitions (apart from the lookup operation: []). These operations are range, new and extend. Range returns the number of repetition positions, new makes it possible to allocate an entire new repetition, and extend is used for dynamically extension of the repetition. Note that the repetition pattern cannot be used as a *superpattern:*. Also note, that it is only allowed to make repetitions of integer, boolean, char, real, and object references.

2.4 Basic Object Patterns

The basic object oriented patterns are the object pattern and the object oriented variants of the basic patterns.

2.4.1 The Object Pattern

The object pattern functions as the implicit *superpattern:* for all patterns which do not have any explicit*superpattern:*. The pattern object is defined as follows:

```
object: (* general  superpattern: *)
   (# _struc:
        (# R: ##object
        do ...
        exit R##
        #);
      _new:
        (# O: ^object
        do ...
        exit O[]
        #)
   do inner
   #);
```

The attribute _struc returns a reference to the structure of the current object (i.e. a pattern reference to the pattern from which this object was created). The attribute _struc is maintained in v1.6 for backward-compatability reasons. It will be removed in next release, since it will become obsolite, since O## is allowed in the case where O is the name of an object. However, the r4.1 compiler disallows this construct in some cases due to an error. Only in these cases, it is recommended to use the O._struc construct.

The attribute _new returns a new object, that is qualified exactly as THIS(object). This new object is default initialized.

Objects of simple types

All patterns except char, integer, real, boolean, true and false are *subpatterns:* of Object. To enable handling integers, reals, characters and booleans like any other objects in the system, the patterns charObject, integerObject, realObject, booleanObject, trueObject, and falseObject have been introduced. They are genuine patterns, corresponding to the basic patterns, described above. They are specializations of Object that can be used instead of the basic patterns. They have all properties ordinary patterns have (in contrast to the basic patterns).

2.4.2 The charValue, integerValue, booleanValue and realValue Patterns

The `charValue`, `integerValue`, `booleanValue`, and `realValue` patterns are object oriented variants of the basic patterns (i.e. built-in patterns). They are used in all cases where a value of the given type is needed. E.g. a `booleanValue` may be used as *superpattern:* for patterns, returning boolean results.

2.4.3 The charObject, integerObject, booleanObject, trueObject, falseObject and realObject Patterns

These patterns are *subpatterns:* of the `charValue`, `integerValue`, etc. patterns above, and gives the final functionality to allow these variants of the basic patterns to be used a genuine patterns.

These patterns can for instance be utilized when programming general data structures. Consider a data structure `list` which defines its element type as a virtual `Object`.

```
list:
  (# element :< object;
     insert: (# e: ^element enter e[] do ... #)
     remove: (# e: ^element do ... exit e[] #)
  #)
```

When `list` is applied in a specific application, the element type is bound. The following is an example of how a list of integers and a list of editors can be declared and manipulated:

```
(# integerList: @list(# element::< integerObject #);
   editor:
     (# window: ...
        menus: ...
        cut: ...
        copy: ...
        paste: ...
     #);
   ed: ^editor;
   editorList: @list(# element::< editor #);
   io: ^integerObject;
do &integerObject[] -> io[]; 7 -> io;
   io[] -> integerList.insert;
   ...
   editorList.remove -> ed[];
   ...
#)
```

2.5 The Streams Patterns

A `stream` is a generalization of internal and external text objects. An internal text object (`text`) is a sequence (repetition) of chars. An external text object (`file`) corresponds to a traditional text file. `Stream`, `text` and `file` are organized in the following hierarchy:

Stream, text, and file

```
stream: (# ... #);
  text: stream(# ... #);
  file: stream(# ... #);    (* described in a later chapter *)
    unixFile: file(# ... #);(* described in a later chapter *)
    macFile: file(# ... #); (* described in the Macintosh
                             * Library manual *)
```

2.5.1 The Stream Pattern

The `stream` pattern is an abstract *superpattern:* which provides general stream manipulating procedure patterns: `getPos`, `setPos`, `eos`, `length`, `reset`, `newline`, `put`, `get`, `peek`, `putint`, `getint`, `puttext`, `gettext`, `putline`, `getNonBlank`, `getline`, `getAtom`, `scan`, `scanBlanks`, `scanToNL` and `scanAtom`. The `stream` pattern also defines exception patterns (e.g.`EOSerror`).

2.5.1.1 Substreams

substreams.bet

There is an additional library called `substreams.bet`, which implements a substrems concept (and a subtext concept). A `substream` refers to a (consequtive) portion of another `stream`. Manipulations on the substream will thereby actually change that portion of this other stream. All usual stream operations applies to a substream. The interface af substream is not included in this manual.

2.5.2 The Text Pattern

Small texts

The text concept is intended for “small” texts, but there is no size limit. Some of the operations might however be inefficient on large text objects.

A `text` is a sequence of characters. The range of a text object `T` is `[1,T.length]`. A text can be initialized by executing `T.clear` or by assigning it with another (initialized) text. Like the predefined patterns `integer`, `real`, `char` and `boolean`, `Text` objects are self-assignable. A text constant has the form `'foo' or 'a'`. The `'` character may by specified as part of a text constant by repeating it, e.g. `'is''s like this'` is the text constant:`is's like this`.

Besides defining the implementations of the abstract `stream` operations (e.g. `put` and `get`), the `text` pattern defines the following patterns: `empty`, `clear`, `inxGet`, `inxPut`, `append`, `prepend`, `scanAll`, `sub`, `insert`, `delete`, `equal`, `equalNCS`, `less`, `greater`, `makeLC`, `makeUC`, `find, findAll`, `findText, findTextAll`, `copy`, and `asInt`.

All error messages from exceptions originating from `text` objects are followed by the text lines:

```
Error in text which begins as follows:
<THIS(text)>.....
```

where `<THIS(text)>` is the text where the error occurred.

2.5.2.1 Text utilities

TextUtils.bet

The library `textUtils.bet` contains a number of additional text attributes, such as `getBoolean`, `putBoolean`, `set`, `setText`, `setInt`, `setBased`, `setReal, and setBoolean`. The interface file is not included in the manual.

Texthash.bet

The library `texthash.bet` contains a single operation: `honeyman`, which is an efficient and nearly optimal hash-function for text hash keys.

2.5.2.2 Using Text

This example gives examples of how to use the `text` pattern. The `text` object `Records` consists of a sequence of records. Each record has the form:

```
name Job:aJob Salary:aSalary /
```

The program shows various patterns for manipulating the `Records` text:

textRecords.bet

```
ORIGIN '~beta/basiclib/v1.6/betaenv';

--- program: descriptor --
(* Demo example shwoing examples of how to use the text concept from betaenv.
 * The text object Records, consists of a sequence of records. Each record
 * has the form:
 *    name Job:aJob Salary:aSalary /
 * The program shows various patterns for manipulating the Records text.
 *)
(# GetName: (* read next name from T *)
     (# T: ^text; T1: @text
     enter T[]
     do (* scan and skip until a letter is met *)
        T.scan(# while::<(#do NOT (ch->Ascii.isLetter)->value #)#);
        (* scan and read while letters in T *)
        T.scan
        (# while::<(#do ch->Ascii.isLetter->value #)
        do ch->T1.put
        #)
     exit T1
     #);
   GetRecord:
     (* Get the record with the name N  and return name and dat part *)
     (# N: ^Text; name,data: @Text
     enter N[]
     do Records.reset;
        FindName:
          (if not Records.eos then
              Records[]->getName->name;
              data.clear;
              (* scan and read until '/' is met *)
              Records.scan(# while::<(#do (ch<>'/')->value#) do ch->data.put #);
            (if not (N[]->name.equal) then restart FindName
        if)if)
     exit(name,data)
     #);
   GetJobAndSalary:
     (* get the  job and salary from data part, which is the part after name *)
     (# Data: ^Text; Job: @Text; Salary: @integer
     enter Data[]
     do Data.reset;
        Data[]->GetName; Data.get (* skip ':' *); Data[]->GetName->Job;
        Data[]->GetName; Data.get; Data.GetInt->Salary
     exit(Job,Salary)
     #);

   Records: @Text;
do '----------1:'->putLine;
   (*  initialize Records *)
   'John Job:Programmer salary:120000 / '->Records.append;
   'Joan Job:Doctor salary:130000 / '->Records.append;
   'Mary Job:Boss salary:140000 / '->Records.append;
   Records[]->putLine;

   '----------2:'->putLine;
   (* split Records into atoms *)
   Records.reset;
   scan:
   cycle
     (#
     do Records.getAtom->putline;
        (if Records.eos then leave scan
     if)#);
   '---------3:'->putLine;
   (* Find record with name Joan and decode data part *)
   (# Name,Data,Job: @ text; Salary: @Integer
   do 'Joan'->GetRecord->(Name,Data);
      'Ms. '->Name.prePend;
      ' II'->name.append;
      Name[]->putText; ' has the data: '->putText;
      Data[]->GetJobAndSalary->(Job,Salary);
      'Job='->putText; Job.makeUC;  Job[]->putText;
      ' Salary='->putText; Salary->putInt; newline;
   #)

#)
```

2.6 Exceptions and Program Termination

Pattern exception

The pattern `exception` is used as a *superpattern:* for all exceptions in the system. The default action of an exception is to stop the program execution and print an informative error message on the stream `screen`. In addition, the file `<programname>.dump` contains a dump of the call stack. `Exception` uses the pattern `Stop` for termination. Specific error messages can be defined by specializing the `exception` pattern. The attribute `msg` of `exception` is a `text` object that is used to accumulate error messages in the classification hierarchy of exceptions. If the programmer wishes to prevent the program execution from being stopped in order to handle the exception himself, the boolean attribute `continue` of `exception` must be set to `true`.

The exceptions are often defined as virtual procedure patterns of other patterns (such as the `file` pattern, discussed below). At the appropriate levels in the pattern hierarchy, the virtual patterns are bound so that the error messages are tailored to the specific context. The user can augment these error messages by means of the `msg` text object or choose to ignore the exception and continue execution.

In order to differentiate between potential fatal exceptions and more harmless exceptions, the `notification` pattern is defined as:

```
notification : exception(# do true->continue; INNER #);
```

2.6.1 Examples Using Exception

In order to illustrate the use of exceptions, let us return to the previous file exception example. Without using the exception handling facilities an attempt to open a non-existing file will produce the following error messages:

```
**** Exception processing
Error in file 'in.bet'
```

```
No such file
```

Now let us see what can be done by using exceptions.

The binding of `noSpaceError` shows that a message can be added to `msg`. `Msg` could also have been overwritten, by first clearing `msg` (`msg.clear`). The binding of `noSuchFileError` shows how to prevent the system from stopping the execution when the program attempts to open a non-existing file. Instead the user is prompted for another file name. In fact there exists a procedure pattern (`exists`) that tests for the existence of a file, but this has not been used in this example.

```
(# outFile: @file
     (# noSpaceError::
          (# do 'It is time to delete garbage!'->msg.putline #)#);
   inFile: @file
     (# noSuchFileError:: (# do true->continue; false->OK #)#);
   OK: @ boolean;

do 'in.bet' -> inFile.name;
   true -> OK;
   openFile:
   (#
   do inFile.openRead;
      (if not OK then
         'File does not exist!' ->  screen.putline;
         'Type input file name: ' ->  screen.puttext;
         inFile.readFileName;
         true -> OK;
         restart openFile
   if)#);

   'out.bet' -> outFile.name;
   outFile.openWrite;
   readFile:
   (#
   do (if not inFile.eos then
      false -> inFile.gettext -> outFile.puttext;
      outFile.newline;
      restart readFile
    else leave readFile
   if)#);
   inFile.close;
   outFile.close;
#)
```

In case of disk space exhausted, the following message will be printed on the screen before the program execution is stopped:

```
**** Exception processing
Error in file 'in.bet'
File system is full
It is time to delete garbage!
```

The first line is from the general pattern `exception`, the second and the third lines are from the binding of `noSpaceError` in `file` and the fourth line is from the binding above, i.e. at the user level.

An attempt to open a non-existing file will produce the following error messages:

```
File does not exist!
Type input file name:
```

It gives the programmer the possibility to proceed with another file name.

2.7 Various Other Patterns

2.7.1 Control Patterns

`Betaenv` contains three predefined control patterns: `forTo`, `cycle` and `loop`. They are respectively defined as:

```
forTo: (* for inx in [min,max] do inner *)
  (# min,max,inx: @integer enter (min,max) do ... inner; ... #);

cycle: (* executes inner forever *)
  (# do l:(#... inner; ...#) #);

loop: (* control pattern for while- and repeat-loop *)
  (# while:< booleanValue(# do ... inner; ... #);
     until:< booleanValue(# do ... inner; ... #);
  do ... inner; ...
  #)
```

Recall that control patterns are procedure patterns that are to be used as *superpatterns*. The first example illustrates the use of `forTo` in an inserted item:

```
do ...
   (3,17) -> forTo(# do inx*inx -> putint; newline #);
   ...
```

It will cause printing of the values 3^2, 4^2, ..., 17^2. The next example illustrates the use of `cycle` in the definition of another control pattern.

```
countCycle: cycle
(* increments inx and executes inner forever *)
(# inx: @integer
do inx + 1 -> inx;
   inner
#);
```

Finally, the following `loop` example reads a sequence of integers from standard input until either a non-positive integer is read or the sum of integers exceeds 1000:

```
loop(# while::< (# do getint->i; i>0->value #);
       until::< (# do sum>1000->value #);
       sum, i: @integer
     do i+sum->sum
     #)
```

2.7.2 Input/Output Patterns

Screen and keyboard

Standard input/output is available through dynamic references to objects that are instances of the pattern `stream` (see later). These streams are automatically opened. Abbreviations for the most often used input/output operations are defined (e.g. `put` for `screen.put` and `get` for `keyboard.get`).

```
(# j: @integer
do (for i: 5 repeat
     '? '-> puttext; getint -> j;
     j*j -> putint; newline
   for)
#)
```

Single character input - iget.bet

The `iget.bet` library implements immediate character input (i.e. non-buffered input). This is through the `iget` operation, which returns a single character. `iget` returns as soon as a character is typed.

2.7.3 Command Line Arguments

It is possible to let BETA programs access the command line arguments trough the `noOfArguments` and `arguments` patterns. `NoOfArguments` returns the number of text atoms on the command line, including the program name. The text atoms are numbered from `1` to `noOfArguments` . The program name is obtained by `1 -> arguments` , the first argument is obtained by `2 -> arguments` , etc.

The following example displays the number of arguments of a command line followed by the arguments.

```
(# do
   'This program was called with ' -> puttext;
   noOfArguments -> putint; ' argument(s):' -> putline;
   (for i: noOfArguments repeat
     i -> arguments -> puttext; ' ' -> put;
   for);
   newline;
#)
```

2.7.4 Object Pool

The `objectPool` is for keeping track of unique instances of patterns. A call of the form

```
objectPool .get(# type::< T #) -> obj[]
```

will return an instance of `T`. The first call will create an instance of `T`. Subsequent calls will return this instance again. The `objectPool` is useful in systems where many fragments must refer to the same unique instance of a pattern `T`. `ObjectPool` also defines a `scan` operation which may be used for scanning the objects in the pool. For more operations, please consult the interface descriptions later.

2.7.5 Concurrency

systemEnv

The current version of the Mjølner System includes an experimental implementation of concurrency. The complete environment for concurrency is defined in the `systemEnv` library, described in a later chapter. However, some concurrency facilities are necessarily defined in the `betaenv` library. Please refer to the later chapter on the `systemEnv` library for details.

2.7.6 External Language Interface

External and cStruct

Betaenv contains the `shortInt` , `external` and `cStruct` patterns for interfacing into facilities written in other languages, such as `C` and `Pascal`. Please refer to the later chapter on the `external` library for details.

2.8 Time and Date Library

The basic libraries also contains a library: `timedate.bet` , which defines a series of patterns for manipulating time and date informations. The interface of `timedate.bet` is not further described in this manual.

timedate.bet

2.9 Interface Description for the betaenv Library

```
BODY 'private/betaenvbody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1984-96
 *       All rights reserved.
 *
 * This fragment implements the very basic patterns, utilized by most
 * BETA programs
 *)
-- betaenv: descriptor --
(# <<SLOT lib: attributes>>;
   (****************************************************************)
   (* The simple patterns for simple values and variables.  These
    * simple patterns are treated special by the compiler.
    *)
   integer: (* 32 bit signed long *) (# #);
   shortInt: (* 16 bit unsigned half *) (# #);
   char: (* 8 bit unsigned byte *) (# #);
   boolean: (* 8 bit unsigned byte, values 0 or 1 *) (# #);
   false: boolean (* 8 bit unsigned byte with value 0 *) (# #);
   true: boolean (* 8 bit unsigned byte with value 1 *) (# #);
   real: (* double precision floating point number *) (# #);

   object: (* General  superpattern: *)
     (# _struc:
          (* Exit a pattern reference for THIS(Object).
           * Is now obsolete: the new form obj## is preferred
           * to the old form obj.struc
           *)
          (#
          exit this(object)##
          #);
        _new:
          (* returns a new object, that is qualified exactly
           * as THIS(object)
           *)
          (# newObj: ^object; oType: ##object
          do this(object)##->oType##; &oType[]->newObj[]; INNER _new
          exit newObj[]
          #)
     do INNER object
     #);
   (* The following patterns define 'real' patterns corresponding to
    * the predefined simple patterns
    *)
   integerValue: (# value: @integer do INNER integerValue exit value #);
   integerObject: integerValue(# enter value do INNER integerObject #);
   charValue: (# value: @char do INNER charValue exit value #);
   charObject: charValue(# enter value do INNER charObject #);
   booleanValue: (# value: @boolean do INNER booleanValue exit value #);
   booleanObject: booleanValue(# enter value do INNER booleanObject #);
   trueObject: booleanObject(# do true->value; INNER trueObject #);
   falseObject: booleanObject(# do INNER falseObject #);
   realValue: (# value: @real do INNER realValue exit value #);
   realObject: realValue(# enter value do INNER realObject #);

   (****  Implementation dependent constants ************************)
   maxInt: integerValue(# ... #);
   minInt: integervalue(# ... #);
   maxReal: realValue(# ... #);
   minReal: realValue(# ... #);
   infReal: (* Returns the real value 'Infinity' *)
     realValue(# ... #);

   (*****  Functional patterns **************************************)
   min: (* Returns the minimum of 2 integers *)
     (# a,b: @integer
     enter (a,b)
     do (if (a < b) then a->b if)
     exit b
     #);
   max: (* Returns the maximum of 2 integers *)
     (# a,b: @integer
     enter (a,b)
     do (if (a < b) then b->a if)
     exit a
     #);
   abs: (* Returns the absolute value of an integer *)
     (# n: @integer
     enter n
     do (if (n < 0) then -n->n if)
     exit n
     #);

   (*****  Simple standard input/output patterns ********************)
   keyboard, screen: ^stream;
   get:
     (# ch: @char; getC: @keyboard.get do getC->ch; INNER get exit ch #);
   put:
     (# ch: @char; putC: @screen.put enter ch do INNER put; ch->putC #);
```

```
newline:
  screen.newline(# #);
putint:
  screen.putint(# do INNER putint #);
getint:
  keyBoard.getint(# do INNER getint #);
puttext:
  (# t: ^text; putT: @screen.puttext
  enter t[]
  do INNER puttext; t[]->putT
  #);
putline:
  screen.putline(# do INNER putline #);
getNonBlank:
  keyBoard.getNonBlank(# do INNER getNonBlank #);
scanAtom:
  keyBoard.scanAtom(# do INNER scanAtom #);
getAtom:
  (# t: ^text; getA: @keyBoard.getAtom
  do getA->t[]; INNER getAtom
  exit t[]
  #);
getline:
  (# t: ^text; getL: @keyBoard.getline
  do getL->t[]; INNER getline
  exit t[]
  #);

(*****  Control patterns *****************************************)
forTo: (* for 'inx' in [low:high] do INNER forTo *)
  (# low, high, inx: @integer;
  enter (low, high)
  ...
  #);
cycle: (* Executes INNER forever *)
  (# ... #);
loop:
  (# while:< booleanValue(# do true->value; INNER while #);
     until:< booleanValue;
     whilecondition: @while;
     untilcondition: @until;
  ...
  #);
qua:
  (* Pattern replacing the BETA language construct QUA.  To be
   * used as 't1[]->qua(# as::< Tn #)->t2[]'.  The 'qua' pattern
   * checks, whether 't1' is qualified by 'Tn'.  If not, the
   * 'quaError' exception is invoked.  Otherwise, a reference
   * qualified by 'Tn', and referring to the same object as 't1[]'
   * is referring, is returned.
   *)
  (# as:< object;  R: ^object;  thisObj: ^as;
     quaError:< exception
       (# do 'Qualification error'->msg.append; INNER quaError #)
  enter R[]
  ...
```

```
     exit thisObj[]
     #);

   (*****  Stream patterns *******************************************)
   stream:
     (# <<SLOT streamLib: attributes>>;
        length:< integerValue (* returns the length of THIS(stream) *)
          (#
          do -1->value; INNER length
          #);
        position: (* current position of THIS(stream) *)
          (#
          enter setPos
          exit getPos
          #);
        eos:< (* returns 'true' if THIS(stream) is at end-of-stream *)
          booleanValue;
        reset: (* sets 'position' to zero *)
          (#
          do 0->setPos
          exit THIS(stream)[]
          #);
        peek:< (* looks at the next character of THIS(stream) *)
          (# ch: @char
          do INNER peek
          exit ch
          #);
        get:< (* reads a character from THIS(stream) *)
          (# ch: @char
          do INNER get
          exit ch
          #);
        getNonBlank:
          (* Reads first non-whitespace character from THIS(stream).
           * If called at end-of-stream the character 'ascii.fs' is
           * returned
           *)
          (# ch: @char;
             skipblanks: @scanWhiteSpace;
             testEOS: @EOS;
             getCh: @get;
          ...
          exit ch
          #);
        getint: integerValue
          (* Reads an integer: skips whitespace characters and
           * returns the following digits.
           *
           * See numberio.bet for more numerical output operations
           *)
          (# syntaxError:< streamException
               (#
               do 'getint: syntax error - looking at: "'->msg.append;
                  peek->msg.put; '"'->msg.putline; INNER syntaxError
               #);
             geti: @...
          do geti; INNER getint
          #);
        getAtom:<
          (* Returns the next atom (i.e. sequence of non-white
           * characters - skipping leading blanks)
           *)
          (# txt: ^text;
          do &text[]->txt[]; INNER getAtom;
          exit txt[]
          #);
        getline:<
          (* Reads a sequence of characters until nl-character
           * appears and returns the characters read.
           *)
          (# txt: ^text;
          do &text[]->txt[]; INNER getline
          exit txt[]
          #);
        asInt:
          (* converts THIS(text) to an integer value, ignoring
           * leading and trailing whitespace.  See numberio.bet for
           * more numerical conversion operations.
           *)
          (# i: @integer;
             syntaxError:< streamException
               (# peekCh: @char
               enter peekCh
               do 'asInt: syntax error - looking at: "'->msg.append;
                  peekCh->msg.put; '"'->msg.put;
                  INNER syntaxError
               #)
          ...
          exit i
          #);
        put:< (* writes a character to THIS(stream) *)
          (# ch: @char
          enter ch
          do INNER put
          exit THIS(stream)[]
          #);
        newline: (* writes the nl-character *)
          (#
          do ascii.newline->put
          exit THIS(stream)[]
          #);
        putint:
          (* Writes an integer to THIS(stream); The format may be
           * controlled by the 'signed', 'blankSign', 'width',
           * 'adjustLeft' and 'zeroPadding' variable attributes.
           * 'width' is extended if it is too small.  Examples:
           * '10->putint' yields: '10'; '10*pi->putint(# do 10->width;
           * true->adjustLeft #)' yields: '10 '; and '10->putint(# do *
10->width; true->zeroPadding #)' yields: '0000000010'.
           *
           * See numberio.bet for more numerical output operations
           *)
        (#  n: @integer;
            signed: @boolean
             (* If integer is positive, a '+' will always be
              * displayed
              *);
            blankSign: @boolean
             (* If integer is positive, a ' ' space is displayed as
              * the sign.  Ignored if 'signed=true'
              *);
            width: @integer
             (* Minimum width *);
            adjustLeft: @boolean
             (* Specifies if the number is to be aligned left or
              * right, if padding of spaces is necessary to fill up
              * the specified width.
              *);
            zeroPadding: @boolean
             (* width is padded with leading zero instead of
              * spaces.  Ignored if 'adjustLeft=true'
              *);
            format:< (# do INNER format #);
            puti: @...
        enter n
        do 1->width; format; INNER putint; puti
        exit THIS(stream)[]
        #);
      puttext:< (* Writes a text to THIS(stream). *)
        (# txt: ^text
        enter txt[]
        do (if txt[]<>NONE then INNER puttext if)
        exit THIS(stream)[]
        #);
      putline:
        (* 'puttext' followed by 'newline' *)
        (# T: ^text; putT: @puttext; newL: @newline
        enter T[]
        do INNER putline; T[]->putT; newL
        exit THIS(stream)[]
        #);
      scan:
        (* Scan chars from current position in THIS(stream) while
         * '(ch->while)=true'; perform INNER for each char being
         * scanned
         *)
        (#  while:<
             (# ch: @char; value: @boolean
             enter ch
             do true->value; INNER while
             exit value
             #);
            ch: @char;
            whilecondition: @while;
            testEOS: @EOS;
            getPeek: @peek;
            getCh: @get;
        ...
```

```
          exit THIS(stream)[]
          #);
        scanWhiteSpace: scan
          (* Scan whitespace characters *)
          (# while::< (# do ch->ascii.isWhiteSpace->value #);
          do INNER scanWhiteSpace
          exit THIS(stream)[]
          #);
        scanAtom:
          (* Scan until first non-whitespace char.  Scan the next
           * sequence of non-whitespace chars.  Stop at first
           * whitespace char.  For each non-whitespace char an INNER
           * is performed. Usage: 'scanAtom(# do ch-><destination> #)'
           *)
          (# ch: @char;
          ...
          exit THIS(stream)[]
          #);
        scanToNl:
          (* Scan all chars in current line including newline char *)
          (# ch: @char; getCh: @get;
          ...
          exit THIS(stream)[]
          #);
        streamException: exception
          (# do INNER streamException #);
        EOSerror:< streamException
          (* Raised from 'get' and 'peek' when attempted to read past
           * the end of the stream.
           *)
          (#
          do 'Attempt to read past end-of-stream'->msg.putline;
             INNER EOSerror
          #);
        otherError:< streamException
          (* Raised when some other kind of stream error apart from
           * the one mentioned above occurs.
           *);
        getPos:< (* returns current position of THIS(Stream) *)
          integerValue;
        setPos:< (* sets current position in THIS(stream) to 'p' *)
          (# p: @integer
          enter p
          do INNER setPos
          exit THIS(stream)[]
          #)
     #); (* pattern stream *)

  (*****  Text pattern ********************************************)
  text: stream
    (* A text is a sequence of characters.  Let 'T: @text'. The
     * range of 'T' is '[1,T.length]'.  A text can be initialized by
     * executing 'T.clear' or by assigning it another (initialized)
     * text.  A text-constant has the form 'foo'.  The 'text' pattern
     * is primarily intended for small texts but there is no upper
     * limit in the size. However, most of the operations becomes
```

```
 * less efficient with larger texts.
 *)
(# <<SLOT textLib: attributes>>;
   length::< (* Returns the length of THIS(text) *)
     (# do lgth->value; INNER length #);
   eos::<(# ... #);
   empty:
     (# exit (lgth = 0) #);
   clear: (* Sets the length and position of THIS(text) to zero *)
     (#
     do 0->pos->lgth
     exit THIS(text)[]
     #);
   equal: booleanValue
     (* Tests if THIS(text) is equal to the entered text.  If
      * 'NCS' is further bound to 'trueObject', the comparison
      * will be done Non Case Sensitive.
      *)
     (# txt: ^text;
        NCS:< booleanObject
     enter txt[]
     ...
     #);
   equalNCS: equal
     (* As 'equal', except the the comparison will be done Non
      * Case Sensitive
      *)
     (# NCS:: trueObject #);
   less: booleanValue
     (* Tests whether the entered text 'T1[1: length]' is less
      * than 'THIS(text)[1: T1.length]'.  The lexicographical
      * ordering is used.
      *)
     (# T1: ^text
     enter T1[]
     ...
     #);
   greater: booleanValue
     (* Tests whether the entered text 'T1[1: length]' is
      * greater than 'THIS(text)[1: T1.length]'.  The
      * lexicographical ordering is used.
      *)
     (# T1: ^text
     enter T1[]
     ...
     #);
   peek::<
     (* Returns the character at current position; does not
      * update 'position'
      *)
     (# ... #);
   get::<
     (* Returns the character at current position; increments
      * 'position'
      *)
     (# ... #);
```

```
inxGet: charValue
  (* Returns the character at position 'i' *)
  (# i: @integer;
     iget: @...
  enter i
  do iget
  #);
getAtom::<
  (* Returns the next atom (i.e. sequence of non-white
   * characters - skipping leading blanks)
   *)
  (# ... #);
getline::<
  (* Reads a sequence of characters until nl-character
   * appears and returns the characters read.
   *)
  (# ... #);
put::<
  (* writes the character 'ch' at current position in
   * THIS(text); increments 'position'
   *)
  (# ... #);
inxPut:
  (* Replaces the character at position 'i' *)
  (# ch: @char;
     i: @integer;
     iput: @...
  enter (ch,i)
  do iput
  exit THIS(text)[]
  #);
puttext::<(# ... #);
append:
  (* Appends a text to THIS(text); does not change 'position'
   *)
  (# T1: ^text
  enter T1[]
  ...
  exit THIS(text)[]
  #);
prepend:
  (* Inserts the text in 'T1' in front of THIS(text); updates
   * current position to 'position+T1.length' if 'position>0'
   *)
  (# T1: ^text
  enter T1[]
  ...
  exit THIS(text)[]
  #);
insert:
  (* Inserts a text before the character at position 'inx'.
   * Note: inx<1 means inx=1; inx>length means inx=length+1.
   * If 'position>=inx' then 'position+T1.length->position'.
   *)
  (# T1: ^text;
     inx: @integer
  enter (T1[],inx)
  ...
  exit THIS(text)[]
  #);
delete:
  (* Deletes THIS(text)[i: j]; updates current position:
   *      i<=position<j => i-1->position
   *      j<=position   => position-(j-i+1)->position
   *)
  (# i,j: @integer;
     deleteT: @...
  enter (i,j)
  do deleteT
  exit THIS(text)[]
  #);
makeLC: (* Converts all characters to lower case *)
  (# ...
  exit THIS(text)[]
  #);
makeUC:
  (* Converts all characters to upper case *)
  (# ...
  exit THIS(text)[]
  #);
sub:
  (* Returns a copy of THIS(text)[i:j].  If 'i<1', 'i' is
   * adjusted to 1. If 'j>length', 'j' is adjusted to
   * 'length'.  If (after adjustment) 'i>j', an empty text is
   * returned.
   *)
  (# i,j: @integer; T1: ^text;
     subI: @...
  enter (i,j)
  do subI
  exit T1[]
  #);
copy:
  (# T1: ^text;
     copyI: @...
  do copyI
  exit T1[]
  #);
scanAll:
  (* Scans all the elements in THIS(text).  For 'ch' in '[1:
   * THIS(text).length]' do INNER
   *)
  (# ch: @char
  do (for i: lgth repeat T[i]->ch; INNER scanAll for)
  exit THIS(text)[]
  #);
find:
  (* find all occurrences of the character 'ch' in
   * THIS(text), executing INNER for each occurrence found,
   * beginning at 'THIS(text).position'.  'inx' will contain
   * the position of each 'ch' in THIS(text).  If 'NCS' is
   * further bound to 'trueObject', the comparison will be
```

```
          * done Non Case Sensitive.  If 'from' is further bound, the
          * search will begin at position 'from'.
          *)
         (# ch: @char;
            inx: @integer;
            NCS:< booleanObject;
            from:< integerObject(# do pos->value; INNER from #)
         enter ch
         ...
         exit THIS(text)[]
         #);
       findAll: find
         (* As 'find', except that the entire text will be searched.
          * Replaces 'findCh' in previous versions of betaenv (v1.4
          * and earlier)
          *)
         (# from:: (# do 0->value #)
         do INNER findAll
         #);
       findText:
         (* find all occurrences of the 'txt' in THIS(text),
          * executing INNER for each occurrence found, beginning at
          * 'THIS(text).position'.  'inx' will contain the position
          * of the first character of each occurrence found
          * THIS(text).  If 'NCS' is further bound to 'trueObject',
          * the comparison will be done Non Case Sensitive.  If
          * 'from' is further bound, the search will begin at
          * position 'from'.
          *)
         (# txt: ^text;
            inx: @integer;
            NCS:< booleanObject;
            from:< integerObject(# do pos->value; INNER from #)
         enter txt[]
         ...
         exit THIS(text)[]
         #);
       findTextAll: findText
         (* As 'findText', except that the entire text will be
          * searched
          *)
         (# from:: (# do 0->value #)
         do INNER findTextAll
         #);
       extend:
         (* Extend THIS(text) with 'L' (undefined) chars. Notice
          * that it is only the representation of the THIS(text),
          * that is extended, the 'length' and 'position' are not
          * changed.
          *)
         (# L: @integer
         enter L do L->T.extend
         exit THIS(text)[]
         #);
       indexError:< streamException
         (* Raised from 'Check' when the index goes outside the
        * range of the text. Message: "Index error in text!".
        *)
       (# inx: @integer
       enter inx
       ...
       #);
     EOSerror::<
       (* Raised from 'get' and 'peek' when the end of the stream is
        * passed.
        *)
       (# ... #);
     otherError::<
       (* Raised when an error other than the Index-/EOSerror
        * occurs.
        *)
       (# ... #);
     setPos::<(# ... #);
     getPos::<(# do pos->value; INNER getPos #);
     (* Private attributes: !!OBS!! The 3 attributes 'T', 'lgth'
      * and 'pos' declared below MUST be the first data items
      * declared in 'stream' and 'text' since their addresses are
      * hardcoded into the compiler.
      *)
     T: [16] @char;
     lgth,pos: (* 16 is default size *) @integer;
     setT: (# enter T do T.range->lgth->pos #)
  enter setT
  exit T[1: lgth]
  #) (* Pattern text *);
(*****  ASCII character constants and attributes *****************)
ascii: @
  (# <<SLOT asciiLib: attributes>>;
     nul: (# exit 0 #);
     soh: (# exit 1 #);
     stx: (# exit 2 #);
     etx: (# exit 3 #);
     eot: (# exit 4 #);
     enq: (# exit 5 #);
     ack: (# exit 6 #);
     bel: (# exit 7 #);
     bs: (# exit 8 #);
     ht: (# exit 9 #);
     nl: (# exit 10 #);
     vt: (# exit 11 #);
     np: (# exit 12 #);
     cr: (# exit 13 #);
     so: (# exit 14 #);
     si: (# exit 15 #);
     dle: (# exit 16 #);
     dc1: (# exit 17 #);
     dc2: (# exit 18 #);
     dc3: (# exit 19 #);
     dc4: (# exit 20 #);
     nak: (# exit 21 #);
     syn: (# exit 22 #);
```

```
         etb: (# exit 23 #);
         can: (# exit 24 #);
         em: (# exit 25 #);
         sub: (# exit 26 #);
         esc: (# exit 27 #);
         fs: (# exit 28 #);
         gs: (# exit 29 #);
         rs: (# exit 30 #);
         us: (# exit 31 #);
         sp: (# exit 32 #);
         capA: (# exit 65 #);
         smalla: (# exit 97 #);
         del: (# exit 127 #);
         newline: @char; (* either 'lf' or 'cr' *)

         init: ...;
         upCase: @charObject
           (# ... #);
         lowCase: @charObject
           (# ... #);
         testChar: booleanValue
           (# ch: @char
           enter ch
           do INNER testchar
           #);
         isUpper: @testChar
           (# ... #);
         isLower: @testChar
           (# ... #);
         isDigit: @testChar
           (# ... #);
         isLetter: @testChar
           (# ... #);
         isSpace: @testChar
           (* True if 'ch' in sp,cr,nl,np,ht,vt *)
           (# ... #);
         isWhiteSpace: @testChar
           (* True if 'ch' is a whitespace char *)
           (# ... #);
         private: @...
     #);
   (*****  Exception Patterns ****************************************)
   stop:
     (* Terminates program execution: 'termCode=normal': normal
      * termination; 'termCode=failure': abnormal termination;
      * 'termCode=failureTrace': abnormal termination with trace of
      * run-time stack on dump-file; 'T' will be printed on the
      * screen.
      *)
     (# termCode: @integer; T: ^text
     enter (termCode,T[])
     do ...
     #);
   normal: (# exit 0 #);
   failure: (# exit -1 #);
```

```
failureTrace: (# exit -2 #);
exception:
  (# <<SLOT exceptionLib: attributes>>;
      msg:
        (* append text to this 'msg' vatiable to specify the
         * exception error message for this(exception)
         *)
        @text;
      continue: @boolean
        (* the valur of this variable determines the control-flow
         * behaviour of this(exception):
         *    true:  continue execution after exception
         *    false: terminate execution by calling 'stop'; default
         *);
      propagate:<
        (* if further bound to trueObject, this(exception) allows
         * propagation (i.e. this(exception will _not_ terminate)
         *)
        booleanValue;
      error:
        (* used to define local exception conditions which can be
         * handled separately.  All 'error's that are not handled
         * separately will be handled by this(exception)
         *)
        (# <<SLOT errorLib: attributes>>
        do false->continue;
           INNER;
           '**** Error processing'->msg.prepend;
           (if not propagate and not continue then this(exception) if)
        exit propagate
        #);
      notify: error
        (* used to define local notification conditions which can be
         * handled separately.  All 'notify's that are not handled
         * separately will be handled by this(exception)
         *)
        (# do true->continue; INNER #);
      termCode: @integer
        (* Arg. to pattern 'stop'; initial failureTrace *);
  do failureTrace->termCode;
     INNER exception;
     (if not continue and not propagate then
         '**** Exception processing'->msg.prepend;
         (termCode,msg[])->stop
     if)
  #);
notification: exception
  (# do true->continue; INNER notification #);
(*****  Object Pool ********************************************)
objectPool: @
  (# <<SLOT objectPoolLib: attributes>>;
      get:
        (# type:< object;
           obj: ^type;
           exact:< booleanValue;
           init:< object(* Called if an object was created *)
```

```
          ...
          exit obj[]
          #);
        strucGet:
          (# type: ##object;
             obj: ^object;
             exact:< booleanValue;
             init:< object(* Called if an object was created *);
          enter type##
          ...
          exit obj[]
          #);
        scan:
          (* Scan through all objects in 'objectPool', (at least)
           * qualified by 'type'.
           *)
          (# type:< object;
             current: ^type;
             exact:< booleanValue;
          ...
          #);
        strucScan:
          (* Scan through all objects in 'objectPool', (at least)
           * qualified by 'type'
           *)
          (# type: ##object;
             current: ^object;
             exact:< booleanValue
          enter type##
          ...
          #);
        put:
          (* Puts a given object into 'objectPool'. If an object with
           * (at least) the qualification of the given object is
           * already present in 'objectPool', the exception
           * 'alreadyThere' is raised.
           *)
          (# obj: ^object;
             exact:< booleanValue;
             alreadyThere:< exception;
             putObj: @...
          enter obj[]
          do putObj
          #);
        private: @...;
     #);

   (*****  Command line arguments ************************************)
   noOfArguments: integervalue
     (* Return the number of arguments on command line *)
     (# ... #);
   arguments:
     (* Returns argument number argNo.
      * Number 1 is the program name, number 2 is the first program
      * argument, etc.
      *)
```

```
    (# argNo: @integer; theArg: ^text;
    enter argNo
    ...
    exit theArg[]
    #);
  (*****************************************************************)
  (* External language interface: See file 'external.bet' for further
   * patterns.
   *)
  external:
    (* Is only meaningful with interface to externals *)
    (#  callC, callPascal, pascal, pascalTrap, callStd,
        cExternalEntry, pascalExternalEntry, stdExternalEntry: @text
    #);
  cStruct: (* Super-pattern for describing cStruct-patterns *)
    (# <<SLOT cStructLib: attributes>>;
       R: [(byteSize-1) div 4 + 1] @integer;
       byteSize:< integerObject
         (* 'R' is the bytestream containing THIS(cStruct).  Must be
          * declared as the first attribute
          *);
       chkBounds:< (* Number of bytes in THIS(cStruct) *)
         (# inx: @integer; continue: @boolean
         enter inx
         do (if ((inx < 0) or (R.range*4 < inx)) then
                INNER chkBounds;
                (if not continue then
                    (failureTrace,'Index error in cStruct')->&stop
                if)
            if)
         #)
    #);
  data:
    (* The 'data' pattern may be used for defininin simple data
     * objects.  Data-objects have no 'type' information. They can
     * thus NOT be allocated dynamically in the BETA heap. They do * not
have the overhead of extra attributes used for virtual
     * dispatch and garbage collection. One main use of data-objects
     * is as interface to external data such as 'cstruct'.  For
     * details see the manuals
     *)
    (# #);
  doGC: (* will force a garbage collection to happen *)
    (# ... #);
  machine_type:
    (* Exits a reference to a copy of a text indicating the machine
     * type in lowercase, e.g. 'sun4s', 'linux', 'nti'.
     *)
    (#  T: @Text;
    do machine_typeexternal->T;
    exit T.copy
    #);
  machine_typeexternal: external
    (#  T: [1]@Char; do 'machine_type'->callC; exit T #);
  program: (* descriptor executed by this environment *)
    ...;
   theProgram: ^|program;
   theScheduler: ^|object
     (* Scheduler installed by 'basicSystemEnv' (if used in program) *);
   (*****************************************************************)
   (* The following patterns are only used by the compiler and should
    * NOT be used for other purposes.
    *)
   repetition:
     (# range: (* Returns the range of THIS(repetition) *)
          (# n: @integer
          exit n
          #);
        new:
          (* Allocates a new repetition of 'n' elements. The previous
           * elements in THIS(repetition) become inaccessible
           * hereafter
           *)
          (# n: @integer
          enter n
          #);
        extend:
          (* Extends THIS(repetition) by 'n' elements.  The existing
           * elements are retained.  The new elements are allocated
           * after the existing elements (i.e. with index from the
           * 'range+1')
           *)
          (# n: @integer
          enter n
          #)
     #);
   errorName: (# #)
   (*****************************************************************)
do ...;
   &|program[]->theProgram[];
   theProgram;
   (if theScheduler[]<>NONE then theScheduler if);
   ... ;
#)
```

3. The math Library

This library provides mathematical patterns: trigonometric, hyperbolic, exponential and logarithmic, floating point manipulation, and miscellaneous constants.

3.1 Using the math Fragment

A program using the`math` fragment will have the following structure:

```
INCLUDE '~beta/basiclib/v1.6/math
--- program: descriptor ---
(# ...
   r: @real;
do ...
   1.234 -> sin -> r;
   ...
#)
```

3.2 Interface Description for the math Library

```
ORIGIN 'betaenv';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1984-96
 *       All rights reserved.
 *
 * Math.bet: mathematical functions
 *
 * This library provides mathematical patterns: trigonometric,
 * hyperbolic, exponential and logarithmic, floating point
 * manipulation and miscellaneous constants.
 *)
--- LIB: Attributes ---

(* miscellaneous constants *)

e:       (# Exit 2.7182818284590452354 #);
log2e:   (# Exit 1.4426950408889634074 #);
log10e:  (# Exit 0.43429448190325182765 #);
ln2:     (# Exit 0.69314718055994530942 #);
ln10:    (# Exit 2.30258509299404568402 #);
pi:      (# Exit 3.14159265358979323846 #);
pihalf:  (# Exit 1.57079632679489661923 #);
piforth: (# Exit 0.78539816339744830962 #);

(* trigonometric functions *)

acos: external
  (* returns the arccosine of x in radians, in the range 0 to pi.  If
   * x is a NaN (Not-a-Number) or if the absolute value of x exceeds
   * 1.0, acos(x) returns a NaN.  Invalid operation/DOMAIN error is
   * signaled if x is a NaN or if |x| > 1.0.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
asin: external
  (* returns the arcsine of x in radians, in the range -pi/2 to pi/2.
   * If x is a NaN or if the absolute value of x exceeds 1.0, asin
   * returns a NaN.  Invalid operation/DOMAIN error is signaled if x
   * is a NaN or if |x| > 1.0.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
atan: external
  (* returns the arctangent of x, in the range of -pi/2 to pi/2
   * radians.  If x = +-infinity, then atan(x) returns +-pi/2.  If x
   * is +- 0, then atan returns x. If x is a NaN, then atan returns a
   * NaN.  An invalid operation/DOMAIN error is signaled by atan only
   * if x is a NaN.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
atan2: external
  (* returns the arctangent of y/x in radians, in the range -pi to
   * pi, using the signs of both arguments to determine the quadrant
   * of the return value.
   *
   * If x is a NaN or if y is a NaN or if both x and y are infinities,
   * atan2 returns a NaN.  If both x and y are zero, atan2 returns
   * zero.  Invalid operation/DOMAIN error is signaled by atan2 if
   * both x and y are infinite or if either x or y is a NaN.
   *)
  (# y,x,res: @Real;
  enter (y,x)
  exit res
  #);
cos: external
  (* computes the cosine of x, where x is expressed in radians.
   *
   * The cos function uses an argument reduction based on the
   * remainder function and pi.  The cos function is periodic with
   * respect to pi, so its period differs slightly from its
   * mathematical counterpart and diverges from its counterpart when
   * the argument becomes very large.
   *
   * If x is infinite or a NaN, then cos returns a NaN and signals
   * invalid/DOMAIN error.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
sin: external
  (* computes the sine of x, where x is expressed in radians.
   *
   * The sin function uses an argument reduction based on the
   * remainder function and pi.  The sin function is periodic with
   * respect to pi, so its period differs slightly from its
   * mathematical counterpart and diverges from its counterpart when
   * the argument becomes very large.
   *
   * If x is infinite or a NaN, then sin returns a NaN and signals
   *    invalid/DOMAIN error.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
tan: external
  (* computes the tangent of x, where x is expressed in radians.
   *
   * The tan function uses an argument reduction based on the
   * remainder function and pi The tan function is periodic with
   * respect to pi, so its period differs slightly from its
   * mathematical counterpart and diverges from its counterpart when
   * the argument becomes very large.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);

(* hyperbolic functions *)

cosh: external
  (* returns the hyberbolic cosine of x.
   *
   * If x is a NaN, cosh returns a NaN.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
sinh: external
  (* returns the hyberbolic sine of x.
   *
   * If x is a NaN, sinh returns a NaN.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
tanh: external
  (* returns the hyberbolic tangent of x.
   *
   * If x is a NaN, tanh returns a NaN.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);

(* exponential and logarithmic functions *)

exp: external
  (* returns the base-e or natural exponential e^x.
   *
   * Special cases for exp:
   *
   * If x = +infinity, then exp returns +infinity and does not signal
   * an exception.  If x = -infinity, then exp returns    0 and does not
   * signal an exception.  If x is a NaN, then exp returns a NaN.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
ldexp: external
  (* returns the quantity x * 2^exp. *)
  (# x,res: @Real;
     exp: @Integer;
  enter (x,exp)
  exit res
  #);
log: external
  (* returns the base e or natural logarithm of its argument x.
   *
   * Special cases for log:
   *
   * If x is +infinity, then log returns +infinity and signals no
   * exceptions.  If x is 0, then log returns -infinity and signals
   * divide-by-zero.  If x < 0, then log returns a NaN and signals
   * invalid/DOMAIN error.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
log10: external
  (* returns the base 10 logarithm of x.
   *
   * If x is a NaN or is negative, log10 returns a NaN. If x is
   * +infinity, log10(x) returns +infinity.  If x is zero, log10
   * returns -infinity and signals divide by zero/SING error.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);

(* floating point manipulation *)

modf: external
  (* returns the fractional part of x and stores the integral part
   * indirectly in the location pointed to by ipPtr.  Both the return
   * value and the value stored in ipPtr share the same sign as x.
   *
   * If x is infinite, modf returns a zero with the sign of x and sets
   * ipPtr to x.  If x is a NaN, mod returns a NaN and sets ipPtr to
   * the same NaN.
   *)
  (# x,res: @Real;
     ipPtr: @Integer;
  enter (x,ipPtr)
  exit res
  #);
pow: external
  (* returns x^y *)
  (# x,y,res: @Real;
  enter (x,y)
  exit res
  #);
sqrt: external
  (* computes the square root of x.
   *
   * Special cases for sqrt:
   *
   * If x is a NaN, sqrt returns a NaN and signals no exceptions.  If
   * x is a NaN or if x < 0, sqrt returns a NaN and signals invalid
   * operation/DOMAIN error.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
ceil: external
  (* returns the smallest integer value (in real format) not less
   * than x.
   *
   * If x is a NaN, ceil returns a NaN.  If x is infinite, ceil
   * returns x.  Invalid operation is signaled by ceil if x is a NaN.
   * If x is a non-integral finite value, ceil signals inexact
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
fmin: (* Returns the minimum of 2 reals *)
  (# a,b: @real
  enter (a,b)
  do (if (a < b) then a->b if)
  exit b
  #);
fmax: (* Returns the maximum of 2 reals *)
  (# a,b: @real
  enter (a,b)
  do (if (a < b) then b->a if)
  exit a
  #);
fabs: external
  (* returns |x|, the absolute value of x *)
  (# x,res: @Real;
  enter x
  exit res
  #);
```

```
floor: external
  (* largest integer value (in real format) not greater than x.
   *
   * If x is a NaN, floor returns a NaN.  If x is infinite, floor
   * returns x. Invalid operation is signaled by floor if x is a NaN.
   * If x is a non-integral finite value, floor signals inexact.
   *)
  (# x,res: @Real;
  enter x
  exit res
  #);
fmod: external
  (* Whenever possible, the fmod pattern returns the number f with
   * the same sign  as x, such that x = i*y + f for some integer i, and
   * |f| < |y|. If y is 0, fmod returns   a NaN.
   *)
  (# x,y,f: @Real;
  enter (x,y)
  exit f
  #);
```

4. The numberio Library

Based integers and reals

`Numberio` is a library for reading and writing numerals (integers, based integers, radix integers, and reals). The format of these numerals corresponds directly to the format of these numerals as defined by the BETA language (except for radix integers that are not supported by the BETA language). `Numberio` also contains a general `getNumber` operation, that is able to read any numeral, and return the proper value read.

Grammar for getNumber operation

The following grammar defines the exact syntax of the numerals:

```
      N ::= D+                        int              314
         |  D+ '.' D+                 real             3.14
         |  D+ '.' D+ 'E' E           real             3.14E8
                                      real             3.14E+8
                                      real             3.14E-8
         |  D+  'E' E                 real             3E8
                                      real             3E+8
                                      real             3E-8
         |  D+ 'X' (D|L)+             based            2X0101
                                      based            8x0845
                                      based            16xAF12
         |  (D|L)+                    radix            AF12
      D ::= '0' | ... | '9'
      L ::= 'A' | ... | 'Z'
      E ::= D+
         |  '+' D+
         |  '-' D+
```

Integer examples:`10`, `0`, `123`.

A based integer has the form`<base>X<number>` . Examples are:

`2X101`	base=2,	number= 4*1 + 2*0 + 1*1 = 5
`8X12`	base=8,	number= 8*1 + 1*2 = 10
`16x2A1`	base=16,	number= 256*2 + 16*10 + 1*1 = 673
`0x2A1`	base=16, i.e. base=0 is interpreted as base=16	

Examples of reals are:

`3.14`, `3.14E-8`, `3E+8`.

All letters may be in lower or upper case.

The various read and write operations contain several facilities for controlling the read and write. Please consult the interface description later for details.

4.1 Using the numberio Fragment

A program using the`numberio` fragment will have the following structure:

```
INCLUDE '~beta/basiclib/v1.6/numberio
--- program: descriptor ---
(# ...
   r: @real;
do ...
   getReal -> r;
   ...
#)
```

4.1.1 Reading numbers from standard input

The following example illustrates using numberio to read general numerics, as well as based, integer and reals from keyboard.

NumberioDemo.bet

```
ORIGIN '~beta/basiclib/v1.6/numberio';
--- program: descriptor ---
(#
do L: (#
      do 'Enter a general number: '->puttext;
         getNumber
         (# valueError::< (# do msg[]->screen.putline; true->continue #);
            syntaxError::< (# do msg[]->screen.putline; get; true->continue #);
            baseError::< (# do msg[]->screen.putline; true->continue #);
            integerValue::<
              (# do '\tInteger='->puttext; value->putInt #);
            basedValue::<
              (#
              do '\tBase='->puttext; base->putint;
                 ' Value='->puttext; value->putInt;
                 ' BasedValue='->puttext; (base,value)->putBased;
              #);
            realValue::<
              (#
              do '\tReal='->puttext;
                 value->screen.putReal(# format::< (# do exp->style; true->upcase
#)
              #) #);
         do newline;
         #);
         'Enter a based number: '->puttext;
         getBased
         (# valueError::< (# do msg[]->screen.putline; true->continue #);
            syntaxError::< (# do msg[]->screen.putline; true->continue #);
            baseError::< (# do msg[]->screen.putline; true->continue #);
         do '\tBase='->puttext; b->putint;
            ' Value='->puttext; i->putInt;
            ' BasedValue='->puttext; (b,i)->putBased;
         #); newline;
         'Enter a real number: '->puttext;
         getReal
         (# valueError::< (# do msg[]->screen.putline; true->continue #);
            syntaxError::< (# do msg[]->screen.putline; true->continue #);
            baseError::< (# do msg[]->screen.putline; true->continue #);
         do '\tReal='->puttext
         #)->screen.putreal(# format::< (# do exp->style; true->upcase #) #);
newline;
         'Enter an integer number: '->puttext;
         getInteger
         (# valueError::< (# do msg[]->screen.putline; true->continue #);
            syntaxError::< (# do msg[]->screen.putline; true->continue #);
            baseError::< (# do msg[]->screen.putline; true->continue #);
         do '\tInteger='->puttext
         #)->putInt; newline;
         restart L
      #)
#)
```

4.2 Interface Description for the numberio Library

```
ORIGIN 'betaenv';
BODY 'private/numberioBody'
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1992-96
 *       All rights reserved.
 *
 * This fragment implements the following stream operations:
 *    getNumber  * reads a number from THIS(stream).
 *               * The number may be either an integer,
 *               * a based number or a real number
 *    getBased   * reads a based number from THIS(stream)
 *    getRadix   * reads a based number from THIS(stream),
 *               * without the 'bbx' part
 *    getInteger * reads an integer number from THIS(stream)
 *    getReal    * reads a real number from THIS(stream)
 *    putReal    * appends a textual rep. of a real value to
 *               * THIS(stream)
 *    putBased   * appends a textual rep. of a integer value in a
 *               * particular to THIS(stream).  The textual
 *               * representation in in the given base
 *    putRadix   * as putBased, except that the 'bbx' part is
 *               * not printed
 *    getHex     * similar to getRadix with radix 16, but more efficient
 *    putHex     * similar to putBased with base 16, but more efficient.
 *    getBinary  * similar to getRadix with radix 2, but more efficient.
 *    putBinary  * similar to putBased with base 2, but more efficient.
 *    putByteHex    * like putHex, except that it only prints one byte.
 *    putByteBinary * like putBinary, except that it only prints one byt
 *
 *    asNumber   * abstract pattern for the following asBased,
 *               * asRadix, and asReal operations
 *    asBased    * returns the based number present in
 *               * THIS(stream)
 *    asRadix    * returns the based number present in
 *               * THIS(stream), without the 'bbx' part
 *    asReal     * returns the real number present in
 *               * THIS(stream)
 *
 * The corresponding short-cuts for keyboard.getNumber, etc, and
 * screen.putReal, etc. are also included in this fragment.
 *
 * Since the asNumber operations does not make sence for keyboard,
 * no short-cuts are defined for these.
 *)
```

```
--- StreamLib: attributes ---
getNumber:
  (* getNumber reads a number from the current position of
   * this(stream).
   * The number is either an integer (in base 10), an integer with a
   *  given base, or a real.
   * Integer examples:  10, 0, 123
   * A based integer has the form <base>X<number>. Examples are:
   *    2X101         base=2, number= 4*1 + 2*0 + 1*1 = 5
   *    8X12          base=8, number= 8*1 + 1*2 = 10
   *    16x2A1        base=16, number= 256*2 + 16*10 + 1*1  = 673
   *     0x2A1        base=16, i.e. base=0 is interpreted as base=16
   * Examples of reals are:
   *   3.14, 3.14E-8, 3E+8
   * The following grammar defines the exact syntax of the numbers:
   *
   * N ::= D+                        Int             314
   *    |  D+ '.' D+               real            3.14
   *    |  D+ '.' D+ 'E' E         real            3.14E8
   *                                 real            3.14E+8
   *                                 real            3.14E-8
   *
   *    |  D+  'E' E                 real            3E8
   *                                 real            3E+8
   *                                 real            3E-8
   *    |  'X' D | L+                based           2X0101
   *                                 based           8x0845
   *                                 based           16xAF12
   * D ::= '0' | ... | '9'
   * L ::= 'A' | ... | 'Z'
   * E ::= D+
   *    |  D+ '+' D+
   *    |  D+ '-' D+
   *
   * All letters may be in lower or upper case.
   * After the call, the stream is positioned
   * after the first char not in the number.
   *)
  (# integerValue:<
       (* the number has the form
        *     x
        * value contains the integer value
        *)
       integerValuePtn;
     integerValuePtn:
       (# value: @integer enter value do INNER #);
     basedValue:<
       (* the number has the form
        *     bXy
        * base contains the base number
        * value contains the integer value (in base 10)
        *)
       basedValuePtn;
     basedValuePtn:
        (# base,value: @integer enter (base,value) do INNER #);
      realValue:<
        (* the  number has the form
         *       x.yEz
         *       l is the number of leading zero's in y. i.e. in
         *       3.0017E-12, x=3, y=17,l=2 and z=-12
         * value contains the real value
         *)
        realValuePtn;
      realValuePtn:
        (# x,y,l,z: @real; value: @real enter(x,y,l,z,value)
        do INNER #);
      syntaxError:< streamException
        (# peekCh: @char
        enter peekCh
        do 'getNumber: Syntax error - looking at: "'->msg.append;
           (if peekCh = ascii.nul then 'NUL'->msg.puttext
            else peekCh->msg.put if);
           '"'->msg.put; INNER #);
      baseError:< streamException
        (# base: @integer
        enter base
        do 'getNumber: Error in base - looking at: "'->msg.append;
           base->msg.putInt; '"'->msg.put; INNER #);
      valueError:< streamException
        (# peekCh: @char
        enter peekCh
        do 'getNumber: Illegal value type - looking at: "'->msg.append;
           peekCh->msg.put; '"'->msg.put; INNER #);
      getn: @...
  do getn;
     INNER getNumber
  #);
getReal: getNumber
  (# r: @real;
      realValue::< (# do value->r #);
  do INNER getReal
  exit r
  #);
getBased: getNumber
  (# i, b: @integer;
      basedValue::< (# do value->i; base->b #);
  do INNER getBased
  exit (b,i)
  #);
getInteger: getNumber
  (# i: @integer;
      integerValue::< (# do value->i #);
  do INNER getInteger
  exit i
  #);
getRadix:
  (* gets a number in the specified radix.  GetRadix is similar to
   * getBased, except that is does NOT expect the 'bbx' prefix
   *)
  (# radix, value: @integer;
     radixError:< streamException
       (# radix: @integer
       enter radix
       ...
       #);
     getr: (* private *) @...
  enter radix
  do getr;
     INNER getRadix
  exit value
  #);

putBased:
  (* Takes a number and a base, and prints the number in that base.
   * If base is 0, base 16 is assumed, and the format "0xnnn" is used.
   * If base is negative, 1 or greater that 126, the baseError
   * exception is invoked.
   *
   * The format is default "bbxnnnn", where "bb" is the base (in
   * decimal), and "nnnn" is the number, printed in the base. "x"
   * separates the two parts.  The format may be controlled by the
   * signed, blankSign, upcase, uppercase, width, adjustLeft,
   * zeroPadding, noBasePrefix, baseWidth and baseZeroPadding variable
   * attributes. If noBasePrefix is true, the "bbx" part is omitted.
   *)
  (# value, base: @integer;
     baseError:< streamException
       (# base: @integer
       enter base
       do 'putBased: Illegal base: "'->msg.append;
          base->msg.putInt; '"'->msg.put; INNER #);
     (* The format may be further controlled by the signed, blankSign,
      * width, adjustLeft and zeroPadding variable attributes.
      * width is extended if it is too small.
      *
      * Examples:
      *   (10,10)->putBased
      *      yields: '10x10'
      *   (2,5)->putBased(# do 10->width; true->adjustLeft #);
      *      yields: '2x101     '
      *   (2,5)->putBased(# do 10->width; true->zeroPadding #);
      *      yields: '2x00000101'
      *)
     signed:
       (* If the number is positive, a '+' will always be displayed
        *)
       @boolean;
     blankSign:
       (* If the number is positive, a ' ' space is displayed as the
        * sign.  Ignored if signed=true
        *)
```

```
          @boolean;
        upcase: @boolean
          (* Specifies whether an upcase 'X' or a lowcase 'x' is the
           * be used in the 'bbx' part.
           *);
        uppercase: @boolean
          (* Specifies whether uppercase letters or lowercase letters
           * are used in the 'nnnn' part (for base>9).
           *);
        width: (* Minimum width *) @integer;
        adjustLeft: @boolean
          (* Specifies if the number is to be aligned left or right,
           * if padding of spaces is necessary to fill up the specified
           * width.
           *);
        zeroPadding:
          (* width is padded with leading zero instead of spaces.
           * Ignored if adjustLeft=true
           *)
          @boolean;
        noBasePrefix: (* If true, the 'bbx' part is omitted *)
          @boolean;
        baseWidth: (* minimun width for the 'bbx' part *)
          @integer;
        baseZeroPadding:
          (* baseWidth is padded with leading zero instead of spaces *)
          @boolean;
        format:< (# do INNER #);
        putb: @...
     enter (base, value)
     do INNER putBased; putb
     #);
   putReal:
     (* Append a real to THIS(stream). The format may be controlled by
      * the style, signed, blankSign, precision, upcase, width,
      * adjustLeft and zeroPadding variable attributes
      *)
     (# r: @real;
        style: @integer
          (* Controls the style, and may be one of plain, exp and noexp
           * (noexp is the default)
           *);
        noexp: (* The notation [-]mmm.dddddd is used *)
          (# exit 0 #);
        exp: (* The notation [-]m.ddddddE[+|-]xx is used *)
          (# exit 1 #);
        plain:
          (* In this style, precision is the total number of digits in
           * the printed real (not the number of digits in the fraction,
           * as in the other styles).
           *
           * The exp or noexp style is used, dependent on the value being
           * printed.  Exp style is used only if the exponent is less
```

```
         * than -4 or greater than or equal to the precision; otherwise
         * the noexp notation is used.  Trailing zeros are not printed
         * as part of the fractional part and a decimal point is
         * printed if not followed by a digit
         *)
        (# exit 2 #);
      signed: (* If real is positive, a '+' will always be displayed *)
        @boolean;
      blankSign:
        (* If real is positive, a ' ' space is displayed as the sign.
         * Ignored if signed=true
         *)
        @boolean;
      precision: @integer
        (* The number of d's in the expressions above, default 6 *);
      upcase: @boolean
        (* Specifies whether an upcase 'E' or a lowcase 'e' is the
         * be used in the exp style.
         *);
      width: (* Minimum width *)
        @integer;
      adjustLeft: @boolean
        (* Specifies if the number is to be aligned left or right,
         * if padding of spaces is necessary to fill up the specified
         * width.
         *);
      zeroPadding:
        (* width is padded with leading zero instead of spaces.
         * Ignored if adjustLeft=true
         *)
        @boolean;
      (* Examples:
       *   10*pi -> putreal;
       *      yields: '31.415926'
       *   10*pi -> putreal(# do 10->width; true->adjustLeft #);
       *      yields: '31.415926   '
       *   10*pi -> putreal(# do exp->style; true->upcase #);
       *      yields: '3.1415926E+01'
       *   10*pi -> putreal(# do exp->style; 2->precision #);
       *      yields: '3.14e+01'
       *)
      format:< (# do INNER #);
      putr: @...
  enter r
  do 1->width; 6->precision; format; INNER putReal; putr
  #);
putRadix: putBased
  (#
  do true->noBasePrefix; INNER putRadix
  #);

putHex:
  (* prints a hexadecimal representation of x (as unsigned word) on
   * this(stream). Similar to
```

```
       * (16,x)->putRadix(# do true->zeroPadding; 8->width #)
       * but more efficient.
       *)
      (# x: @integer;
         putH: (*private*)@...
      enter x
      do INNER putHex; putH
      #);

    putByteHex:
      (* prints a hexadecimal representation of byte 'byte' in x (as
       * unsigned word) on this(stream)
       *)
      (# x: @integer;
         byte: @integer;
         putBH: (*private*)@...
      enter (x,byte)
      do INNER putByteHex; putBH
      #);

    putBinary:
      (* prints a binary representation of x (as unsigned word) on
       * this(stream). Similar to
       * (2,x)->putRadix(# do true->zeroPadding; 32->width #)
       * but more efficient.
       *)
      (# x: @integer;
         putB: (*private*)@...
      enter x
      do INNER putBinary; putB
      #);

    putByteBinary:
      (* prints a binary representation of byte 'byte' of x (as unsigned
       * word) on this(stream)
       *)
      (# x: @integer;
         byte: @integer;
         putBB: (*private*)@...
      enter (x, byte)
      do INNER putByteBinary; putBB
      #);

    getHex:
      (* reads a hexadecimal number from this(stream) and returns the
       * value in x (as unsigned word). Similar to 16->getRadix but more
       * efficient.
       *)
      (# x: @integer;
         noNumberError:< streamException
           (# peekCh: @char
           enter peekCh
           do 'getHex: the number begins with: "'->msg.append;
```

```
          (if peekCh = ascii.nul then 'NUL'->msg.puttext
           else peekCh->msg.put if);
          '".  Not a legal hexadecimal digit'->msg.puttext;
          INNER noNumberError
       #);
     getH: (*private*)...
  do INNER getHex; getH
  exit x
  #);
getBinary:
  (* reads a binary number from this(stream) and returns the value in
   * x (as unsigned word). Similar to 2->getRadix but more efficient.
   *)
  (# x: @integer;
     noNumberError:< streamException
       (# peekCh: @char
       enter peekCh
       do 'getBinary: the number begins with: "'->msg.append;
          (if peekCh = ascii.nul then 'NUL'->msg.puttext
           else peekCh->msg.put if);
          '".  Not a legal binary digit'->msg.puttext;
          INNER noNumberError
       #);
     getB: (*private*)...
  do INNER getBinary; getB
  exit x
  #);

asNumber:
  (# syntaxError:< streamException
       (# peekCh: @char
       enter peekCh
       do 'asNumber: Syntax error - looking at: "'->msg.append;
          peekCh->msg.put; '"'->msg.put;
          INNER syntaxError
       #);
     baseError:< streamException
       (# base: @integer
       enter base
       do 'asNumber: Error in base - looking at: "'->msg.append;
          base->msg.putInt; '"'->msg.put;
          INNER baseError
       #);
     valueError:< streamException
       (# peekCh: @char
       enter peekCh
       do 'asNumber: Illegal value type - looking at: "'->msg.append;
          peekCh->msg.put; '"'->msg.put;
          INNER valueError
       #);
  do reset;
     INNER asNumber;
     ScanWhiteSpace; (if not eos then peek->syntaxError if)
  #);
```

```
asReal: asNumber
  (# r: @real
    ...
  exit r
  #);
asBased: asNumber
  (# i, b: @integer
    ...
  exit (b,i)
  #);
asRadix: asNumber
  (# radix, value: @integer
  enter radix
  ...
  exit value
  #);
asInteger: asNumber
  (# i: @integer
    ...
  exit i
  #)

--- lib: attributes ---

getNumber: keyboard.getNumber
  (# do INNER getNumber #);
getReal: keyboard.getReal
  (# do INNER getReal #);
getBased: keyboard.getBased
  (# do INNER getbased #);
getRadix: keyboard.getRadix
  (# do INNER getRadix #);
getInteger: keyboard.getInteger
  (# do INNER getInteger #);

putReal: screen.putReal
  (# do INNER putReal #);
putBased: screen.putBased
  (# do INNER putBased #);
putRadix: screen.putRadix
  (# do INNER putRadix #);

getHex: keyboard.getHex
  (# do INNER getHex #);
getBinary: keyboard.getBinary
  (# do INNER getBinary #);

putHex: screen.putHex
  (# do INNER putHex #);
putByteHex: screen.putByteHex
  (# do INNER putByteHex #);
putBinary: screen.putBinary
```

```
  (# do INNER putBinary #);
putByteBinary: screen.putByteBinary
  (# do INNER putByteBinary #);
```

5. The formatio Library

Formatted input/output

This library provides facilities for formatted input and output (similar to the `scanf` and `printf` functions in C). These facilities are implemented in the form of the `getFormat` and `putFormat` operations that are added as stream attributes, making formatted input and output available for any stream.

Format string

Both `getFormat` and `putFormat` take a text string as argument. This text string must contain a format specification of the input to be read from (respectively output to) the stream. The format string may be any string, possibly with one or more embedded markers. The markers specify the variable parts of the expected input (respectively output), such as integer values. The markers are indicated in the string by a leading `'%'`. Following the `'%'` is the specification of the marker type.

`getFormat` accepts the following marker syntax:

```
%[width][.[precision]]dioxXrRbBfeEgGcsn%
```

`putFormat` accepts the following marker syntax:

```
%-+ [[0]width][.[[0]precision]]dioxXrRbBfeEgGcsn%
```

As it can be seen, the marker syntax is very similar.

Marker type

Corresponding to every marker type (given by the `dioxXrRbBfeEgGcsn` part of the marker syntax), `getFormat` and `putFormat` defines an attribute (with the same name) as the marker symbol – e.g. there is an attribute with the name d, corresponding to the d marker type. Due to BETA not being case-sensitive in identifiers, the attributes corresponding to the upper-case marker types are called e.g. uG for upper g.

The functionality of formatted input and output can now easiest be described by showing the following example:

```
(# t, s: @text;
   theName: ^text; theValue: integer;
do 'name: temperature value: 72 name: speed value: 35' -> t;
   0->t.pos;
   'name: %s value: %d'
     -> t.getFormat(# do s->theName[];  d->theValue #);
   'The name is %s and the value is %d\n'
     -> s.putFormat(# do theName[]->s; theValue->d #);
   'name: %s value: %d'
     -> t.getFormat(# do s->theName[];  d->theValue #);
   'The name is: %s and the value is: %d\n'
     -> s.putFormat(# do theName[]->s; theValue->d #)
#)
```

At the end of this program, s will contain the following text:

```
The name is: temperature and the value is: 72
The name is: speed and the value is: 35
```

The `getFormat` and `putFormat` operations raise exceptions if the format specifications are not satisfied.

5.1 Using the formatio Fragment

A program using the`formatio` fragment will have the following structure:

```
INCLUDE '~beta/basiclib/v1.6/formatio
--- program: descriptor ---
(# ...
   r: @real;
do ...
   'real value: %e' -> putformat(# do r->e #);
   ...
#)
```

5.1.1 Illustrating putFormat

putformatDemo.bet

```
ORIGIN '~beta/basiclib/v1.6/formatio';
--- program: descriptor ---
(#
do 'string=%s, integer=%i, real=%f\n\n'
     -> putFormat(# do 'abc'->s; -30->i; 3.14->f #);

   '(1234567){12345678}[123456789]\n' -> putFormat;
   '(%7i){%8i}[%9i]\n\n' -> putFormat(# do 27->i; 30->i; -45->i #);

   '"%%s,%%s,%%s" = %s,%s,%s\n\n'
     -> putFormat(# do '27'->s; 'Hello'->s; '-45'->s #);

   '  123456789   1234567890123456       12345678\n' -> putFormat;
   'x=%9.3f,y=%16.e,z=%.8f\n\n'
     -> putFormat(# do 27.5->f; 30.5->e; -45.5->f #);

   '  1234567890   1234567890   12345678\n' -> putFormat;
   'x=%*.*f,y=%*.2e,z=%8.*f\n\n'
     -> putFormat(# do 10->width; 3->precision; 27.5->f; 30.5->e; -45.5->f #);

#)
```

5.1.2 Illustrating getFormat

getformatDemo.bet

```
ORIGIN '~beta/basiclib/v1.6/formatio';
--- program: descriptor ---
(# i1, i2, i3, i4, i5: @integer;
   s1, s2: ^text;
   r1, r2, r3: @real;
   chr: @char;
   t, fmt: ^text;
do '123,%456, 789 JorgenLKnudsen 1.2,2+2=43.45, 67.89 abc, 101, 0x101, 0101,
101xxx'->t[];
   0->t.pos;

   'input:\t"%s"\n\n' -> putFormat(# do t[]->s #); newline;

   '%i,%%%i, %i' -> fmt[] -> t.getFormat(# do i->i1; i->i2; i->i3 #);
   'format:\t%s\nread:\t%i, %i, %i\n\n'
     -> putFormat(# do fmt[]->s; i1->i; i2->i; i3->i #);
   '%6s%c%s' -> fmt[] -> t.getFormat(# do s->s1[]; c->chr; s->s2[] #);
   'format:\t"%s"\nread:\t"%s", "%c", "%s"\n\n'
     -> putFormat(# do fmt[]->s; s1[]->s; chr->c; s2[]->s #);

   '%f,2+2=4%e, %f' -> fmt[] -> t.getFormat(# do f->r1; e->r2; f->r3 #);
   'format:\t"%s"\nread:\t%f, %f, %f\n\n'
     -> putFormat(# do fmt[]->s; r1->f; r2->f; r3->f #);

   '%x, %o, %i, %i, %i%s' -> fmt[]
     -> t.getFormat(# do x->i1; o->i2; i->i3; i->i4; i->i5; s->s1[] #);
   'format:\t"%s"\nread:\t%i, %i, %i, %i, %i, "%s"\n'
     -> putFormat(# do fmt[]->s; i1->i; i2->i; i3->i; i4->i; i5->i; s1[]->s #);

   '123,%456, 124 %JorgenLKnudsen 1.2,2+2=43.45, 67.89 abc, 111, 0x111, 0111,
111xxx'->t[];
   0->t.pos;
   '\n\ninput:\t"%s"\n\n' -> putFormat(# do t[]->s #); newline;

   '%i,%%%i, %i %6s%c%s%f' -> fmt[] -> t.getFormat(# do i->i1; i; i->i2; s; c; s;
f->r1; #);
   'format:\t%s\nread:\t%i, %i, %f\n\n'
     -> putFormat(# do fmt[]->s; i1->i; i2->i; r1->f; #);

#)
```

5.2 Interface Description for the formatio Library

```
ORIGIN 'betaenv';
BODY 'private/formatioBody'
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1984-96
 *       All rights reserved.
 *
 *)
--- streamLib: attributes ---
getFormat: formatter
  (* getFormat accepts the following syntax for markers:
   *
   *      %[width][.[precision]]dioxXrRbBfeEgGcsn%
   *
   * where width is a (unsigned) decimal number (or '*'), and
   *   precision is a (unsigned) decimal number (or '*'),
   *
   * and where
   *     [ ... ] means that the enclodes is optional,
   * and  ...  means one if the enclosed characters.
   *
   * Width is only interpreted in conjunction with the 's' marker.
   * Precision is only interpreted in conjunction with the
   * 'r' and 'R' markers.
   *
   * For all but the 'c' marker, leading white space are skipped.
   *
   * The '%' marker means that a '%' is expected on the input stream.
   *
   * Actually, getFormat accepts the same syntax as putFormat (see
   * later):
   *
   *      %-+ [[0]width][.[[0]precision]]dioxXrRbBfeEgGcsn%
   *
   * but only the above part of that syntax is actually interpreted by
   * getFormat.  The reason for accepting the same syntax is to allow
   * a format string to be used both for getFormat and putFormat.
   *)
  (# width:
       (* sets the default precision to be used in future '*' width
        * specifications
        *)
       (# w: @integer enter w ... #);
     precision:
```

```
  (* sets the default precision to be used in future '*'
   * precision specifications
   *)
  (# p: @integer enter p ... #);
marker: scanForMarker
  (# formatEOS:: (# do mark->missingMarker #)
  ...
  #);
d: marker (* read a decimal number *)
  (# mark:: (# do 'd'->value #);
     value: @integer
  ...
  exit value
  #);
i: marker
  (* read a number, either decimal, octal or hexadecimal using C
   * conventions: 0nnn implies octal, 0xnnn implies hexadecimal,
   * decimal otherwise.
   *)
  (# mark:: (# do 'i'->value #);
     value: @integer
  ...
  exit value
  #);
o: marker (* read an octal number *)
  (# mark:: (# do 'o'->value #);
     value: @integer
  ...
  exit value
  #);
x: marker (* read a hexadecimal number *)
  (# mark:: (# do 'x'->value #);
     value: @integer
  ...
  exit value
  #);
uX: marker (* read a hexadecimal number.  Identical to 'x' *)
  (# mark:: (# do 'X'->value #);
     value: @integer
  ...
  exit value
  #);
r: marker
  (* read a number in radix given by precision *)
  (# mark:: (# do 'r'->value #);
     value: @integer
  ...
  exit value
  #);
uR: marker
  (* read a number in radix given by precision.  Identical to
   * 'r'
   *)
  (# mark:: (# do 'R'->value #);
     value: @integer
  ...
  exit value
  #);
b: marker
  (* read a number as based number (i.e. in the bbxnnn format) *)
  (# mark:: (# do 'b'->value #);
     base, value: @integer
  ...
  exit (base,value)
  #);
uB: marker
  (* read a number as based number (i.e. in the bbxnnn format).
   * Identical to 'b'
   *)
  (# mark:: (# do 'B'->value #);
     base, value: @integer
  ...
  exit (base,value)
  #);
f: marker (* read a real *)
  (# mark:: (# do 'f'->value #);
     value: @real
  ...
  exit value
  #);
e: marker (* read a real.  Identical to 'f' *)
  (# mark:: (# do 'e'->value #);
     value: @real
  ...
  exit value
  #);
uE: marker (* read a real.  Identical to 'f' *)
  (# mark:: (# do 'E'->value #);
     value: @real
  ...
  exit value
  #);
g: marker (* read a real.  Identical to 'f' *)
  (# mark:: (# do 'g'->value #);
     value: @real
  ...
  exit value
  #);
uG: marker (* read a real.  Identical to 'f' *)
  (# mark:: (# do 'G'->value #);
     value: @real
  ...
  exit value
  #);
c: marker (* read a single character *)
  (# mark:: (# do 'c'->value #);
     value: @char
  ...
```

```
            exit value
            #);
          s: marker
            (* read a text atom (i.e. a sequence of non-white space
             * characters, terminated by a white space character - similar
             * to getAtom).  If width is non-zero, at most width characters
             * are read.  Otherwise, characters are read until next
             * whitespace character
             *)
            (# mark:: (# do 's'->value #);
               value: ^text
            ...
            exit value[]
            #);
          n: marker
            (* return the number of characters read until this point *)
            (# mark:: (# do 'n'->value #);
               value: @integer
            ...
            exit value
            #);
          match:: (* private *)
            (# ... #)
       do INNER getFormat
       #);

     putFormat: formatter
       (* putFormat accepts the following syntax for markers:
        *
        *      %-+ [[0]width][.[[0]precision]]dioxXrRbBfeEgGcsn%
        *
        * where width is a (unsigned) decimal number (or '*'), and
        *   precision is a (unsigned) decimal number (or '*') and where
        *
        *     [ ... ] means that the enclodes is optional,
        * and  ...  means one if the enclosed characters.
        *
        * '-' implies output leftjustified in field
        * '+' implies output numbers signed (always with leading '+' or
        *                                     '-')
        * ' ' implies output numbers with blank sign (i.e. leading ' ' if
        *                                             positive)
        * '0' in front of either width or precision implies zero padding
        * width specifies the minimun width of the output field.
        * precision is interpreted for various things in the 'rRbBfeEgG'
        *     markers.
        *
        * The '%' marker means that a '%' is put on the output stream.
        *)
       (# width:
            (* sets the default precision to be used in future '*' width
             * specifications
             *)
            (# w: @integer enter w ... #);
```

```
precision:
  (* sets the default precision to be used in future '*'
   * precision specifications
   *)
  (# p: @integer enter p ... #);
marker: scanForMarker
  (# formatEOS:: (# do mark->missingMarker #)
  ...
  #);
d: marker (* insert the integer *)
  (# mark:: (# do 'd'->value #);
     value: @integer
  enter value
  ...
  #);
i: marker (* insert the integer. Identical to 'd' *)
  (# mark:: (# do 'i'->value #);
     value: @integer
  enter value
  ...
  #);
o: marker (* insert the integer in octal *)
  (# mark:: (# do 'o'->value #);
     value: @integer
  enter value
  ...
  #);
x: marker (* insert the integer in hexadecimal *)
  (# mark:: (# do 'x'->value #);
     value: @integer
  enter value
  ...
  #);
uX: marker
  (* insert the integer in hexadecimal, using uppercase letters
   *)
  (# mark:: (# do 'X'->value #);
     value: @integer
  enter value
  ...
  #);
r: marker
  (* insert the integer in radix given by precision *)
  (# mark:: (# do 'r'->value #);
     value: @integer
  enter value
  ...
  #);
uR: marker
  (* insert the integer in radix given by precision, using
   * uppercase letters if radix>10
   *)
  (# mark:: (# do 'R'->value #);
     value: @integer
  enter value
  ...
  #);
b: marker
  (* insert the integer as based number (i.e. in the bbxnnn
   * format) with base given by precision
   *)
  (# mark:: (# do 'b'->value #);
     value: @integer
  enter value
  ...
  #);
uB: marker
  (* insert the integer as based number (i.e. in the bbxnnn
   * format) with base given by precision, using uppercase
   * letters if base>10
   *)
  (# mark:: (# do 'B'->value #);
     value: @integer
  enter value
  ...
  #);
f: marker (* insert the real in noexp style *)
  (# mark:: (# do 'f'->value #);
     value: @real
  enter value
  ...
  #);
e: marker (* insert the real in exp style *)
  (# mark:: (# do 'e'->value #);
     value: @real
  enter value
  ...
  #);
uE: marker (* insert the real in exp style and upcase 'E' *)
  (# mark:: (# do 'E'->value #);
     value: @real
  enter value
  ...
  #);
g: marker (* insert the real in plain style *)
  (# mark:: (# do 'g'->value #);
     value: @real
  enter value
  ...
  #);
uG: marker (* insert the real in plain style and upcase 'E' *)
  (# mark:: (# do 'G'->value #);
     value: @real
  enter value
  ...
  #);
c: marker (* insert the char *)
  (# mark:: (# do 'c'->value #);
           value: @char;
           putc: @(* Private *)...
        enter value
        do putc
        #);
      s: marker (* insert the text *)
        (# mark:: (# do 's'->value #);
           value: ^text;
           puts: @(* Private *)...
        enter value[]
        do puts
        #);
      n: marker
        (* return the length of the result string until this point *)
        (# mark:: (# do 'n'->value #);
           value: @integer
        ...
        exit value
        #);
      match:: (* private *)
        (# ... #)
   do INNER putFormat
   #);

formatter: (* superpattern: for putFormat and getFormat *)
   (# illegalFormat:< exception
        (# mark: @char
        enter mark
        ...
        #);
      missingMarker:< exception
        (# mark: @char
        enter mark
        ...
        #);
      missingField:< exception
        (# ... #);
      inputError:< exception
        (# chFound, chExpected: @char
        enter (chFound, chExpected)
        ...
        #);
      scanForMarker: (* Private *)
        (# mark:< charValue;
           formatEOS:< exception;
           fieldWidth, precisionSpec: @integer;
           leftFlag, signedFlag, blankFlag,
           alternativeFlag, zeroFlag, longFlag: @boolean
        ...
        #);
      match:< (* private *)
        (# ch: @char
        enter ch
        do INNER match
```

```
          #);
        private: @...;
        formatStr: ^text
     enter formatStr[]
     ...
     exit this(stream)[]
     #);

--- lib: attributes ---

getFormat: keyboard.getFormat(# do INNER getFormat #);

putFormat: screen.putFormat(# do INNER putFormat #);
```

6. The random Library

This library provides elaborate random number generation facilities. The random library contains random generators with different statistical distributions, such as uniform, normal, binomial and poison distributions.

simple random generator : `ignlgi`

The simple random generator most often used (namely the uniform integer random generator) is in this library the `ignlgi` generator which returns a uniformly distributed integer in the range [1, 2147483562]. It is also available in the `ignuin` variant which returns a uniformly distributed integer in a range specified by the user.

Another simple and often used random generator is the uniform real random generator which in this library is available as the `ranf` generator which returns a uniformly distributed real in the range]0, 1[(i.e. 0 and 1 are never returned). The `genunf` variant returns a uniformly distributed real in a range specified by the user.

The library offers facilities for specifying the seeds of the generator by the `setsd` operation.

The library offers facilities for maintaining up to 32 parallel random generators.

For more details, see the interface descriptions below.

6.1 Using the random Fragment

A program using the`random` fragment will have the following structure:

```
INCLUDE '~beta/basiclib/v1.6/random
--- program: descriptor ---
(# ...
   i: @integer;
do ...
   (123, 456) -> setsd;
   ...
   (1,100) -> inguin -> i;
   ...
#)
```

6.1.1 Illustrating Integer Random Generator

tstign.bet

```
ORIGIN '~beta/basiclib/v1.6/random';
--- program: descriptor ---
(# testIgn:
     (# iarray: [mxint]@integer;
        itmp: @integer;
        lo, hi, i, up: @integer
     do (for i:nrep repeat
             (1,mxint)->ignuin->itmp;
       itmp->screen.putint(# do 7->width #);
             (if (i mod 10) = 0 then newline if);
             iarray[itmp]+1->iarray[itmp];
        for);
        newline;
        '         Counts of Integers Generated:'->putline;
        (* Print 10 to a line using 7 characters for each field.         *)
        mxint->up; 1->lo;
        l: (if lo<=up then
               (lo+9,up)->min->hi;
               lo->i;
              ll: (if i<=hi then
                       iarray[i]->screen.putint(# do 7->width #);
                       i+1->i;
                       restart ll
                   if);
               screen.newline;
               lo+10->lo;
               restart l;
           if)
     #);

   mxint,nrep: @integer;

do ' Tests uniform random integer generator.'->putline; newline;

   ' Enter two seeds to initialize rn generator: '->puttext;
   (getint,getint)->setall;

   ' Enter maximum uniform integer: '->puttext;
   getint->mxint;
```

```
     ' Enter number of randoms to generate: '->puttext;
     getint->nrep;

     testIgn;
  #)
```

6.2 Interface Description for the random Library

```
ORIGIN 'betaenv';
BODY 'private/randombody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1995-96
 *       All rights reserved.
 *)
--- lib: attributes ---

initgn: external
  (*
   * INIT-ialize current G-e-N-erator
   *      Reinitializes the state of the current generator
   *
   * Arguments
   *      isdtyp -> The state to which the generator is to be set
   *        isdtyp = -1 => sets the seeds to their initial value
   *        isdtyp = 0 => sets the seeds to the first value of the
   *                       current block
   *        isdtyp = 1 => sets the seeds to the first value of the
   *                       next block
   *
   * Method
   *      This is a transcription from Pascal to Fortran of routine
   * Init_Generator from the paper
   *      L'Ecuyer, P. and Cote, S. "Implementing a Random Number
   * Package with Splitting Facilities."  ACM Transactions on
   * Mathematical Software, 17:98-111 (1991)
   *)
  (# isdtyp: @integer
  enter (isdtyp)
  #);

gscgn: external
  (*
   * Get/Set GeNerator
   *      Gets or returns in G the number of the current generator
```

```
     *
     * Arguments
     *      getset --> 0 Get 1 Set
     *      g <-- Number of the current random number generator (1..32)
     *)
    (# getset, g: @integer
    enter (getset, g)
    #);

setsd: external
    (*
     * SET S-ee-D of current generator
     *      Resets the initial seed of the current generator to ISEED1
     *      and ISEED2. The seeds of the other generators remain
     *      unchanged.
     * Arguments
     *      iseed1 -> First integer seed
     *      iseed2 -> Second integer seed
     * Method
     *      This is a transcription from Pascal to Fortran of routine
     * Set_Seed from the paper L'Ecuyer, P. and Cote, S. "Implementing a
     * Random Number Package with Splitting Facilities."  ACM
     * Transactions on Mathematical Software, 17:98-111 (1991)
     *)
    (#  iseed1, iseed2: @integer
    enter (iseed1, iseed2)
    #);

getsd: external
    (*
     * GET SeeD
     *      Returns the value of two integer seeds of the current
     *      generator
     * Arguments
     *      iseed1 <- First integer seed of generator G
     *      iseed2 <- Second integer seed of generator G
     * Method
     *      This is a transcription from Pascal to Fortran of routine
     * Get_State from the paper L'Ecuyer, P. and Cote, S. "Implementing
     * a Random Number Package with Splitting Facilities."  ACM
     * Transactions on Mathematical Software, 17:98-111 (1991)
     *)
    (# iseed1, iseed2: @integer
    enter (iseed1, iseed2)
    #);

setall: external
    (*
     * SET ALL random number generators
     *      Sets the initial seed of generator 1 to ISEED1 and
     * ISEED2. The initial seeds of the other generators are set
     * accordingly, and all generators states are set to these seeds.
     *
     * Arguments
```

```
   *      iseed1 -> First of two integer seeds
   *      iseed2 -> Second of two integer seeds
   *
   * Method
   *      This is a transcription from Pascal to Fortran of routine
   * Set_Initial_Seed from the paper L'Ecuyer, P. and Cote,
   * S. "Implementing a Random Number Package with Splitting
   * Facilities."  ACM Transactions on Mathematical Software,
   * 17:98-111 (1991)
   *)
  (# iseed1, iseed2: @integer
  enter (iseed1, iseed2)
  #);

setant: external
  (*
   * SET ANTithetic Sets whether the current generator produces
   *      antithetic values.  If X is the value normally returned from
   *      a uniform [0,1] random number generator then 1 - X is the
   *      antithetic value. If X is the value normally returned from a
   *      uniform [0,N] random number generator then N - 1 - X is the
   *      antithetic value. All generators are initialized to NOT
   *      generate antithetic values.
   * Arguments
   *      qvalue -> nonzero if generator G is to generating antithetic
   *                values, otherwise zero
   * Method
   *      This is a transcription from Pascal to Fortran of routine
   * Set_Antithetic from the paper L'Ecuyer, P. and Cote,
   * S. "Implementing a Random Number Package with Splitting
   * Facilities."  ACM Transactions on Mathematical Software,
   * 17:98-111 (1991)
   *)
  (# qvalue: @integer
  enter qvalue
  #);

advnst: external
  (*
   * ADV-a-N-ce ST-ate
   *      Advances the state of the current generator by 2^K values
   *      and resets the initial seed to that value.
   * Arguments
   *      k -> The generator is advanced by2^K values
   * Method: routine Advance_State from the paper L'Ecuyer, P. and
   *      Cote, S. "Implementing a Random Number Package with
   *      Splitting Facilities."  ACM Transactions on Mathematical
   *      Software, 17:98-111 (1991)
   *)
  (# k: @integer
  enter k
  #);

(*==================================================================*)
```

```
(*===== Integer random number generators ==========================*)
(*=================================================================*)

ignlgi: external
  (*
   * Integer GeNerate LarGe Integer
   *      Returns a random integer following a uniform distribution
   * over (1, 2147483562) using the current generator.
   *
   * Method
   *      This is a transcription from Pascal to Fortran of routine
   * Random from the paper L'Ecuyer, P. and Cote, S. "Implementing a
   * Random Number Package with Splitting Facilities."  ACM
   * Transactions on Mathematical Software, 17:98-111 (1991)
   *)
  (# random: @integer
  exit random
  #);

ignuin: external
  (*
   * GeNerate Uniform INteger
   *      Generates an integer uniformly distributed between LOW and
   * HIGH.
   *
   * Arguments
   *      low --> Low bound (inclusive) on integer value to be
   *              generated
   *      high --> High bound (inclusive) on integer value to be
   *               generated
   *
   * Note
   *      If (HIGH-LOW) > 2,147,483,561 prints error message on * unit
   * and stops the program.
   *)
  (# low, high: @integer;
     random: @integer
  enter (low, high)
  exit random
  #);

ignbin: external
  (*
   * Integer GeNerate BINomial random deviate
   *      Generates a single random deviate from a binomial
   * distribution whose number of trials is N and whose probability of
   * an event in each trial is P.
   *
   * Arguments
   *      n --> The number of trials in the binomial distribution from
   *            which a random deviate is to be generated.
   *      p --> The probability of an event in each trial of the
   *            binomial distribution from which a random deviate is
   *            to be generated.
   *      ignbin <-- A random deviate yielding the number of events
   *            from N independent trials, each of which has a
   *            probability of event P.
   * Method
   *      This is algorithm BTPE from: Kachitvichyanukul, V. and
   * Schmeiser, B.  W.  Binomial Random Variate Generation.
   * Communications of the ACM, 31, 2 (February, 1988) 216.
   *)
  (# n: @integer;
     p: @real;
     random: @integer
  enter (n, p)
  exit random
  #);

ignnbn: external
  (*
   * Integer GeNerate Negative BiNomial random deviate
   *      Generates a single random deviate from a negative binomial
   * distribution.
   *
   * Arguments
   *      N --> The number of trials in the negative binomial
   *            distribution from which a random deviate is to be
   *            generated.
   *      P --> The probability of an event.
   * Method
   *      Algorithm from page 480 of Devroye, Luc, Non-Uniform Random
   * Variate Generation.  Springer-Verlag, New York, 1986.
   *)
  (# n: @integer;
     p: @real;
     random: @integer
  enter (n, p)
  exit random
  #);

ignpoi: external
  (*
   * GENerate POIsson random deviate
   *      Generates a single random deviate from a Poisson
   *      distribution with mean AV.
   *
   * Arguments
   *      mu --> The mean of the Poisson distribution from which a
   *             random deviate is to be generated.
   *
   * Method
   *      Renames KPOIS from TOMS as slightly modified by BWB to use
   * RANF instead of SUNIF.  For details see: Ahrens, J.H. and Dieter,
   * U.  Computer Generation of Poisson Deviates From Modified Normal
   * Distributions.  ACM Trans. Math. Software, 8, 2 (June
   * 1982),163-179
   *)
  (# mu: @real;
     random: @integer
  enter mu
  exit random
  #);

(*=================================================================*)
(*===== Real random number generators =============================*)
(*=================================================================*)

ranf: external
  (*
   * RANnom number generator as a Function
   *      Returns a random floating point number from a uniform
   * distribution over 0 - 1 (endpoints of this interval are not
   * returned) using the current generator
   *
   * Method
   *      This is a transcription from Pascal to Fortran of routine
   * Uniform_01 from the paper L'Ecuyer, P. and Cote, S. "Implementing
   * a Random Number Package with Splitting Facilities."  ACM
   * Transactions on Mathematical Software, 17:98-111 (1991)
   *)
  (# random: @real
  exit random
  #);

genunf: external
  (*
   * GENerate UNiForm real
   *      Generates a real uniformly distributed between LOW and HIGH.
   *
   * Arguments low --> Low bound (exclusive) on real value to be
   *      generated high --> High bound (exclusive) on real value to
   *      be generated
   *)
  (# low, high: @real;
     random: @real
  enter (low, high)
  exit random
  #);

genbet: external
  (*
   * GeNerate BETa random deviate
   *      Returns a single random deviate from the beta distribution
   * with parameters A and B.  The density of the beta is x^(a-1) *
   * (1-x)^(b-1) / B(a,b) for 0 < x < 1
   *
   * Arguments
   *      aa --> First parameter of the beta distribution
   *      bb --> Second parameter of the beta distribution Method
   *
   * Method: described in R. C. H. Cheng Generating Beta Variatew with
   *      Nonintegral Shape Parameters Communications of the ACM,
   *      21:317-322 (1978) (Algorithms BB and BC)
   *)
  (# aa, bb: @real;
     random: @real
  enter (aa,bb)
  exit random
  #);

genchi: external
  (*
   * GENerate random value of CHIsquare variable
   *      Generates random deviate from the distribution of a
   * chisquare with DF degrees of freedom random variable.
   *
   * Arguments
   *      df --> Degrees of freedom of the chisquare (Must be
   * positive)
   *
   * Method:
   *      Uses relation between chisquare and gamma.
   *)
  (# df: @real;
     random: @real
  enter df
  exit random
  #);

genexp: external
  (*
   * GENerate EXPonential random deviate
   *      Generates a single random deviate from an exponential
   * distribution with mean AV.
   *
   * Arguments
   *      av --> The mean of the exponential distribution from which a
   *             random deviate is to be generated.  Method:
   *
   * Renames SEXPO from TOMS as slightly modified by BWB to use
   * RANF instead of SUNIF.
   * For details see: Ahrens, J.H. and Dieter,
   * U.  Computer Methods for Sampling From the Exponential and Normal
   * Distributions.  Comm.  ACM, 15,10 (Oct. 1972), 873 - 882.
   *)
  (# av: @real;
     random: @real
  enter av
  exit random
  #);

genf: external
  (*
   * GENerate random deviate from the F distribution
   *      Generates a random deviate from the F (variance ratio)
```

```
     * distribution with DFN degrees of freedom in the numerator and DFD
     * degrees of freedom in the denominator.
     *
     * Arguments
     *      dfn --> Numerator degrees of freedom (Must be positive)
     *      dfd --> Denominator degrees of freedom (Must be positive)
     *
     * Method
     *      Directly generates ratio of chisquare variates
     *)
    (# dfn, dfd: @real;
       random: @real
    enter (dfn, dfd)
    exit random
    #);

  gengam: external
    (*  GENerates random deviates from GAMma distribution
     *      Generates random deviates from the gamma distribution whose
     * density is (A**R)/Gamma(R) * X**(R-1) * Exp(-A*X)
     *
     * Arguments
     *      a --> Location parameter of Gamma distribution
     *      r --> Shape parameter of Gamma distribution
     *
     * Method
     *      Renames SGAMMA from TOMS as slightly modified by BWB to use
     * RANF instead of SUNIF.  For details see: (Case R >= 1.0) Ahrens,
     * J.H. and Dieter, U.  Generating Gamma Variates by a Modified
     * Rejection Technique.  Comm. ACM, 25,1 (Jan. 1982), 47 - 54.
     * Algorithm GD (Case 0.0 <= R <= 1.0) Ahrens, J.H. and Dieter, U.
     * Computer Methods for Sampling from Gamma, Beta, Poisson and
     * Binomial Distributions.  Computing, 12 (1974), 223-246/ Adapted
     * algorithm GS.
     *)
    (# a, r: @real;
       random: @real
    enter (a, r)
    exit random
    #);

  gennch: external
    (*  GENerate random value of Noncentral CHisquare variable
     *      Generates random deviate from the distribution of a
     * noncentral chisquare with DF degrees of freedom and noncentrality
     * parameter xnonc.
     *
     * Arguments
     *      df --> Degrees of freedom of the chisquare (Must be > 1.0)
     *      xnonc --> Noncentrality parameter of the chisquare (Must be
     *                >= 0.0)
     * Method
     *      Uses fact that noncentral chisquare is the sum of a
     * chisquare deviate with DF-1 degrees of freedom plus the square of
```

```
   * a normal deviate with mean XNONC and standard deviation 1.
   *)
  (# df, xnonc: @real;
     random: @real
  enter (df, xnonc)
  exit random
  #);

gennf: external
  (*  GENerate random deviate from the Noncentral F distribution
   *      Generates a random deviate from the noncentral F (variance
   * ratio) distribution with DFN degrees of freedom in the numerator,
   * and DFD degrees of freedom in the denominator, and noncentrality
   * parameter XNONC.
   *
   * Arguments
   *      dfn --> Numerator degrees of freedom (Must be >= 1.0)
   *      dfd --> Denominator degrees of freedom (Must be positive)
   *      xnonc --> Noncentrality parameter (Must be nonnegative)
   *
   * Method
   *      Directly generates ratio of noncentral numerator chisquare
   * variate to central denominator chisquare variate.
   *)
  (# dfn, dfd, xnonc: @real;
     random: @real
  enter (dfn, dfd, xnonc)
  exit random
  #);

gennor: external
  (*
   * GENerate random deviate from a NORmal distribution
   *      Generates a single random deviate from a normal distribution
   * with mean, AV, and standard deviation, SD.
   *
   * Arguments
   *      av --> Mean of the normal distribution.
   *      sd --> Standard deviation of the normal distribution.
   *
   * Method
   *      Renames SNORM from TOMS as slightly modified by BWB to use
   * RANF instead of SUNIF.  For details see: Ahrens, J.H. and Dieter,
   * U.  Extensions of Forsythe's Method for Random Sampling from the
   * Normal Distribution.  Math. Comput., 27,124 (Oct. 1973), 927 -
   * 937.
   *)
  (# av, sd: @real;
     random: @real
  enter (av, sd)
  exit random
  #);

sexpo: external
```

```
  (*
   * Standard EXPonential distribution
   *
   * Method
   *      For details see: Ahrens, J.H. and Dieter, U.  Computer
   * Methods For Sampling From The Exponential And Normal
   * Distributions.  COMM. ACM, 15,10 (Oct. 1972), 873 - 882.  All
   * statement numbers correspond to the steps of algorithm 'SA' in
   * the above paper (slightly modified implementation) Modified by
   * Barry W. Brown, Feb 3, 1988 to use RANF instead of SUNIF.  The
   * argument IR thus goes away.
   *
   *      Q(N) = SUM(ALOG(2.0)**K/K!)  K=1,..,N , The highest N (here
   * 8) is determined by Q(N)=1.0 within standard precision
   *)
  (# random: @real
  exit random
  #);

sgamma: external
  (*
   * Standard GAMMA distribution
   *
   *      Sample from the GAMMA-(A)-distribution coefficients Q(K)
   *         - for Q0 = SUM(Q(K)*A**(-K)) coefficients A(K)
   *         - for Q = Q0+(T*T/2)*SUM(A(K)*V**K) coefficients E(K)
   *         - for EXP(Q)-1 = SUM(E(K)*Q**K)
   *
   * Arguments
   *      A ---> Parameter (mean) of the standard GAMMA distribution
   * Method
   *      CASE A >= 1.0 !
   *      For details see: Ahrens, J.H. and Dieter, U.  Generating
   * GAMMA variates by a modified rejection technique.  COMM. ACM,
   * 25,1 (Jan.  1982), 47 - 54.  Step numbers correspond to algorithm
   * 'GD' in the above paper (straightforward implementation) Modified
   * by Barry W. Brown, Feb 3, 1988 to use RANF instead of SUNIF.  The
   * argument IR thus goes away.
   *      CASE 0.0 < A < 1.0 !
   *      For details see: Ahrens, J.H. and Dieter, U.  Computer
   * Methods for Sampling from GAMMA, BETA, Poisson and Binomial
   * Distributions.  COMPUTING, 12 (1974), 223 - 246.  (adapted
   * implementation of algorithm 'GS' in the above paper)
   *)
  (# a: @real;
     random: @real
  enter a
  exit random
  #);

snorm: external
  (*
   * Standard NORmal distribution
   *
```

```
 * Method
 *      For details see: Ahrens, J.H. and Dieter, U.  Extensions of
 * Forsythe's method for Random Sampling from the NORMAL
 * distribution.  MATH.  COMPUT., 27,124 (OCT. 1973), 927 - 937.
 * All statement numbers correspond to the steps of algorithm 'FL'
 * (M=5) in the above paper (slightly modified implementation)
 * Modified by Barry W. Brown, Feb 3, 1988 to use RANF instead of
 * SUNIF.  The argument IR thus goes away.
 *)
(# random: @real
exit random
#);
```

7. The regexp Library

The regexp library augments the text pattern in betaenv with four new attributes:

```
regexp_match
regexp_search
regexp_replace
regexp_replace_literally
```

All four operations gives facilities for working with regular expressions in text strings.

Regular expression

A *regular expression* (regexp, for short) is a pattern that denotes a set of strings, possibly an infinite set. Searching for matches for a regexp is a very powerful operation that editors on Unix systems have traditionally offered.

Regular expressions are used to locate occurrences of substrings in text, where the substring is expected to be of a certain structure. E.g. the regexp `'[Ww]ord'` will match the substring `'word'` or `'Word'`. The four operations offers slightly different possibilities, described below:

7.1.1 regexp_match

Takes a regexp as enter parameter (in the form of a reference to a text, containing the regexp. Matches `THIS(text)` against the regexp. `INNER` is executed if `THIS(text)` matches the regexp, and the virtual notification `noMatch` is invoked otherwise. Returns `true` if a match is found, `false` otherwise. The regexp must be found starting at the current position of `THIS(text)`.

7.1.2 regexp_search

Like `regexp_match`, except that the match is allowed to be found anywhere between the current position and the end of `THIS(text)`.

7.1.3 regexp_replace

Like `regexp_search`, except that it takes a second enter parameter, `replacement_string`. `Regexp_replace` searches for the regexp, and replaces the matched substring of `THIS(text)` with the replacement string. The replacement string may contain \0, \1, ..., \9, representing the substring matched by the i'th parenthesis in the regexp. \0 represents the entire substring matched. `INNER` is executed after the replace have taken place.

7.1.4 regexp_replace_literally

Like `regexp_replace`, except that the replacement string is taken literally (i.e. \0, \1, etc. are not representing any matched substrings)

All four regexp operations defines the same local attributes:

`start`:	start position for search in `THIS(text)`. Default: `pos`
`limit`:	end position for search in `THIS(text)`. Default: `length`
`posToMatchEnd`:	if `true`, move `THIS(text).pos` to the end of the matched substring. Default: `false`
`regs`:	structure for getting access to the matched substring. `noMatch`: invoked if no matches are found.
`regexpError`:	is invoked if error occurs in the regexp implementation.
`value`:	`true`, if any match is found.

`Regexp_string` are compiled implicitly by the operations to ensure efficient matching in all operations. If the same `regexp_string` is to be used several times, the repetition of this compilation can be avoided by generating an instance of the operation (say `regexp_search`), and then use that instance repeatedly (as is illustrated in the following example):

```
(# regexp_s: @mytext.regexp_search(# ... #);
do ...
   regexp_string[]->regexp_s;(* first search, implicit regexp
                              * compilation *)
   ...
   regexp_s; (* repeating the search with the same regexp.
              * No regexp compilation *)
   ...
   regexp_s; (* repeating the search with the same regexp.
              * No regexp compilation *)
   ...
#)
```

7.1.5 Syntax of Regular Expressions

The syntax of regular expressions in this library follows the syntax of Emacs regular expressions. Some of the documentation below is from the interactive Emacs Info system.

Regular expressions have a syntax in which a few characters are special constructs and the rest are *ordinary*. An ordinary character is a simple regular expression which matches that character and nothing else. The special characters are `'$'`, `'^'`, `'.'`, `'*'`, `'+'`, `'?'`, `'['`, `']'` and `'\'`; no new special characters will be defined. Any other character appearing in a regular expression is ordinary, unless `'\'` precedes it.

For example, `'f'` is not a special character, so it is ordinary, and therefore `'f'` is a regular expression that matches the string `'f'` and no other string. (It does not match the string `'ff'`.) Likewise, `'o'` is a regular expression that matches only `'o'`.

Any two regular expressions A and B can be concatenated. The result is a regular expression which matches a string if A matches some amount of the beginning of that string and B matches the rest of the string.

As a simple example, we can concatenate the regular expressions `'f'` and `'o'` to get the regular expression `'fo'`, which matches only the string `'fo'`. Still trivial. To do something nontrivial, you need to use one of the special characters. Here is a list of them.

List of special characters

`'.'` is a special character that matches any single character except a newline. Using concatenation, we can make regular expressions like `'a.b'` which matches any three-character string which begins with `'a'` and ends with `'b'`.

`'*'` is not a construct by itself; it is a suffix, which means the preceding regular expression is to be repeated as many times as possible. In `'fo*'`, the `'*'` applies to the `'o'`, so `'fo*'` matches one `'f'` followed by any number of `'o'`s. The case of zero `'o'`s is allowed: `'fo*'` does match `'f'`.

`'*'` always applies to the smallest possible preceding expression. Thus, `'fo*'` has a repeating `'o'`, not a repeating `'fo'`.

The matcher processes a `'*'` construct by matching, immediately, as many repetitions as can be found. Then it continues with the rest of the pattern. If that fails, backtracking occurs, discarding some of the matches of the `'*'`-modified construct in case that makes it possible to match the rest of the pattern. For example, matching `'ca*ar'` against the string `'caaar'`, the `'a*'` first tries to match all three `'a'`s; but the rest of the pattern is `'ar'` and there is only `'r'` left to match, so this try fails. The next alternative is for `'a*'` to match only two `'a'`s. With this choice, the rest of the regexp matches successfully.

`'+'` Is a suffix character similar to `'*'` except that it requires that the preceding expression be matched at least once. So, for example, `'ca+r'` will match the strings 'car' and 'caaaar' but not the string `'cr'`, whereas `'ca*r'` would match all three strings.

`'?'` Is a suffix character similar to `'*'` except that it can match the preceding expression either once or not at all. For example, `'ca?r'` will match `'car'` or `'cr'`; nothing else.

`'[...]'` `'['` begins a *character set*, which is terminated by a `']'`. In the simplest case, the characters between the two form the set. Thus, `'[ad]'` matches either one `'a'` or one `'d'`, and `'[ad]*'` matches any string composed of just `'a'`s and `'d'`s (including the empty string), from which it follows that `'c[ad]*r'` matches `'cr'`, `'car'`, `'cdr'`, `'caddaar'`, etc.

Character ranges can also be included in a character set, by writing two characters with a `'-'` between them. Thus, `'[a-z]'` matches any lower-case letter. Ranges may be intermixed freely with individual characters, as in `'[a-z$%.]'`, which matches any lower case letter or `'$'`, `'%'` or period.

Note that the usual special characters are not special any more inside a character set. A completely different set of special characters exists inside character sets: `']'`, `'-'` and `'^'`.

To include a `']'` in a character set, you must make it the first character. For example, `'[]a]'` matches `']'` or `'a'`. To include a `'-'`, write `'--'`, which is a range containing only `'-'`. To include `'^'`, make it other than the first character in the set.

`'[^...]'` `'[^'` begins a *complement character set*, which matches any character except the ones specified. Thus, `'[^a-z0-9A-Z]'` matches all characters except letters and digits.

`'^'` is not special in a character set unless it is the first character. The character following the `'^'` is treated as if it were first (`'-'` and `']'` are not special there).

Note that a complement character set can match a newline, unless newline is mentioned as one of the characters not to match.

`'^'` is a special character that matches the empty string, but only if at the beginning of a line in the text being matched. Otherwise it fails to match anything. Thus, `'^foo'` matches a `'foo'` which occurs at the beginning of a line.

`'$'` is similar to `'^'` but matches only at the end of a line. Thus, `'xx*$'` matches a string of one`'x'` or more at the end of a line.

`'\'` has two functions: it quotes the special characters (including `'\'`), and it introduces additional special constructs.

Because `'\'` quotes special characters, `'\$'` is a regular expression which matches only `'$'`, and `'\['` is a regular expression which matches only`'['`, and so on.

Note, that `'\'` *is also a special character in BETA literals. This implies, that in order to specify a* `'\'` *regexp special character in a BETA string literal, you have to type it twice, e.g.*`'US\\$'`.

Note: for historical compatibility, special characters are treated as ordinary ones if they are in contexts where their special meanings make no sense. For example, `'*foo'` treats `'*'` as ordinary since there is no preceding expression on which the `'*'` can act. It is poor practice to depend on this behavior; better to quote the special character anyway, regardless of where is appears.

For the most part, `'\'` followed by any character matches only that character. However, there are several exceptions: characters which, when preceded by `'\'`, are special constructs. Such characters are always ordinary when encountered on their own. Here is a table of`'\'` constructs.

Table of `'\'` constructs

`'\|'` specifies an alternative. Two regular expressions A and B with `'\|'` in between form an expression that matches anything that either A or B will match.

Thus, `'foo\|bar'` matches either`'foo'` or `'bar'` but no other string.

`'\|'` applies to the largest possible surrounding expressions. Only a surrounding `'\( ... \)'` grouping can limit the grouping power of `'\|'`.

Full backtracking capability exists to handle multiple uses of `'\|'`.

`'\(...\)'` is a grouping construct that serves three purposes:

1. To enclose a set of `'\|'` alternatives for other operations. Thus, `'\(foo\|bar\)x'` matches either`'foox'` or `'barx'`.
2. To enclose a complicated expression for the postfix `'*'` to operate on. Thus, `'ba\(na\)*'` matches `'bananana'`, etc., with any (zero or more) number of`'na'` strings.
3. To mark a matched substring for future reference.

This last application is not a consequence of the idea of a parenthetical grouping; it is a separate feature which happens to be assigned as a second meaning to the same `'\( ... \)'` construct because there is no conflict in practice between the two meanings. Here is an explanation of this feature:

`'\DIGIT'` after the end of a `'\( ... \)'` construct, the matcher remembers the beginning and end of the text matched by that construct. Then, later on in the regular expression, you can use `'\'` followed by `DIGIT` to mean “match the same text matched the `DIGIT`’th time by the `'\( ... \)'` construct.”

The strings matching the first nine `'\( ... \)'` constructs appearing in a regular expression are assigned numbers 1 through 9 in order that the open-parentheses appear in the regular expression. `'\1'` through `'\9'` may be used to refer to the text matched by the corresponding `'\( ... \)'` construct.

For example, `'\(.*\)\1'` matches any newline-free string that is composed of two identical halves. The `'\(.*\)'` matches the first half, which may be anything, but the `'\1'` that follows must match the same exact text.

`'\''` matches the empty string, provided it is at the beginning of the buffer.

`'\''` matches the empty string, provided it is at the end of the buffer.

`'\b'` matches the empty string, provided it is at the beginning or end of a word. Thus, `'\bfoo\b'` matches any occurrence of `'foo'` as a separate word. `'\bballs?\b'` matches `'ball'` or `'balls'` as a separate word.

`'\B'` matches the empty string, provided it is not at the beginning or end of a word.

`'\<'` matches the empty string, provided it is at the beginning of a word.

`'\>'` matches the empty string, provided it is at the end of a word.

`'\w'` matches any word-constituent character. The editor syntax table determines which characters these are.

`'\W'` matches any character that is not a word-constituent.

`'\sCODE'` matches any character whose syntax is `CODE`. `CODE` is a character which represents a syntax code: thus, `'w'` for word constituent, `'-'` for whitespace, `'('` for open-parenthesis, etc.

`'\SCODE'` matches any character whose syntax is not `CODE`.

A complicated regexp

Here is a complicated regexp, used to recognize the end of a sentence together with any whitespace that follows. It is given in BETA text string syntax to enable you to distinguish the spaces from the tab characters. In BETA text string syntax, the string constant begins and ends with a double-quote. `'""'` stands for a double-quote as part of the regexp, `'\\'` for a backslash as part of the regexp, `'\t'` for a tab and `'\n'` for a newline.

```
"[.?!][]""')]*\\($\\|\t\\| \\)[ \t\n]*"
```

This contains four parts in succession: a character set matching period, `'?'` or `'!'`; a character set matching close-brackets, quotes or parentheses, repeated any number of times; an alternative in backslash-parentheses that matches end-of-line, a tab or two spaces; and a character set matching whitespace characters, repeated any number of times.

7.2 Regular Expression Registers

The special regexp parenthesis (i.e. `\( ... \)`) are used for three different purposes:

(1) to make the regexp operators work on entire sub-regexp: e.g.

`123*4` matches `124`, `1234`, `12334`, `123334`, etc., whereas `1\(23\)*4` matches `124`, `1234`, `123234`, `12323234`, etc. (and not `12334`, `123334`, etc.).

(2) to delimit alternatives: e.g.`1\(2\|3\)4` matches `124` and `134`.

(3) to mark a matched substring for future reference. In `regexg_replace`, `\0`, `\1`, `\2`, ..., `\9` substrings will be replaced with the matched substring. We will here give a short introduction to accessing these registers from a BETA program.

We would like to give a little more details on the (3) purpose. Let us assume, that we invoke:

```
'sum: \([0-9]+\)\.\([0-9]\) US$'
   -> t.regexp_search
        (#
        do 'The amount is: '->puttext;
           (1->regs.start+1,1->regs.end)->t.sub->puttext;
           ' dollars and '->puttext;
           (2->regs.start+1,2->regs.end)->t.sub->puttext;
           ' cents'->putline
        #)
```

The intent is to find some amount `i` the `t` text variable. The amount should be found in a context of the form: `'sum: XXX.YYY US$'`, where `XXX` and `YYY` may be any number. If found, the amount will be written on standard screen in the form: `'The amount is: XXX dollars and YYY cents'` .

Note the use of `i->regs.start` and `i->regs.end`, which returns the character positions of the i'th matched substring (in this case the 1'st and 2'nd substring, containing the amount of dollars and the amount of cents, respectively). As it can be seen, these constructs can be used within the action-parts of all regexp operations to gain access to the matched substrings (note that the 0'th substring is the entire string matched).

7.3 Known bugs or Inconveniences

1) `\i (i= 0, 1, ...,9)` matches the empty string if it is positioned *before* the i'th parenthesis. According to the specification, `\i` should match `'i'` if positioned before the i'th parenthesis. E.g.`\1` matches the empty string in`\1\(2\)3`.

2) There is a slight error in the current implementation of registers, when the parenthesis bounds a repetitive regexp, i.e. a `'*'` or `'+'` regexp. In this case, the register will not contain the correct character positions, but instead refer to the empty string, immediately after the matched substring. Use an extra pair of parenthesis to get the correct substring (e.g. use `1\(\(2\)*\)3` instead of `1\(2\)*3` to refer to the matched `'2'`s - the matched `'2'`s can be referred to through register 1.

7.4 Using the regexp Fragment

A program using the`regexp` fragment will have the following structure:

```
INCLUDE '~beta/basiclib/v1.6/regexp
--- program: descriptor ---
(# ...
   t: ^text
do ...
   'Hello world' -> t[];
   ('\\<w.*\\>', '\\0 champion') -> t.regexp_replace;
   t[] -> putline; (* prints 'Hello world champion' *)
   ...
#)
```

7.5 Regexp Demos

7.5.1 Search for regexp

The following example is a simple program that enters a regexp from the keyboard, followed by another test string, trying to find the regexp in the test string. If found, the registers are printed on the screen, and the corresponding matched substrings:

searchDemo.bet

```
ORIGIN '~beta/basiclib/v1.6/regexp';
--- program: descriptor ---
(#
do loop:
     (# regex: ^text;
     do 'Search for: ' -> puttext; getLine -> regex[];
        (if regex.length = 0 then leave loop if);
        loop:
          (# string: ^text;
          do 'Search for: '->puttext; regex[]->puttext;
             ' in: ' -> puttext; getline -> string[];
             (if string.length = 0 then leave loop if);
             0 -> string.pos;
             regex[] -> string.regexp_search
             (# noMatch::< (# do '\tNo match' -> putline #)
             do (for i: regexp_numberOfRegisters repeat
                     (if (i-1->regs.start)>=0 then
                         '\tregister '->puttext; i-1 -> putint; ': '->puttext;
                         i-1->regs.start->putint;
                         ' to '->puttext;
                         i-1->regs.end->putint;
```

```
                        ': '->puttext;
                        ((i-1->regs.start)+1, i-1->regs.end)->string.sub->putline
                    if)
               for);
            #);
            restart loop
         #);
         restart loop
     #)
#)
```

Output from the demo might be:

```
Search for: 1\(234\)5
Search for: 1\(234\)5 in: dfjdfhdf1235jfdkj12345ghdf
        register 0: 17 to 22: 12345
        register 1: 18 to 21: 234
Search for: 1\(234\)5 in: 12345
        register 0: 0 to 5: 12345
        register 1: 1 to 4: 234
Search for: 1\(234\)5 in: 123
        No match
Search for: 1\(234\)5 in:

Search for: \(22\).*\1
Search for: \(22\).*\1 in: xxx22yyy2zzz
        No match
Search for: \(22\).*\1 in: xxx22yyy22zzz
        register 0: 3 to 10: 22yyy22
        register 1: 3 to 5: 22
```

7.5.2 Replace with a regexp

The following example is nearly identical to the previous, except that it allows for the specification of a replacement string:

replaceDemo.bet

```
ORIGIN '~beta/basiclib/v1.6/regexp';
--- program: descriptor ---
(#
do loop:
     (# regex: ^text;
     do 'Replace: ' -> puttext; getLine -> regex[];
        (if regex.length = 0 then leave loop if);
        loop:
          (# string: ^text;
          do 'Replace: '->puttext; regex[]->puttext;
             ' in: ' -> puttext; getline -> string[];
             (if string.length = 0 then leave loop if);
             loop:
               (# t, replacement: ^text
               do string.copy -> t[]; 0 -> t.pos;
                  'Replace: '->puttext; regex[]->puttext;
                  ' in: ' -> puttext; string[]->puttext;
                  ' with: '->puttext; getline -> replacement[];
                  (if replacement.length = 0 then leave loop if);
                  (regex[], replacement[]) -> t.regexp_replace
                  (# noMatch::< (# do '\tNo match'->screen.putline #) #);
                  'Replace: '->puttext; regex[]->puttext;
                  ' in: ' -> puttext; string[]->puttext;
                  ' with: '->puttext; replacement[]->puttext;
                  ' gives: '->puttext; t[] -> putline;
                  restart loop
               #);
             restart loop
          #);
        restart loop
     #)
#)
```

Output from the demo might be:

```
Replace: [Ww]ord
Replace: [Ww]ord in: This Word is in capital
Replace: [Ww]ord in: This Word is in capital with: sentence
Replace: [Ww]ord in: This Word is in capital with: sentence gives: This sentence
is in capital
```

7.6 Interface Description for the Regexp Library

```
ORIGIN 'betaenv';
BODY 'private/regexplib';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1992-96
 *       All rights reserved.
 *)
--- textlib: attributes ---
regexp_operation:
  (* generic  superpattern: for all regexp text operations:
   *     regexp_match, regexp_search, regexp_replace,
   *     regexp_replace_literally.
   * regexp_string: text string containing the regexp.
   * start: start position for match in THIS(text).
   *        Default: pos
   * limit: end position for match in THIS(text).
   *        Default: length
   * posToMatchEnd: if true, move THIS(text).pos to the end of the
   *                matched substring.
   *        Default: false
   * regs: structure for getting access to the matched substring.
   * noMatch: invoked if no matches are found.
   * regexpError: is invoked if syntax error occurs in the specified
   * regexp.
   *
   * value: true, if any match is found.
   *)
  (# regexp_string: ^text;
     start:< integerObject(# do pos -> value; INNER #);
     limit:< integerObject(# do length -> value; INNER #);
     posToMatchEnd:< booleanObject;
     regs: @regexp_registers;
     noMatch:< Notification;
     regexpError:< Exception(# do 'Syntax error in regular expression'-
>msg #);
     value: @boolean;
     private: @(* private *)...
  enter (#
        enter regexp_string[]
        do ...
        #)
  do INNER regexp_operation
  exit value
  #);
```

```
regexp_match: regexp_operation
(* Takes a regexp as enter parameter (in the form of a reference to a
 * text, containing the regexp.  Matches THIS(text) against the
 * regexp.  INNER is executed if THIS(text) matches the regexp, and
 * the virtual notification noMatch is invoked otherwise.  Returns
 * true if a match is found, false otherwise.  The regexp must be
 * found starting at the current position of THIS(text).
 *)
(# do ... #);

regexp_search: regexp_operation
(* Like regexp_match, except that the match is allowed to be found
 * anywhere between the current position and the end of THIS(text).
 *)
(# do ... #);

regexp_replace: regexp_search
(* Like regexp_search, except that it takes a second enter parameter,
 * replace_string.  Regexp_replace searches for the regexp, and
 * replaces the matched substring of THIS(text) with the replacement
 * string.  The replacement string may contain , , ..., ,
 * representing the substring matched by the i'th parenthesis in the
 * regexp.   represents the entire substring matched.  INNER is
 * executed after the replace have taken place.
 *)
(# replace_string: ^text;
enter replace_string[]
do ...
#);

regexp_replace_global:
  (* replaces all occurences of m with r in THIS(text) using
   * regexp_replace, starting from THIS(text).pos.
   *)
  (# m,r: ^Text; more: @boolean;
     replaceOp: @regexp_replace
     (# noMatch:: (# do false->more #);
        posToMatchEnd:: (# do true->value #)
     #);
  enter (m[],r[])
  do true->more;
     (m[],r[])->replaceOp;
     loop:
       (if more and not eos then
           replaceOp;
           restart loop
       if)
  #);

regexp_replace_literally: regexp_search
(* Like regexp_replace, except that the replacement string is taken
 * literally (i.e , , etc. are not substituted with any matched
 * substrings).
 *)
(# **replace_string**: ^text
enter replace_string[]
do ...
#);

**regexp_replace_literally_global**
  (* replaces all occurences of m with r in THIS(text) using
   * regexp_replace_literally, starting from THIS(text).pos.
   *)
  (# **m**,**r**: ^Text; **more**: @boolean;
     **replaceOp**: @regexp_replace_literally
     (# noMatch:: (# do false->more #);
        posToMatchEnd:: (# do true->value #)
     #);
  enter (m[],r[])
  do true->more;
     (m[],r[])->replaceOp;
     **loop**:
       (if more and not eos then
           replaceOp;
           restart loop
       if)
  #);

--- lib: attributes ---
**regexp_numberOfRegisters** (# exit 10 #);

**regexp_registers**: Cstruct
  (* Structure for accessing the substrings matched by some regexp. *)
  (# **getRegisterValue**:
       (# regNr, value: @integer;
          pos:< integerValue;
          thePos: @pos (* private: for efficiency *)
       enter regNr
       do ...
       exit value
       #);
     **start**: @getRegisterValue
       (# pos::< (# do 0 -> value #) #);
     **end**: @getRegisterValue
       (# pos::< (# do 40 -> value #) #);
     **byteSize**::< (* private *) (# do regexp_numberOfRegisters*2*4 -> va
#)
  #)
```

8. The file Library

The file library implements the `diskEntry`, `file`, `fileRep`, patterns, all used to model external storage media such as disk files.

8.1.1 File and diskEntry

The `file` library implements the `file` pattern that is used to represent external storage media such as disk files. `File` is an abstract subpattern of `stream`, specifying the machine independent attributes of such media. Specific *subpatterns:* exists for the different machine types such as UNIX and Macintosh (`unixFile`, respectively `macFile`, see later).

The attributes of `file` are divided into two parts: the disk entry attributes and the contents related attributes. The disk entry attributes are located in the `entry` attribute of `file`, whereas the contents related attributes are ordinary attributes of `file`.

Disk entry attributes

The disk entry attributes are defined in the `diskEntry` pattern: `path`, `size`, `readable`, `writable`, `isFile`, `isDirectory`, `exists`, `modtime`, `touch`, and `rename`. Also related to the disk entry are the following exceptions: `diskEntryExistsException`, `diskEntryModtimeException`, `diskEntryTouchException`, and `diskEntryRenameException`. These attributes are accessed through the `entry` attribute of a file, e.g. if `aFile` is a file object, then `aFile.entry.modtime` will return the last modification time of the associated disk file.

File attributes

The contents related attributes are: `entry`, `name`, `touch`, `delete`, `openRead`, `openWrite`, `openAppend`, `openReadWrite`, `openReadAppend`, `flush`, and `close`. Note that `file` also inherits all the `stream` attributes (further binding several of them). `File` also defines the following exceptions: `openException`, `accessError`, `writeError`, `readError`, `noSuchFileError`, `fileExistsError`, `noSpaceError`, and `otherError`.

8.1.2 FileRep

The `fileRep` pattern is consisting of a repetition of integers and operations which makes it possible to save and restore this repetition in one chunk from a file. When saving, the repetition elements `[1:top-1]` are saved (`top` is an attribute of `fileRep`) and when restoring, `top` will become equal to range of the stored repetition. The repetition is in the `R` attribute, and `save` and `restore` is used for saving and restoring the repetition onto some file.

8.2 Using the file Fragment

A program using the`file` fragment will have the following structure:

```
INCLUDE '~beta/basiclib/v1.6/file
--- program: descriptor ---
(# ...
   f: @file;
do ...
   'Hello world' -> f.putline;
   ...
#)
```

8.2.1 Using diskEntry

An example showing the use of the `path` attribute of `DiskEntry`: The path specified on the command line is decomposed, and the various parts of it are printed:

decompose.bet

```
ORIGIN '~beta/basiclib/v1.6/file';

--- program: descriptor ---
(* An example showing the use of the 'path' attribute of DiskEntry: The path
 * specified on the command line is decomposed, and the various parts of it are
 * printed.
 *)

(# e: @diskentry;
do (if noOfArguments <> 2 then
       'Usage: ' -> puttext; 1->arguments->puttext; ' path' -> putline;
       stop;
   if);
   2 -> arguments -> e.path;
   'The path '''->puttext;
   e.path -> puttext;
   ''' is composed like this:'->putline;
   'Head of path:      '->puttext; e.path.head -> putline;
   'Name:              '->puttext; e.path.name -> putline;
   'Prefix of name:    '->puttext; e.path.name.prefix -> putline;
   'Suffix of name:    '->puttext; e.path.name.suffix -> putline;
   'Extension of name: '->puttext; e.path.name.extension -> putline;
   (if e.exists then '(and the entry exists on disk)' -> putline;
    else '(there is no such entry on disk)' -> putline;
   if);
#)
```

8.2.2 Using File

Program showing the use of exceptions in files: if the input file specified on the command line cannot be opened, the exception `noSuchFileError` is raised. In this example, it is further bound specifying that the program should continue after the exception is raised. Then a new file name is prompted for. Instead of this approach, of course, the attributes`exists` of `diskEntry` could have been used.

If the output file could not be opened, the exception `noSpaceError` is raised, and this exception is further bound to print a message:

fileerror.bet

```
ORIGIN '~beta/basiclib/v1.6/file';
--- program: descriptor ---

(* Program showing the use of exceptions in files: if the input file
 * specified on the command line cannot be opened, the exception
 * NoSuchFileError is raised. In this example, it is further bound
 * specifying that the program should continue after the exception is raised.
 * Then a new file name is prompted for.
 * Instead of this approach, of course, the attributes 'exists' of DiskEntry
 * could have been used.
 * If the output file could not be opened, the exception NoSpaceError is raised,
 * and this exception is further bound to print a message.
 *)

(# outFile: @file
     (# NoSpaceError ::<
          (# do 'It is time to delete garbage!' -> msg.putLine #);
     #);
   inFile: @ file
     (# NoSuchFileError ::< (# do true -> continue; false -> OK #);
     #);
   OK: @ boolean;

do (if noOfArguments <> 2 then
       'Usage: ' -> puttext; 1->arguments->puttext; ' out-file' -> putline;
       stop;
   if);
   2 -> arguments -> outFile.name;
   outFile.OpenWrite;

   'nofile.bet' -> inFile.name;
   true -> OK;
   openFile:
     (#
     do inFile.OpenRead;
        (if not OK then
            'File does not exist: ' ->  screen.puttext;
            infile.name -> putline;
            'Type input file name: ' ->  screen.putText;
            inFile.name.read;
            true -> OK;
            restart openFile
        if);
     #);

   readFile:
     (#
     do (if not inFile.eos then
            inFile.getatom -> outFile.putText;
            outFile.newLine;
            restart readFile
            else leave readFile
        if)
     #);
   inFile.close;
   outFile.close;
   'Tokens from '''->puttext; infile.name -> puttext;
   ''' copied to '''->puttext; outfile.name -> puttext;
   ''''->putline;
#)
```

8.3 Interface Description for the file Library

```
ORIGIN 'betaenv';
BODY 'private/filebody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1984-96
 *       All rights reserved.
 *
 * The BETA interface to disk-entries in a hierarchic file system
 * files, and directories is organised as follows:
 *
 *   DiskEntry:     Machine independent interface to entries like
 *                  file and directories on the disk
 *                  (file 'file.bet').
 *   UnixEntry:     Unix specific specialization of DiskEntry
 *                  (file 'unixfile.bet').
 *   MacEntry:      Macintosh specific specialization of DiskEntry
 *                  (file 'macfile.bet').
 *
 *   File:          Machine independent interface to disk files. Is
 *                  a specialization of Stream (file 'betaenv.bet'),
 *                  and contains a DiskEntry (file 'file.bet').
 *   UnixFile:      Unix specific specialization of File, which
 *                  contains a UnixEntry (file 'unixfile.bet').
 *   MacFile:       Macintosh  specific specialization of File,
 *                   which contains a MacEntry (file 'macfile.bet').
 *
 *   Directory:     Machine independent interface to
 *                  directories/folders. Contains a DiskEntry
 *                  (file 'directory.bet').
 *   UnixDirectory: Unix specific specialization of Directory,
 *                  which contains a UnixEntry
 *                  (file 'unixdirectory.bet').
 *   MacDirectory:  Macintosh specific specialization of Directory,
 *                  which contains a MacEntry
 *                  (file 'macdirectory.bet').
 *)
-- LIB: Attributes --
DiskEntry:
  (* Pattern describing various attributes of disk-entries like files
   * and directories in a hierarchic file system
   *)
  (# <<SLOT DiskEntryLib: attributes>>;

     (* DISK ENTRY EXCEPTIONS *)
     DiskEntryException: Exception
       (* General exception for disk entries *)
       (# ... #);
```

```
      DiskEntryExistsException: DiskEntryException
        (* Raised if a test for disk entry existence has
         * failed. Message: "Test for disk entry existence failed.",
         * and an indication of why it failed.
         *)
        (# ... #);
      DiskEntryModtimeException: DiskEntryException
        (* Raised if examination or setting of disk entry modtime has
failed.
         * Message: "Examination/setting of disk entry modtime failed.",
and an
         * indication of why it failed.
         *)
        (# ... #);
      DiskEntryTouchException: DiskEntryException
        (* Raised if touch of a disk entry has failed. Message: "Touch
         * of disk entry failed.", and an indication of why it failed.
         *)
        (# ... #);
      DiskEntryRenameException: DiskEntryException
        (* Raised if rename of a disk entry has failed. Message:
         * "Rename of disk entry failed.", and an indication of why it
         * failed.
         *)
        (# ... #);
      pathDesc:<
        (* A virtual descriptor for the full or relative path of
         * THIS(DiskEntry)
         *)
        (#  head:<
             (* The head of the path, e.g. head of '/usr/smith/foo.bet'
              * is '/usr/smith'
              *)
             (# h: ^text
             ...
             exit h[]
             #);
            nameDesc:<
             (* The actual name-part of the path, e.g. name part of
              * '/usr/smith/foo.bet' is 'foo.bet'
              *)
             (#  prefix:<
                  (* exits the prefix part of the name, i.e. what is
                   * before the last dot (.), e.g 'foo' for 'foo.bet'
                   *)
                  (# p: ^text
                  ...
                  exit p[]
                  #);
                 extension:<
                  (* exits the extension part of the name, i.e. what
                   * is after the last dot (.), e.g. 'bet' for 'foo.bet'
                   *)
                  (# e: ^text
                  ...
                  exit e[]
                  #);
                 suffix:<
                  (* like extension, but includes the dot (.),
                   * e.g. '.bet' for 'foo.bet'
                   *)
                  (# s: ^text
                  ...
                  exit s[]
                  #);
                 get:<
                  (* exits "prefix.extension" *)
                  (# n: ^text
                  ...
                  exit n[]
                  #);
             exit get
             #);
            name: @nameDesc;
            set:<
             (* set the entire path *)
             (# p: ^text
             enter p[]
             ...
             #);
            get:<
             (* get the entire path *)
             (# p: ^text
             ...
             exit p[]
             #);
        enter set
        do INNER
        exit get
        #);
      path: @pathDesc;
      exists: BooleanValue
        (* exits a boolean indicating whether the disk entry
         * corresponding to the current setting of path actually exists
         *)
        (# error:< DiskEntryExistsException;
        ...
        #);
      modtime:
        (* exits an integer denoting the (system) time of the last
         *        modification
         *)
        (# time: @integer;
           error:<DiskEntryModTimeException;
        enter (# enter time ... #)
        exit (# ... exit time #)
        #);
      touch:
        (* Updates the modtime to the current (system) time. *)
        (# error:< DiskEntryTouchException;
        ...
        #);
```

```
      rename:
        (* Rename the disk entry. Changes the physical disk entry and
         * updates THIS(DiskEntry).path
         *)
        (# newpath: ^text;
           error:< DiskEntryRenameException;
        enter newpath[]
        ...
        #);
      size: IntegerValue
        (* exits the size of THIS(DiskEntry) in bytes *)
        (# error:<DiskEntryException;
        ...
        #);
      readable: BooleanValue
        (* exits true if THIS(DiskEntry) can be read *)
        (# error:< DiskEntryException;
        ...
        #);
      writeable: BooleanValue
        (* exits true if THIS(DiskEntry) can be written to *)
        (# error:< DiskEntryException;
           checkwrite: @...;
        do checkwrite
        #);
      isFile: BooleanValue
        (* True if THIS(DiskEntry) is a regular file *)
        (# error:< DiskEntryException;
        ...
        #);
      isDirectory: BooleanValue
        (* True if THIS(DiskEntry) is a directory *)
        (# error:<DiskEntryException;
        ...
        #);

      private: @...
   do INNER
   #); (* DiskEntry *)

(* Constants used for specifying mode to File.SetPos. *)
FromBeginning:
  (* Seeks relative to the beginning of a file.  Corresponds to
   * absolute positions in File[0:File.Length-1].
   *)
  (# exit 0 #);
FromCurrent:
  (* Seeks relative to the current position.   *)
  (# exit 1 #);
FromEnd:
  (* Seeks relative to the end of a file. *)
  (# exit 2 #);

File: Stream
  (* Generalization of disk file.  Describes the stream aspects of
   * files, providing buffered I/O, and contains a DiskEntry object
   * describing the other properties of a file.
   *)
  (# <<SLOT FileLib: attributes>>;

     EntryDesc:< DiskEntry;
     Entry: @EntryDesc
       (* The item holding most characterizing attributes of
        * THIS(file)
        *);
     name: @
       (* convenient interface to entry.path *)
       (# read:
            (* Reads the file name from the Keyboard *)
            (# ... #);
       enter entry.path
       exit entry.path
       #);
     Put::<  (# ... #);
     Get::<  (# ... #);
     Peek::< (# ... #);
     PutText::< (# ... #);
     GetAtom::< (# ... #);
     GetLine::< (# ... #);
     Length::<
       (* Returns the byte size of THIS(file). Notice that this is
        * not always the same as entry.size, which is how many bytes
        * THIS(file) occupies on the disk
        *)
       (# ... #);
     GetPos::<
       (* Returns current position of THIS(File) *)
       (# ... #);
     SetPos::<
       (* Sets position on THIS(file).  Enters position and mode. See
        * above for definition of constants to use as mode,
        * FromBeginning, FromCurrent, FromEnd.  Returns the absolute
        * position seeked to [0..File.Length-1].
        *)
       (# mode,newpos: @integer
       enter mode
       ...
       exit newpos
       #);
     Eos::< (# ... #);
     touch: entry.touch
       (* If the disk entry does not exist, an empty file will be
        * created.
        *)
       (# ... #);
     delete:
       (* Deletes THIS(File) *)
       (# ... #);
     binary:< booleanvalue
       (* THIS(File) is binary if value is true. On some systems a
        * non-binary (e.g. textual) file may behave differently, than
        * a binary file. A binary file is always treated as raw bytes,
        * whereas a non-binary file may treat some characters, notably
        * the end-of-line marker, differently.
        *);
     openRead:
       (* opens THIS(File) for reading, starting at the beginning *)
       (# ... #);
     openWrite:
       (* Opens THIS(File) for writing.  truncates the contents of
        * the disk file if it already existed, and creates the disk
        * file if not
        *)
       (# ... #);
     openAppend:
       (* Opens THIS(File) for writing at the end.  Setpos cannot be
        * used to write other places than at the end.  Creates the
        * file if it did not exist.
        *)
       (# ... #);
     openReadWrite:
       (* Opens THIS(File) for both reading and writing.  The file is
        * positioned at the beginning. To switch between writing and
        * reading an intermediate setpos may be necessary.
        *)
       (# ... #);
     openReadAppend:
       (* Like OpenReadWrite, but positiones at the end *)
       (# ... #);
     flush:
       (* Flushes THIS(File). Affects only files opened for output *)
       (# ... #);
     close:  (* Closes THIS(File) *)
       (# ... #);

     (* FILE EXCEPTIONS *)

     FileException: StreamException
       (* General File exception *)
       (# m: ^text
       enter m[]
       ...
       #);
     OpenException: FileException
       (* Raised if opening of a file has failed. Message: "Cannot
        *         open file".
        *)
       (# ... #);
     AccessError:< OpenException
       (* Raised on attempt to access a file with insufficient
        * privilegies.  Message: "Insufficient access privilegies".
        *)
       (# ... #);
     WriteError:< FileException
       (* Raised from Put, PutText and Flush on attempt to write on a
        * non-existing block. Message: "Write block error".
        *)
       (# ... #);
```

```
    ReadError:< FileException
      (* Raised from Get and Peek on attempt to read a non-existing
       * block. Message: "Read block error".
       *)
      (# ... #);
    EOSerror::< (# ... #);
    NoSuchFileError:< FileException
      (* Raised on attempt to open a non-existing file. Message:
       * "File does not exist"
       *)
      (# ... #);
    FileExistsError:< FileException
      (* Raised when creating an already existing file. Message:
       * "File does already exist".
       *)
      (# ... #);
    NoSpaceError:< FileException
      (* Raised when the file system is full. Message: "File system
       * is full".
       *)
      (# ... #);
    OtherError:< FileException
      (* Raised when errors other than the above occur *)
      (# ... #);
    private: @ ...;
    getRep:
      (* OBSOLETE - only supplied for compatibility.
       * The operation getBytes from binfile should be used instead.
       *)
      (# repAdr (* @@ rep[inx]: start address *),
         length (* max. no. of elements to read *): @integer;
      enter(repAdr,length)
      ...
      exit(length div 4)
      #);
    putRep:
      (* OBSOLETE - only supplied for compatibility.
       * The operation putBytes from binfile should be used instead.
       *)
      (# repAdr (* @@rep[inx]: start address *),
         length (* no. of rep-elements to be and was written
                *): @integer;
      enter(repAdr,length)
      ...
      exit (length div 4)
      #);
 #); (* pattern File *)
```

9. The directory Library

The `directory` library defines the `directory` pattern that is the interface into file directories in a hierarchical file system. `Directory` is an abstract pattern, specifying the machine independent attributes of such directories. Specific *subpatterns:* exists for the different machine types such as UNIX and Macintosh (`unixDirectory`, respectively `macDirectory`, see later).

Similar to `file`, the attributes of `directory` are divided into two parts: the disk entry attributes and the directory related attributes. The disk entry attributes are located in the `entry` attribute of `directory`, whereas the directory related attributes are ordinary attributes of `directory`.

The disk entry attributes of `directory` are the same as for `file`, and will therefore not be discussed here (see previous chapter).

Directory attributes

The directory related attributes are: `entry`, `name`, `touch`, `delete`, `createFile`, `deleteFile`, `createDir`, `deleteDir`, `noOfEntries`, `empty`, `findEntry`, and `scanEntries`. `Directory` also defines the following exceptions: `entryExistException`, `dirScanException`, `dirSearchException`, `noSuchException`, and `notFoundException`.

9.1 Using the directory Fragment

A program using the `directory` fragment will have the following structure:

```
INCLUDE '~beta/basiclib/v1.6/directory'
--- program: descriptor ---
(# ...
   dir: @directory;
do ...
   '~beta/basiclib/v1.6/' -> dir.name;
   dir.scanEntries(# do found.name -> putline #);
   ...
#)
```

9.1.1 Listing a directory

Program showing a simple use of `directory`: The directory with the path given as argument is scanned, and the names of all the entries are printed with an indication of what type of entry it is. This is done using the `select` pattern of `scanentries`. This is a more efficient strategy than using `found.entry.isFile` etc., possibly correcting for the case that `'d'` is not current working directory.

If the path given is not a directory, an exception will be raised:

listDir.bet

```
ORIGIN '~beta/basiclib/v1.6/directory';

--- program: descriptor ---

(* Program showing a simple use of directory: The directory with the path
 * given as argument is scanned, and the names of all the entries are printed
 * with an indication of what type of entry it is.
 * This is done using the 'select' pattern of 'scanentries'. This is a more
 * efficient startegy than using 'found.entry.isFile' etc., possibly
 * correcting for the case that 'd' is not current working directory.
 * If the path given is not a directory, an exception will be raised.
 *)

(# arg: ^text;
   d: @directory;
   nl: @boolean;
   full: @boolean;
   usage:
     (# do 'Usage: ' -> puttext;
        1->arguments->puttext;
        ' [-f] path' -> putline;
        stop;
     #);
do (* Parse command line *)
   (if noOfArguments
    // 1 then '.' -> d.name
    // 2 then
       2 -> arguments -> arg[];
       (if '-f' -> arg.equal then
           true -> full;
           '.' -> d.name;
        else
           arg[] -> d.name;
       if)
    // 3 then
       2 -> arguments -> arg[];
       (if '-f' -> arg.equal then
           true -> full;
        else usage;
       if);
       3 -> arguments -> d.name;
    else
       usage;
   if);

   (* Scan directory *)
   newline;
   'The content of '''-> puttext;
   d.name -> puttext;
   ''' is: ' -> putline;
   d.scanEntries
   (#
   do select
      (# whenFile::<
           (# do 'File:      ' -> puttext; #);
         whenDir::<
           (# do 'Directory: ' -> puttext; #);
         whenOther::<
           (# do '(Unknown): ' -> puttext; #);
      #);
      (if full then foundFullPath -> putline;
       else found.path -> putline;
      if);
   #);
   newline;
```

```
  #)
```

9.2 Interface Description for the directory Library

```
ORIGIN 'file';
BODY 'private/directorybody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1984-96
 *       All rights reserved.
 *
 *)
---- LIB: attributes ----
directory:
  (* Generalization of disk folder/directory.  Describes the list
   * aspects of directories and contains a DiskEntry item describing
   * the other properties of a directory.
   *)
  (#
     <<SLOT DirectoryLib: attributes>>;

     EntryDesc:< DiskEntry;
     entry: @EntryDesc
       (* The item holding most characterizing attributes of
        * THIS(directory)
        *);
     name: @
       (* convenient interface to entry.path *)
       (# read:
            (* Reads a directory name from the Keyboard *)
            (# do ... #);
       enter entry.path
       exit entry.path
       #);

     (* Directory exceptions *)

     DirException: Exception
       (* General directory exception *)
       (# do ...; INNER #);
     EntryExistException: DirException
       (* Raised on attempt to create a file or directory that
        * allready existed in THIS(directory). Message: "Directory
        * entry allready exist"
        *)
       (# do ...; INNER #);
     DirScanException: DirException
       (* Raised if a scan of THIS(directory) has failed.  Message:
```

```
     * "Scan of directory failed.", and an indication of why it
     * failed.
     *)
    (# do ...; INNER #);
  DirSearchException: DirException
    (* Raised if a find in THIS(directory) has failed.  Message:
     * "Search of directory failed.", and an indication of why it
     * failed.
     *)
    (# do ...; INNER #);
  NoSuchException: DirException
    (* Raised on attempt to delete a file or directory that did
     * not exist in THIS(directory). Message: "Attempt to delete a
     * nonexisting entry."
     *)
    (# do ...; INNER #);
  NotFoundException: DirException
    (* Raised if findEntry.select is used in findEntry.notFound,
     * or in other situations that findEntry.found[]=NONE. Message:
     * "Attempt to use 'select' in 'findEntry' when the candidate
     * was not found."
     *)
    (# do ...; INNER #);

  (* Manipulations of THIS(directory) *)

  touch: entry.touch
    (* If the disk entry does not exist, an empty directory will
     * be created.
     *)
    (# touchD: @...;
    do touchD
    #);
  delete:
    (* Delete THIS(directory) *)
    (# nosuch:< NoSuchException
         (* Raised if there was no disk entry corresponding to
          * THIS(Directory)
          *);
       error:< entry.DiskEntryException
         (* Raised if other errors occurred *);
       deleteD: @...;
    do deleteD
    #);
  createFile:
    (* Create a file named 'name' in THIS(Directory) *)
    (# name: ^text;
       newEntry: ^EntryDesc;
       exists:< EntryExistException
         (* Raised if en entry of that name already existed *);
       error:< DirException
         (* Raised if other errors occurred *);
    enter name[]
    ...
    exit newEntry[]
    #);
```

```
deleteFile:
  (* Delete a file named 'name' in THIS(Directory) *)
  (# name: ^text;
     nosuch:< NoSuchException
       (* Raised if there was no disk entry in THIS(Directory)
        * named 'name'
        *);
     error:<DirException
       (* Raised if other errors occured *)
  enter name[]
  ...
  #);
createDir:
  (* Create a directory named 'name' in THIS(Directory) *)
  (# name: ^text;
     newEntry: ^EntryDesc;
     exists:< EntryExistException
       (* Raised if en entry of that name already existed *);
     error:< DirException
       (* Raised if other errors occurred *);
  enter name[]
  ...
  exit newEntry[]
  #);
deleteDir:
  (* Delete a directory named 'name' in THIS(Directory) *)
  (# name: ^text;
     nosuch:< NoSuchException
       (* Raised if there was no disk entry in THIS(Directory)
        * named 'name'
        *);
     error:< DirException
       (* Raised if other errors occured *)
  enter name[]
  ...
  #);
noOfEntries: IntegerValue
  (* exit the number of entries in THIS(directory) *)
  (# error:<DirException;
  ...
  #);
empty: BooleanValue
  (* TRUE iff THIS(directory) is empty. Note that this does not
   * always imply NoOfEntries=0
   *)
  (# error:<DirException;
  ...
  #);
findEntry:
  (* Calls INNER if entry was found in THIS(directory), and
   * otherwise calls notFound
   *)
  (# <<SLOT DirFindLib: attributes>>;
      candidate: ^text;
     (* The name of the entry to search for *)
      foundDesc:< DiskEntry;
```

```
(* Qualification of "found" *)
 found: ^foundDesc;
(* Reference to entry, if found.  Notice that 'found.path'
 * is relative to THIS(directory).  The full path may be
 * obtained by 'foundFullPath' Also 'foundFullPath ->
 * found.path' may be needed before is queried for modtime
 * etc., if THIS(Directory) is not the current working
 * directory.
 *)
 foundFile:< File;
(* Qualification of file generated in
 * select.whenfile.thefile
 *)
 foundDir:< Directory;
(* Qualification of directory generated in
 * select.whendir.thedir
 *)
 foundFullPath: (* Fullpath of "found" *)
   (# p: ^text do ... exit p[] #);

 notfound:< (* Called if the entry was not found *)
   (# do INNER #);
 select:
   (* Used to distinguish between the various entries that
    * may be found
    *)
   (#  error:< found.DiskEntryException;
       whenFile:<
        (* Called when the entry found is a file *)
        (# thefile:
             (* Generate an instance of foundFile
              * corresponding to the entry found. Notice that
              * 'found' and 'f.entry' are two distinct
              * objects; 'f.entry' has a full path, 'found'
              * may or may not have a path relative to
              * THIS(Directory).
              *)
             (# f: ^foundFile
             do ...
             exit f[]
             #);
        do INNER
        #);
       whenDir:<
        (* Called when the entry found is a directory *)
        (# theDir:
             (* Generate an instance of foundDir
              * corresponding to the entry found. Notice that
              * 'found' and 'd.entry' are two distinct
              * objects; 'd.entry' has a full path, 'found'
              * may or may not have a path relative to
              * THIS(Directory).
              *)
             (# d: ^foundDir
             do ...
             exit d[]
```

```
                     #);
                  do INNER
                  #);
                 whenOther:<
                  (* Called when the entry found is neither a file nor
                   * a directory
                   *)
                  (# do INNER #);
                 selectImpl:< (* private *)
                  (# selectedInInner: @boolean
                  do ...;
                     INNER; ...;
                  #)
              do selectImpl;
              #); (* select *)
             error:< DirSearchException
              (* Raised if the search fails *);
          enter candidate[]
          do ...;
          #); (* findEntry *)
        scanEntries:
          (* Calls INNER for each entry in THIS(directory) *)
          (# <<SLOT DirScanLib: attributes>>;
             longest: @integer;
            (* The length of the longest entry-name in THIS(directory)
             *)
             foundDesc:< DiskEntry;
            (* Qualification of "found" *)
             found: ^foundDesc;
            (* Reference to entry, if found.  Notice that 'found.path'
             * is relative to THIS(directory).  The full path may be
             * obtained by 'foundFullPath' Also 'foundFullPath ->
             * found.path' may be needed before is queried for modtime
             * etc., if THIS(Directory) is not the current working
             * directory.
             *)
             foundFile:< File;
            (* Qualification of file generated in
             * select.whenfile.thefile
             *)
             foundDir:< Directory;
            (* Qualification of directory generated in
             * select.whendir.thedir
             *)
             foundFullPath: (* Fullpath of "found"  *)
              (# p: ^text do ... exit p[] #);
             select:
              (* Used to distinguish between the various entries that
               * may be found
               *)
              (#  error:< found.DiskEntryException;
                 whenFile:<
                  (* Called when the entry found is a file *)
                  (# thefile:
                       (* Generate an instance of foundFile
                        * corresponding to the entry found. Notice that
                        * 'found' and 'f.entry' are two distinct
                        * objects; 'f.entry' has a full path, 'found'
                        * may or may not have a path relative to
                        * THIS(Directory).
                        *)
                       (# f: ^foundFile
                       do ...
                       exit f[]
                       #);
                  do INNER
                  #);
                 whenDir:<
                  (* Called when the entry found is a directory *)
                  (# theDir:
                       (* Generate an instance of foundDir
                        * corresponding to the entry found. Notice that
                        * 'found' and 'd.entry' are two distinct
                        * objects; 'd.entry' has a full path, 'found'
                        * may or may not have a path relative to
                        * THIS(Directory).
                        *)
                       (# d: ^foundDir
                       do ...
                       exit d[]
                       #);
                  do INNER
                  #);
                 whenOther:<
                  (* Called when the entry found is neither a file nor
                   * a directory
                   *)
                  (# do INNER #);
                 selectImpl:< (* private *)
                  (# selectedInInner: @boolean
                  do ...;
                     INNER; ...;
                  #)
              do selectImpl;
              #); (* select *)
             error:< DirScanException
              (* Raised if the scan fails *);
            (* idx- *) (* idx- *)
          do ...;
          #); (* scanEntries *)
        private: @...;
     #) (* directory *)
```

10. The unixFile Library

The `unixFile` library defines the UNIX specific properties of the disk entry attributes and contents related attributes of UNIX files. This is done through specializations of the `diskEntry` and `file` patterns, namely the `unixEntry` and `unixFile` patterns.

unixFile attributes

`UnixEntry` defines many new attributes, such as `permission`, `owner`, `accessTime`, etc., and `unixFile` defines only two additional attributes: `openExeWrite` and `openExeAppend`.

10.1 Using the unixFile Fragment

A program using the `unixFile` fragment will have the following structure:

```
INCLUDE '~beta/unixlib/v1.6/unixFile'
--- program: descriptor ---
(# ...
   uf: @unixFile;
do ...
   ('a','r') -> uf.entry.permission.add;
   ...
#)
```

Please note, that `unixFile` is not placed in the `basiclib` directory, but in a directory `unixlib`, holding specific files for the UNIX version of the Mjølner System.

10.1.1 Unix Entry Example

An example of using `unixEntry`: First the user is asked if symbolic links should be followed or not. If the `unixEntry` specified on the command line exists, it is then various other UNIX specific attributes are examined. It is examined whether the entry is a symbolic link or not, only if links should not be followed.

Finally it is attempted to change the owner of it to 675:

unixEntry.bet

```
ORIGIN '~beta/unixlib/v1.6/unixfile';
INCLUDE '~beta/sysutils/v1.6/time';
---- PROGRAM: descriptor ----

(* An example of using UnixEntry: First the user is asked if symbolic links should
 * be followed or not. If the UnixEntry specified on the command
 * line exists, it is then various other unix specific attributes are examined.
 * It is examined whether the entry is a symbolic link or not, only if links
should
```

```
 * not be followed.
 * Finally it is attempted to change the owner of it to 675.
 *)

(# e: @unixentry; follow: @boolean;
do (if noOfArguments <> 2 //true then
       'Usage: ' -> puttext; 1->arguments->puttext; ' path' -> putline;
       stop;
   if);
   2 -> arguments -> e.path;

   'Examine symbolic links themselves? (y/n) ' -> puttext;
   (Keyboard.get='n') -> e.followlinks;

   e.path -> puttext;
   (if e.exists
    // true then
       ' exists. It' -> putline;
       (if e.followlinks // false then
           (if e.isSymbolicLink // true then
               '  is a symbolic link' -> putline;;
           if);
       if);
       '  was last modified ' -> puttext;
       e.modtime -> formattime -> putline;
       '  has inode: '->puttext; e.inode -> putint; newline;
       '  has owner id: '->puttext; e.owner -> putint; newline;
       '  the owner '-> puttext;
       (if ('u','w') -> e.permission.has
        // true then 'has write permission to the disk entry' -> putline;
        // false then 'does not have write permission to the disk entry'
           -> putline;
       if);
       'Now lets try to change the owner id to 675.' -> puttext;
       675 -> e.owner;
       ' That succeeded!' -> putline;
    // false then
       ' does not exist.' -> putline;
   if);
#)
```

10.1.2 Split executable file

A simple example of using `unixFile` : The file '`in.bet`' is read word by word, and each word is printed on the output file specified on the command line. This output file has been opened with `openExeWrite` , meaning that the resulting file will be executable. After this, the output file is opened again, this time with `openExeAppend` , meaning that the output file will be executable after the line of text has been appended to it.:

splitexe.bet

```
ORIGIN '~beta/unixlib/v1.6/unixfile';
---- PROGRAM: descriptor ----

(* A simple example of using UnixFile: The file 'input' is read word by word,
 * and each word is printed on the output file specified on the command line.
 * This output file has been opened with OpenExeWrite, mening that the resulting
 * file will be executable.
 * After this, the output file is opened again, this time with OpenExeAppend,
 * meaning that the output file will be executable after the line of text
 * has been appended to it.
 *)
```

```
(# inFile, outFile: @ unixfile;
do (if noOfArguments <> 2 //true then
       'Usage: ' -> puttext; 1->arguments->puttext; ' out-file' -> putline;
       stop;
   if);
   2 -> arguments -> outFile.name;
   outFile.OpenExeWrite;

   'input' -> inFile.name;
   inFile.OpenRead;

   readFile:
     (#
     do (if inFile.eos
         // false then
            inFile.getAtom -> outFile.putText;
            outFile.newLine;
            restart readFile
         // true then
            leave readFile
        if)
     #);
   inFile.close;
   outFile.close;

   (* Now we'll append a little too *)
   outfile.OpenExeAppend;
   'This is one line appended' -> outfile.putline;
   outfile.newline;
   outfile.close;

   'The tokens from the file ''input'' has been put into the file' -> putline;
   ''''->put; outFile.name -> puttext;
   ''', and an extra line of text has been appended.' -> putline;
#)
```

10.2 Interface Description for the unixFile Library

```
ORIGIN '~beta/basiclib/v1.6/file';
BODY 'private/unixfile_body';

---- LIB: attributes ----

UnixEntry: DiskEntry
  (* Pattern describing unix specific attributes of disk-entries like
   * files and directories in a hierarchic unix file system
   *)
  (#
     <<SLOT UnixEntryLib: attributes>>;

     UnixEntryExisted: Exception
       (* Raised on attempt to make an existing unix entry into a
        * symbolic link using linkTo or linkFrom. Message: "Attempt to
        * make an existing UnixEntry into a symbolic link"
        *)
       (# ... #);

     followlinks:
       (* If the disk entry corresponding to THIS(UnixEntry).path
        * exists and is a link to some other entry, then if
        * followlinks is true the disk entry pointed to is manipulated
        * by the operations below, otherwise the link itself is
        * manipulated.  Defaults to TRUE.
        *)
       (# f: @boolean;
       enter (# enter f ... #)
       exit (# ... exit f #)
       #);

     permission: @permissiondesc
       (* An object to manipulate the protection and mode of
        * THIS(UnixEntry).  This gives a more fine-grained access to
        * the unix permissions than the inherited attributes
        * "readable" and "writeable".  All operations on "permission"
        * are used like this :
        *
        * (who,mode) -> permission.operation ... ;
        *
        * "who" denotes a category of users : 'a' -- all, 'u' -- user
        * (owner), 'g' -- group, 'o' -- other.
        *
        * "mode" denotes the mode of THIS(UnixEntry) : 'r' -- read,
        * 'w' -- write, 'x' -- execute.
   *);
permissiondesc:<
  (# <<SLOT UnixFilePermLib: attributes>>;

     (* PERMISSION EXCEPTIONS *)
     WrongUserError:< Exception
       (* Raised if something other than 'a', 'u', 'g', or 'o'
        * is specified for "who". Message: "Wrong user
        * specification for disk entry permission."
        *)
       (# ... #);
     WrongModeError:< Exception
       (* Raised if something other than 'r', 'w', or 'x' is
        * specified for "mode". Message: "Wrong mode
        * specification for disk entry permission."
        *)
       (# ... #);
     ChangePermError:< DiskEntryException
       (* Raised if a change of permissions has failed for
        * THIS(UnixEntry).  Message: "Failed to change permission
        * for disk entry."
        *)
       (# ... #);
     OtherError:< DiskEntryException
       (* Raised if other errors occurred *);

     has:
       (* True if "who" has the "mode" access to
        * THIS(UnixEntry).
        *)
       (# mode,who: @char;
          result: @boolean;
       enter (who,mode)
       ...
       exit result
       #);
     add:
       (* Give "who" the "mode" access to THIS(UnixEntry) *)
       (# mode,who: @char;
       enter (who,mode)
       ...
       #);
     remove:
       (* Remove the "mode" from the way "who" may access
        * THIS(UnixEntry).
        *)
       (# mode,who: @char;
       enter (who,mode)
       ...
       #);
  #); (* permission *)
device: IntegerValue
  (* Integer denoting the device on which the Inode of
   * THIS(UnixEntry) resides
```

```
         *)
        (# error:<DiskEntryException;
        ...
        #);
      inode: IntegerValue
        (* Unambigous integer denoting the identity of this unix entry
         *)
        (# error:<DiskEntryException;
        ...
        #);
      isCharSpecial: BooleanValue
        (# error:<DiskEntryException;
        ...
        #);
      isBlockSpecial: BooleanValue
        (# error:<DiskEntryException;
        ...
        #);
      isSymbolicLink: BooleanValue
        (* True if THIS(UnixEntry) is a symbolic link.  Always false
         * if followlinks is set to true (default)
         *)
        (# error:<DiskEntryException;
        ...
        #);
      isSocket: BooleanValue
        (* True if THIS(UnixEntry) is a socket *)
        (# error:<DiskEntryException;
        ...
        #);
      hardLinks: IntegerValue
        (* Number of hard links to THIS(UnixEntry) *)
        (# error:<DiskEntryException;
        ...
        #);
      deviceType: IntegerValue
        (* If THIS(UnixEntry) is a device, the integer exited denotes
         * the type of the device
         *)
        (# error:<DiskEntryException;
        ...
        #);
      owner:
        (* Set/get the userid (an integer) denoting the identity of
         * the owner of THIS(UnixEntry)
         *)
        (# uid: @integer;
           error:<DiskEntryException;
           set: (# enter uid do ... #);
           get: (# ... exit uid #);
        enter set
        exit get
        #);
      group:
```

```
  (* Set/get the group id (an integer) denoting the identity of
   * the group to which the owner of THIS(UnixEntry) belongs
   *)
  (# gid: @integer;
     error:<DiskEntryException;
     set: (# enter gid do ... #);
     get: (# ... exit gid #);
  enter set
  exit get
  #);
accessTime: IntegerValue
  (* The (system) time of the last read or write manipulation of
   * THIS(UnixEntry)
   *)
  (# error:<DiskEntryException;
  ...
  #);
statusTime: IntegerValue
  (* System time of the last status change of
   * THIS(UnixEntry). In our case a status change occurs when the
   * following operations are activated on THIS(UnixEntry):
   * touch, permission.add, permission.remove
   *)
  (# error:<DiskEntryException;
  ...
  #);
optimalBlockSize: IntegerValue
  (# error:<DiskEntryException;
  ...
  #);
blocks: IntegerValue
  (* The actual number of blocks allocated for THIS(UnixEntry) *)
  (# error:<DiskEntryException;
  ...
  #);
linkTo:
  (* Make THIS(UnixEntry) be a symbolic link to dst. The
   * 'exists' exception is raised if THIS(UnixEntry) allready
   * exists.
   *)
  (# dst: ^text;
     error:< DiskEntryException;
     exists:< UnixEntryExisted;
     link: ^UnixEntry;
  enter dst[]
  do ...;
  #);
linkFrom:
  (* Make src be a symbolic link to THIS(UnixEntry). The
   * 'exists' exception is raised if src allready exists. If no
   * errors occur, link is a UnixEntry for the link, with
   * followlinks set to false.
   *)
  (# src: ^text;
          error:<DiskEntryException;
          exists:< UnixEntryExisted;
          link: ^UnixEntry;
       enter src[]
       do ...;
       exit link
       #);

  #); (* UnixEntry *)

UnixFile: File
  (* BETA interface to disk files in UNIX *)
  (#
     <<SLOT UnixFileLib: attributes>>;

     EntryDesc::< UnixEntry;

     OpenExeWrite:
       (* Like OpenWrite, except that the file written will be
        * executable
        *)
       (# ... #);
     OpenExeAppend:
       (* Like OpenAppend, except that the file written will be
        * executable
        *)
       (# ... #);
  #)
```

11. The unixDirectory Library

The unixDirectory library defines the UNIX specific properties of the disk entry attributes and directory related attributes of UNIX directories. This is done through specializations of the `diskEntry` and `file` patterns, namely the `unixEntry` and `unixDirectory` patterns.

`unixDirectory` attributes

The `unixEntry` pattern is the same as the one discussed in the `unixFile` library, and the `unixDirectory` pattern only makes the necessary specializations to make the inherited `directory` properties function on the UNIX hierarchical file system.

11.1 Using the unixDirectory Fragment

A program using the `unixDirectory` fragment will have the following structure:

```
INCLUDE '~beta/unixlib/v1.6/unixDirectory'
--- program: descriptor ---
(# ...
   ud: @unixDirectory;
do ...
   'unixDirectory.bet' -> ud.findEntry(# ... do ... #);
#)
```

Please note, that `unixDirectory` is not placed in the `basiclib` directory, but in a directory `unixlib`, holding specific files for the UNIX version of the Mjølner System.

11.1.1 Listing Unix Directory Example

Program showing a simple use of UNIX directory: The directory with the path given as argument is scanned, and the names of all the entries are printed with an indication of what type of entry it is. This is done using the `select` pattern of `scanEntries`. This is a more efficient strategy than using `found.entry.isFile` etc., possibly correcting for the case that `'d'` is not current working directory.

If the path given is not a directory, an exception will be raised:

listUnix.bet

```
ORIGIN '~beta/unixlib/v1.6/unixdirectory';
-- PROGRAM: descriptor --

(* Program showing a simple use of UNIX directory: The directory with the path
 * given as argument is scanned, and the names of all the entries are printed
 * with an indication of what type of entry it is.
 * This is done using the 'select' pattern of 'scanentries'. This is a more
 * efficient startegy than using 'found.entry.isFile' etc., possibly
 * correcting for the case that 'd' is not current working directory.
 * If the path given is not a directory, an exception will be raised.
 *)

(# arg: ^text;
   d: @UnixDirectory;
   nl: @boolean;
   full: @boolean;
   usage:
     (# do 'Usage: ' -> puttext;
        1->arguments->puttext;
        ' [-f] path' -> putline;
        stop;
     #);
do (* Parse command line *)
   (if noOfArguments
    // 1 then '.' -> d.name
    // 2 then
       2 -> arguments -> arg[];
       (if '-f' -> arg.equal
        // true then
           true -> full;
           '.' -> d.name;
        // false then
           arg[] -> d.name;
       if)
    // 3 then
       2 -> arguments -> arg[];
       (if '-f' -> arg.equal
        // true then
           true -> full;
        // false then usage;
       if);
       3 -> arguments -> d.name;
    else
       usage;
   if);

   (* Scan directory *)
   newline;
   'The content of '''-> puttext;
   d.name -> puttext;
   ''' is: ' -> putline;
   d.scanEntries
   (#
   do false -> found.followlinks;
      select
      (# whenFile::<
           (# do 'File:          ' -> puttext; #);
         whenDir::<
           (# do 'Directory:     ' -> puttext; #);
         whenSocket::<
           (# do 'Socket:        ' -> puttext; #);
         whenSymboliclink::<
           (# do 'Link:          ' -> puttext; #);
         whenCharSpecial::<
           (# do 'Char special:  ' -> puttext; #);
         whenBlockSpecial::<
           (# do 'Block special: ' -> puttext; #);
         whenOther::<
           (# do '(Unknown):     ' -> puttext; #);
      #);
      (if full
       // true then foundFullPath -> putline;
       // false then found.path -> putline;
      if);
   #);
   newline;
```

```
#)
```

11.2 Interface Description for the unixDirectory Library

```
ORIGIN '~beta/basiclib/v1.6/directory';
BODY 'private/unixdirectory_body';

INCLUDE 'unixfile';
---- LIB: attributes ----

unixdirectory: directory
  (* BETA interface to disk files in UNIX *)
  (#
     <<SLOT UnixDirectoryLib: attributes>>;

     EntryDesc::< UnixEntry;

     findEntry: dirFindEntry
       (#
          foundDesc::< UnixEntry;
          foundFile::< UnixFile;
          foundDir::< UnixDirectory;
          select: dirselect
            (#  whenSocket:<       (# do INNER #);
                whenSymboliclink:< (# do INNER #);
                whenCharSpecial:<  (# do INNER #);
                whenBlockSpecial:< (# do INNER #);

                selectImpl::< (* private *)
                 (# ... #)
            #);
          do INNER
       #);
     scanEntries: dirScanEntries
       (#
          foundDesc::< UnixEntry;
          foundFile::< UnixFile;
          foundDir::< UnixDirectory;
          select: dirselect
            (#  whenSocket:<       (# do INNER #);
                whenSymboliclink:< (# do INNER #);
                whenCharSpecial:<  (# do INNER #);
                whenBlockSpecial:< (# do INNER #);

                selectImpl::< (* private *)
                 (# ... #)
            #);
       do INNER
       #);
     (* idx- *) (* idx- *)
  #)

(* Instead of scanEntries and findEntry being virtual (i.e. virtual
 * prefixes):
 *)
---- DirectoryLib: attributes ----
dirFindEntry: findEntry(# do INNER #);
dirScanEntries: scanEntries(# do INNER #);

---- DirFindLib: attributes ----
dirSelect: select(##);

---- DirScanLib: attributes ----
dirSelect: select(##);
```

12. The systemEnv Libraries

The systemEnv libraries define the experimental concurrency system for BETA. The systemEnv libraries contain five closely related libraries: `basicsystemenv.bet` , `systemenv.bet` , and `timehandler.bet` .

basicsystemenv

The `basicsyst emenv.bet` fragment contains the core of the concurrency system, and is the prime fragment for information on the concurrency facilities.

systemenv

The `systemenv.bet` fragment is to be used if the concurrency is to be used in a program that is not using any graphical user interface system, such as the X Window System. `Systemenv.bet` does not define any new attributes at all.

The `timehandler.bet` fragment contains facilities for setting timers in the form of objects to be executed when a given time period have elapsed.

The rest of the chapter will only describe the `basicsystemenv .bet` fragments. For details on the use of the other fragments, please see the documentation in the interface descriptions.

12.1 The basicSystemEnv Library

The `systemEnv` fragment contains abstract *superpattern:*s for describing the BETA concepts of concurrent systems. The basic ideas are:

Basic concepts

1. Components (coroutines) can be executed concurrently.
2. A primitive `semaphore` pattern is available for synchronization. The operations on a `semaphore` must be executed as an indivisible unit.
3. An abstract pattern `Monitor` similar to the monitor proposed by Hoare and Brinch-Hansen.
4. An abstract pattern `System` is defined. `System` defines communication between systems by means of synchronized rendezvous. A concurrency imperative `conc` and an alternation imperative `alt` are defined for `system`.

Currently an exploratory style is used to experiment with different variants of the abstract patterns. The current version is thus far from any final form of definition and may contain errors. The separation into interface and implementation has not been completely carried out.

The abstractions defined here are based on the ones described in chapter 12 of the BETA book.

12.1.1 Changes from original design

The implementation is identical to the design in the BETA book, except for the following changes:

1. The syntax of `fork` is
   ```
   S[]->fork
   ```
   and *not* `S.fork`.
2. The syntax of `conc` is
   ```
   conc(# do S1[]->start; S2[]->start; S3[]->start #)
   ```
   and *not* `conc(# do S1.start; S2.start; S3.start #)`.
3. The syntax of `alt` is
   ```
   alt(#do S1[]->start; S2[]->start; S3[]->start #)
   ```
   and *not* `alt(# do S1.start; S2.start; S3.start #)`.

12.1.2 New facilities

This implementation of `systemenv` included a few new facilities, not described in the BETA book:

4. `semaphore` have an additional attribute: `tryP`, which is a non-blocking call of `P`.
5. In addition to `s[]->fork`, `s[]->kill` is now possible, and in addition to `pause`, `100->sleep` is possible.
6. `system` have a new virtual attribute, `onKilled`, that is invoked before the system terminates
7. `systemenv` has a new virtual attribute, `deadlocked`, that is invoked if all processes are deadlocked.
8. Finally, `systemenv` defines three new attributes to cope with event driven user interfaces: `windowEnvType`, `theWindowEnv`, and `setWindowEnv`. See further details on cooperation with user interface environments below.

12.1.3 The Concurrency is Simulated

In order to implement real concurrency, an interrupt mechanism must be implemented. This is currently *not* done. A component/system will thus keep the control until it makes an explicit or implicit `SUSPEND`. An implicit `SUSPEND` is made when a component must wait for a `semaphore`, executes the `pause` pattern, executes the `sleep` pattern, or performs a blocking communication using the `shellEnv` distribution libraries (not described in this manual).

12.1.4 Concurrency and User Interface Environments

User interface environments (such as X Window System) are usually event-driven in the sense that actions in the program are executed as a response to user input events. To handle this, a number of separate implementations of `SystemEnv` exists for the different user interface libraries, such as xtEnv, awEnv, motifEnv, uiEnv, and MacEnv:

- Use `systemenv.bet` as origin for programs not using event-driven user-interface libraries.
- Use `~beta/Xt/current/xsystemenv .bet` as origin for programs using xtEnv, awEnv or motifEnv.

- Use `~beta/uienv/current/uienvsystemenv .bet` as origin for programs using UIenv.

See `xsystemenv` and `uienvsystemenv` for a description of using `systemenv` in conjunction with X and UIenv programs, respectively.

See `~beta/macenv/current/macsystemenv` for a description of using `systemenv` in conjunction with macenv.

For examples of using `SystemEnv`, see the `demo` directory.

Please note, that programs should only use *one* of the `systemenv`, `xsystemenv`, and `uienvsystemenv` fragments. It is a fairly common mistake in `systemEnv` programs to find more than one of these fragments.

12.2 Using the SystemEnv Fragment

A program using the systemEnv fragment will have the following structure:

```
INCLUDE '~beta/basiclib/v1.6/systemenv'
--- program: descriptor ---
systemenv
  (# process: @ |system(# ... #);
  do ...
     process[] -> fork;
     ...
  #)
```

12.2.1 The Monitor Example

The following is an example of a producer/consumer system with a shared buffer (implemented as a 20 element character buffer, protected as a `monitor`. The producer and consumer are concurrent objects:

buffer.bet

```
ORIGIN '~beta/basiclib/v1.6/systemenv';

---program: descriptor---
SystemEnv
(# buffer: @Monitor
     (# R: [20] @char; in,out: @integer;
        full,empty: @Condition;

        put: Entry
          (# ch: @char
          enter ch
          do (if in = out then full.wait if);
             ch->R[in]; (in mod R.range)+1 ->in;
             empty.signal;
          #);
        get: Entry
          (# ch: @char
          do (if in = (out mod R.range)+1 then empty.wait if);
             R[(out mod R.range)+1->out]->ch;
             full.signal;
          exit ch
          #);
        init::< (# do 1->in; R.range->out #)
     #);
```

```
      prod: @| System(# do cycle(# do keyboard.get->buffer.put #) #);
      cons: @| System(# do cycle(# do buffer.get->screen.put #) #);

   do buffer.init;
      conc(# do prod[]->start; cons[]->start #)
   #)
```

12.2.2 The Monitor with Wait Example

This example is similar to the previous, except that `wait` is used instead of `condition` to control the medium-term scheduling of access to the buffer.

Ocbuf.bet

```
   ORIGIN '~beta/basiclib/v1.6/systemenv';

   ---program: descriptor--
   systemenv
   (# buffer: @Monitor
        (# R: [4] @char; in,out: @integer;
           full: (# exit in=out #);
           empty: (#exit (in = (out mod R.range)+1) #);
           Put: Entry
             (# ch: @char
             enter ch
             do wait(# do (not full)->cond #);
                ch->R[in]; (in mod R.range)+1 ->in;
             #);
           get: Entry
             (# ch: @char
             do wait(# do (not empty)->cond #);
                R[(out mod R.range)+1->out]->ch;
             exit ch
             #);
           init::< (# do 1->in; R.range->out #)
        #);

      prod: @| System(# do cycle(# do keyboard.get->buffer.put #) #);
      cons: @| System(# do cycle(# do buffer.get->screen.put #) #);

   do buffer.init;
      conc(# do prod[]->start; cons[]->start #)
   #)
```

12.2.3 The Ports Example

The following is an example of three communicating objects: `S`, `R1` and `R2`. `R1` and `R2` are similar (instances of the same `Rtype` pattern).

`Rtype` defines two ports: `p1` and `p2`, with a `get entry` in `p1` and a `put entry` in `p2`. The `get` entry prints the value of the `x` attribute on standard output, and `put` prints the `y` attribute. `Rtype` objects repeatedly `accepts` p1 communications followed by `p2` communications (i.e. since in this example, only `get` is define in `p1`, and only `put` in `p2`, this implies `get` followed by `put`). Note that the use of ports, allows specializations of `Rtype` to define new entries in `p1` and/or `p2`, such that these communications follow the same communication structure.

The S object defines two internal objects, C1 and C2, which are executed alternating. C1 is responsible for the communication with R1 and C2 is responsible with the communication with R2. It is in this way ensured that the communication pattern between S and R1 follows the get followed by put pattern (the same for S and R2), but these two communication patterns may be non-deterministically interleaved.

altex1.bet

```
ORIGIN '~beta/basiclib/v1.6/systemenv';

---program: descriptor--
SystemEnv
(# S: @| System
     (# C1: @| System
          (#
          do cycle(# do '1'->put; R1.get; '2'->put; R1.put #);
          #);
        C2: @| System
          (#
          do cycle(# do 'a'->put; R2.get; 'b'->put; R2.put #)
          #)
     do alt(# do C1[]->start; C2[]->start #)
     #);

   Rtype: System
     (# get: p1.entry(# do x->screen.put; #); p1: @port;
        put: p2.entry(# do y->screen.put; #); p2: @port;
   x,y: @char
     do cycle(# do p1.accept; p2.accept #)
     #);
   R1: @| Rtype;
   R2: @| Rtype;

do '('->R1.x; ')'->R1.y; '['->R2.x; ']'->R2.y;
   conc(# do S[]->start; R1[]->start; R2[]->start #)
#)
```

12.2.4 The ObjectPort Example

This example illustrates the use of the ObjectPort facility. The S object defines f1, f2, and f3 communication entries, where f1 is controlled by an objectPort . This enables S to control exactly which object which is allowed to communicate f3's in an accept. Note, that initially, R.R1 is allowed, and later R.R3 is allowed:

sys1.bet

```
ORIGIN '~beta/basiclib/v1.6/systemenv';

---program: descriptor--
SystemEnv
(# S: @| System
     (# P1: @ObjectPort;
        f1: P1.entry(# do 'f1 called' ->putline #);
   P2: @Port;
        f2: P2.entry(# do 'f2 called' ->putline #);
   P3: @Port;
        f3: P3.entry(# do 'f3 called' ->putline #);
     do R.R1[]->P1.accept;
        P2.accept;
        P3.accept;
        R.R3[]->P1.accept
     #);

   R: @| System
     (# R1: @| System(# do S.f1; S.f3 #);
        R2: @| System(# do S.f2; #);
        R3: @| System(# do S.f1 #);
     do 'Start R'->putLine;
        conc(# do R1[]->start;  R2[]->start; R3[]->start #);
        'End R'->putLine
     #);

do 'Start'->putLine;
   conc(# do R[]->start; S[]->start  #);
   'End'->putLine
#)
```

12.3 Interface Description for the systemenv Library

12.3.1 Interface Description for the basicsystemEnv Library

```
ORIGIN 'betaenv';
BODY 'private/basicsystemenvbody'
---LIB:attributes---
(*
 * COPYRIGHT
 *       Copyright Mjolner Informatics, 1992-96
 *       All rights reserved.
 *
 * This fragment contains abstract   superpattern:s for describing the
 * BETA concepts of concurrent systems.
 *
 * The basic ideas are
 *
 *      A. Components (coroutines) can be executed concurrently
 *
 *      B. A primitive semaphore pattern is available for
 *         syncronization.
 *
 *      C. An abstract pattern 'Monitor' similar to the monitor
 *         proposed by Hoare and Brinch-Hansen
 *
 *      D. An abstract pattern 'System' is defined. System defines
 *         communication between systems by means of synchronized
 *         rendezvous.  A concurreny imperative 'conc' is defined for
 *         systems.
 *
 * The abstractions defined here are identical to the ones described
 * in chapter 12 of the BETA book except for the following points:
 *
 * 1. The syntax of 'fork' is
 *      S[]->fork
 *    and NOT S.fork
 *
 * 2. The syntax of 'conc' is
 *      conc(# do S1[]->start; S2[]->start; S3[]->start #)
 *    and NOT conc(# do S1.start; S2.start; S3.start #)
 *
 * 4. THE CONCURRENCY IS SIMULATED In order to implement real
 *    concurreny, an interrupt mechanism must be implemented. This is
 *    currently NOT done. A component/system will thus keep the
 *    control until it makes an explicit or implicit SUSPEND.  An
 *    implicit SUSPEND is made when a component must wait for a
 *    semaphore, executes the pause pattern, executes the sleep
 *    pattern, or performs a blocking communication using the shellEnv
 *    distribution abstractions.  As the concurrency is simulated,
 *    there is no difference between the implementation of the alt and
 *    conc imperatives.
 *
 * 5. A program using concurrency must have the form:
 *      systemenv(# ... do ... #)
 *
 * 6. Concurrency and X-Windows/macenv/guienv
 *    User interface environments are usually event-driven in the
 *    sense that actions in the program are executed as a response to
 *    user input events.  To handle this, a number of separate
 *    implementations of SystemEnv exists for different user interface
 *    libraries:
 *
 *    Use systemenv.bet as origin for programs not using event-driven
 *    user-interface libraries.
 *
 *    Use ~beta/Xt/current/xsystemenv.bet as origin for programs using
 *    XtEnv, AwEnv or MotifEnv.
 *
 *    Use ~beta/guienv/current/guienvsystemenv.bet as origin for
 *    programs using GUIenv (Lidskvjalv).
 *
 *    See xsystemenv and guienvsystemenv for a description of using
 *    systemenv in conjunction with X and GUIenv programs,
 *    respectively.
 *
 *    See ~beta/macenv/current/macsystemenv for a description of using
 *    systemenv and macenv.
 *
 * For examples of using SystemEnv see the demo directory.
 *)
getSystemEnv:
  (* Returns the unique systemEnv instance running *)
  (# theSystemEnv: ^systemEnv;
  do SystemEnv## -> objectPool.strucGet
     (# init::<
          (#
          do (failure,
             'Program:descriptor must be a subpattern of systemEnv')
               -> stop
     #)#) -> theSystemEnv[];
  exit theSystemEnv[]
  #);
SystemEnv: SysHead
  (# <<SLOT systemlib:attributes >>;
     semaphore:
       (* P and V are the usual semaphore operations.
        *
        * tryP returns true if the P operation succeded. Returns false
        * if a P would block the caller. In that case the P operation
        * is not performed.
```

```
   *
   * Count returns the number of components waiting for the
   *        semaphore.
   *)
  (# P: @...;
     V: @...;
     tryP: @BooleanValue
       (# ... #);
     Count: @
       (# value: @Integer;
       ...
       exit value
       #);
     semRep: @...
  #);
fork: @
  (* S is put into the queue of scheduled systems. The calling
   * system keeps control, i.e. is not preempted.
   *)
  (# first:  @...;
     second: @...;
     S: ^|SysHead
  enter S[]
  do first; second; none -> s[];
  #);
kill: @
  (* Kills S. If S is the active system, this is equivalent to a
   * direct suspend.
   *)
  (# S: ^|SysHead; doKill: @...
  enter S[]
  do doKill
  #);
pause: @
  (* Moves the calling system to the end of the queue of
   * scheduled systems.
   *)
  ...;
sleep: @
  (* Makes the calling system sleep at least time seconds.  If
   * time is 0 or negative, sleep has no effect.
   *)
  (# time: @Real
  enter time
  ...
  #);
sleepUntil:
  (* Makes the calling system sleep until at least time.  If
   * time is less than the current time, sleepUntil has no effect.
   *)
  (# time: @Real
  enter time
  ...
  #);
```

```
timeStamp:
  (# value: @Real;
  ...
  exit value
  #);
Monitor:
  (# (* idx+ *)
      Condition:
        (# q: @Semaphore;
           Wait: ...;
           Signal: ...;
        #);
      Wait:
        (# cond: @boolean
        do INNER;
           (if not cond then
               return; (* exit monitor *)
               pause;
               mutex.P; (* reentry of monitor *)
               restart Wait
           if)
        #);
      Entry: (# do mutex.P; INNER; return #);
      init:< (# do INNER; mutex.V; #);
     (* private:
      *
      * mutex controls entry to the Monitor.  urgent delays a
      * signalling process.
      *
      * return is executed by processes leaving the monitor.
      * Reactivates possible processes waiting for entry: delayed
      * signalling processes (urgent) have first priority
      *)
     mutex: @semaphore;
     urgent: @semaphore;
     return: @...;
  #);
System: SysHead
  (#  Port:
        (# mx,m: @Semaphore;
           entry: (# do m.P; INNER; mx.V #);
           accept: (# do m.V; mx.P #)
        #);
      RestrictedPort:
        (# mx, am: @Semaphore;
           delayed: @...;
           accept:<(# ... #);
           acceptable:<
             (# OK: @Boolean; s: ^|sysHead
             enter s[]
             do INNER
             exit OK
             #);
           restrictedEntry:
             (# ... #);
        #);
       ObjectPort: RestrictedPort
        (# accept::< (# enter sender[] do none->sender[] #);
           acceptable::< (# ... #);
           entry: RestrictedEntry (# do INNER #);
           sender: ^|sysHead
        #);
       QualifiedPort: RestrictedPort
        (# accept::< (# enter sender## do none->sender## #);
           acceptable::< (# ... #);
           entry: RestrictedEntry(# do INNER #);
           sender: ##sysHead
        #);
       conc:
        (# start:
             (# s: ^|system
             enter s[]
             ...
             #);
           concPriv: @...
        do INNER; ...;
        #);
       alt: conc (# do INNER #);
       onKilled:<
        (* Called before this system terminates. *)
        (#
        do (if caller[]<>NONE then (* not the outermost system *)
               caller.dec; NONE -> caller[]
           if);
           INNER;
        #);
       caller: ^protectedInt;
   do INNER;
   #);
  deadLocked:< Exception
   (* This exception is called when all coroutines are blocked
    * and none are waiting for I/O.
    *)
   (#
   do INNER;
      (if not continue then
          ' BasicSystemEnv: All coroutines blocked on semaphores.'
            -> msg.append;
      if);
   #);
  conc:
   (#  start:
        (# s: ^|system
        enter s[]
        ...
        #);
       concPriv: @...
   do INNER; ...;
```

```
          #);
        alt:
          (* Same as conc as a consequence of non-preemtive scheduling.
           *)
          conc (# #);

        (* ATTRIBUTES FOR EVENT-DRIVEN WINDOWING ENVIRONMENTS
         *
         * These attributes are only used when combining SystemEnv with
         * an event-driven windowing environment. This demands an
         * alternative implementation than the standard SystemEnv
         * implementation. See the file: xsystemenv.bet
         *)
        windowEnvType:< Object;
        theWindowEnv: ^windowEnvType;
        setWindowEnv:< Object;

        (* PRIVATE
         *
         * Everything below is in principle private implementation stuff.
         *)
        private: @ ...;
        BasicScheduler: ...;
        theActive: ^|sysHead;
        ProtectedInt: IntegerObject
          (* Used in implementation of conc. *)
          (# mutex: @semaphore;
             atZero: @semaphore;
             dec:
               (#
               do mutex.P;
                  (if (value-1->value)=0 then atZero.V if);
                  mutex.V;
               #);
             waitForZero: (# do atZero.P #);
             init: (# enter value do mutex.V #);
          #);
        initBeforeScheduler:<
          (* Called before the scheduler is activated and before
           * setWindowEnv and the systemenv INNER is called.
           *)
          Object;
     do ...;
        INNER
     #);

   cyclicElm:
     (# s: ^|SysHead;
        next, prev: ^cyclicElm;
        due: @Real
          (* due is used by sleepingQueue. If zero, this element is
           * currently not in a sleepingQueue.
           *)
     #);
```

```
cyclicQueue:
  (# onDelete:< Object;
     onDel: @onDelete;
     onInsert:< Object;
     onIns: @onInsert;
     first, freeList: ^cyclicElm;
     insert: @
       (# s: ^|sysHead; new: ^cyclicElm;
       enter s[]
       ...
       exit new[]
       #);
     append: @
       (# elm: ^cyclicElm;
       enter elm[]
       ...
       #);
     prepend: @
       (# elm: ^cyclicElm;
       enter elm[]
       ...
       #);
     insertBefore: @
       (# new, old: ^cyclicElm;
       enter (new[],old[])
       ...
       #);
     getFirst: @
       (# elm: ^cyclicElm;
       ...
       exit elm[]
       #);
     delete: @
       (# elm: ^cyclicElm;
       enter elm[]
       ...
       exit elm[]
       #);
     remove: @
       (* elm should not be reused after remove. Use delete instead.
        *)
       (# elm: ^cyclicElm; s: ^|sysHead;
       enter elm[]
       ...
       exit s[]
       #);
     scan:
       (# current: ^cyclicElm;
       ...
       #);
     size: @Integer;
  #);
SysHead:
  (# shstatus: @Integer;
     lc: ^Object;     (* Last errorCatcher for distribution errors. *)
     ce: ^cyclicElm;  (* ce,q <> none => this(sysHead) is ce in q.  *)
     q: ^cyclicQueue;
  do INNER
  #);
(* SysHead.shstatus values: *)
SE_RUNNING:  (# exit 1 #);  (* Current system.       *)
SE_WAITING:  (# exit 2 #);  (* Blocked on semaphore. *)
SE_READY:    (# exit 3 #);  (* Ready to run.         *)
SE_SLEEPING: (# exit 4 #);  (* Sleeping.             *)
SE_KILLED:   (# exit 5 #);
```

12.3.2 Interface Description for the systemEnv Library

```
ORIGIN 'basicsystemenv';
BODY 'private/systemenvbody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1984-96
 *       All rights reserved.
 *
 * Use this fragment as the ORIGIN for concurrent BETA
 * programs NOT using X libraries or other UI libraries
 * with a central event-loop.
 *
 * Programs should look something like:
 *
 * ORIGIN 'systemenv';
 * --- program:descriptor ---
 * systemEnv:
 *   (# ...
 *   do ...
 *   #)
 *
 * For details about the concurrency abstractions,
 * see the file basicsystemenv.bet
 *)
```

12.3.3 Interface Description for the timehandler Library

```
ORIGIN 'basicsystemenv';
BODY 'private/timehandlerbody';
(*
 * COPYRIGHT
 *       Copyright (C) Mjolner Informatics, 1984-96
 *       All rights reserved.
 *
 *)
--- systemLib:attributes ---
timeHandler:
  (* timeHandler handles the setting and unsetting of timers.
   *
   * register takes a time to wait and an object reference. It returns
 * a unique id identifying the registration. If the registration is
 * not unregistered before the timer goes off, the object will be
 * executed.  Due to non-preemptive multitasking, it can only be
 * guaranteed that at least time seconds will elapse before obj is
 * executed.
 *
 * unregister takes a registration id and unregisters the
 * registration.  If this happens before the corresponding timer
 * goes off, the registered object will not be executed.
 *
 * init should be called before using the timeHandler.
 *)
(# register:
     (# obj: ^Object; time: @Integer;
        id: @Integer;
     enter (obj[],time)
     ...
     exit id
     #);

   unregister:
     (# id: @Integer;
     enter id
     ...
     #);
   init:
     (#
     ...
     #);
   timeHandlerPrivate: @...;
#)
```

13. The external Library

Interfacing to C and Pascal

The `external` fragment contains a general interface to other programming languages. This interface is provided by the patterns `external` and `cStruct`. The `external` pattern is used to interface to procedures and functions written in other languages e.g. C or Pascal. The `cStruct` pattern is used to be able to create BETA objects with a structure similar to C structs or Pascal records. This interface is used heavily in the system, e.g. in the interface to UNIX and Macintosh, and in the user interface toolkit. The basic `external` and `cStruct` patterns are defined in `betaenv` and the `external` fragment defines the specific attributes of these patterns. Furthermore, the `external` fragment defines the `externalRecord` pattern, which is used for defining the BETA interface to data structures, allocated by the external language (e.g.`C`).

13.1 Using the external Fragment

A program using the`external` fragment will have the following structure:

```
INCLUDE '~beta/basiclib/v1.6/external
--- program: descriptor ---
(# foo: external(# ... #)
do ...
   ... foo ...
#)
```

13.1.1 Interfacing to External C Functions

When interfacing to C, the pattern `callC` must be called in the do-part of the `Exter nal` specialization. The BETA compiler will then generate a call to an external routine with the same name as the BETA pattern, using C’s style of passing parameters.

A pattern of the form

```
foo: external
  (# enter ... do callC exit ... #)
```

describes the interface to an external C function with entry-point `foo`(`_foo`). (The do-part can be left out.) As a convenience, the call to C above need not be specified in which case the compiler will insert it automatically.

If you prefer to give the external a different name from the entry-point name, you can state the entry-point name explicitly. If the entry-point name contains special charac-ters, you are forced to do this:

```
foo: external
  (# ... enter... do 'X$bar' -> callC exit... #);
```

This pattern describes the interface to an external C function, whose entry-point name is `X$bar` (and *not* `foo`).

It is important that there exist a C function with the same name and exactly the number of enter parameters corresponds to the number of parameters of the C function. If the C function returns any results, it is important that an exit parameters is specified in the BETA external pattern, and that this exit value is evaluated in all usage's of this external (due to an error in the current compiler) If the C function does not return any result, *no* exit parameters may be specified.

13.1.1.1 Using call back from C

The following example shows how to install call backs from C to BETA. Readers not familiar with call backs should skip this section.

We use declarations like:

```
callBackProc: external
  (* This pattern describes the interface to the procedure
   * that is called on the call back. It may have the
   * following type definition in C:
   *
   * typedef void (*callBackProcPtr)(int i)
   *)
  (# i: @integer;
  enter i
  (* only the types shortInt, integer, char and boolean
   * can be used in the enter and exit parts
   *)
  do cExternalEntry;
     inner;
     (* Had the return type not been void,
      * the exit part should have appeared here.
      *)
  #);

installCallBack: external
  (* This pattern describes the interface for the C
   * function that installs the call back.
   * It has the following C description:
   *
   * void installCallBack(callBackProcPtr theProcPtr,int j)
   *)
  (# theProcPtr: ##callBackProc;
     j: @integer;
  enter (theProcPtr##,j)
  do callC;
  #);
```

When writing the actual procedure to be called on the call back, it is easiest to specialise the above`callBackProc` pattern, as in:

```
(# ...
   myCallBack: callBackProc
     (# ...
        (* do not specialise the enter part *)
     do 'There is a call back.' -> putline;
        'Value received in parameter i is ' -> puttext;
        i -> putint; '.' -> put; newline;
        (* do not specialise the exit part *)
     #);
   j: @integer;
do 46 -> j;
```

```
      (* install the call back: *)
      (myCallBack##, j) -> installCallBack;
      ...
   #)
```

13.1.2 Interfacing to External Pascal Procedures

When interfacing to Pascal or another programming language with a similar activation record organization the pattern `Pascal` must be called in the do-part of the `External` specialization. The BETA compiler will then generate a call to an external routine with the same name as the BETA pattern, using the Pascal style of passing parameters.

A pattern of the form

```
foo: external
  (# enter ... do Pascal exit ... #)
```

describes the interface to an external Pascal function with entry-point `foo` (`_foo`). (Note the exit parameters that *must* be present in Pascal function interfaces.

A Pascal procedure can be interfaced to through a pattern of the form

```
foo: external
  (# enter ... do Pascal #)
```

If the entry-point of the Pascal function or procedure needs to be explicitly specified, a pattern of the form

```
foo: external
  (# enter ... do 'X$bar' -> Pascal ... #)
```

can be used to describe the interface to an external Pascal procedure, where the entry-point for the Pascal procedure is `X$bar` instead of `foo`.

A pattern of the form

```
foo: external
  (# enter ... do '$...' -> PascalTrap exit ... #)
```

or

```
foo: external
  (# enter ... do '{$...,$...,...}' -> PascalTrap exit ... #)
```

describes the interface to an external Pascal procedure that is called using Motorola traps. The string in the first example is in the form of a single hexadecimal number, preceded with a `$` (e.g. `'$A9FF'`). The string in the second example is in the form of a comma separated list of hexadecimal numbers, each preceded with a `$` and enclosed with braces (e.g. `'{$A9FF,$02F4}'`). Decimal numbers may be used for specifying the traps. This is done by leaving out the `$`.

13.1.2.1 Example Interfacing to Pascal

As an example, an interface routine to a Pascal function (`NewHandle`) may be implemented in the following way:

```
NEWHANDLE: external
  (# theHandle: @integer
  do pascal
  exit theHandle
  #);
```

13.1.2.2 Using call backs from Pascal

Call backs from Pascal is handled similar to call backs from C, except that you should use `pascalExternalEntry` instead of `cExternalEntry`.

13.1.3 Interfacing to External Data Structures

Transferring data to and from the external languages is dealt with through two special purpose patterns: `cStruct` and `externalRecord`.

`cStruct` is the means for specifying a BETA object with a specific storage layout, and with the purpose of transferring this object to the external language for processing. That is, a `cStruct` object is allocated by BETA and made available for processing by the external language.

`ExternalRecord` is the means for specifying a BETA interface into some data structures, allocated by the external language.

13.1.3.1 cStruct

`cStruct` defines `byteSize`[1] that is used for specifying the number of bytes that should be allocated for the BETA object. For specifying the fields, the local patterns `byte`, `short`, `signedShort` and `long` are available. These patterns contain a local virtual attribute, `pos`, that is used to specify the byte position of this field in the `cStruct` object. Note that there is no check for overlapping fields. `cStruct` also defines `put/getByte`, `put/getShort`, `put/getSignedShort` and `put/getLong` operations for accessing the bytes, longs, etc. of the `cStruct` object directly.

13.1.3.2 ExternalRecord

`ExternalRecord` defines `the ptr` attribute, that is used to contain the memory address of the externally allocated data structure. For specifying the interface into the fields of this data structure, `externalRecord` defines `byte`, `short`, `signedShort` and `long` with local virtual attribute, `pos`, to describe the byte position of each field (just as `cStruct`). `ExternalRecord` also defines the `put/getByte`, etc. to make direct access to the bytes, shorts, etc. of the external data structure.

The connection between the `externalRecord` object and the external data structure is established by letting some external routine return the address of the external data structure, and then transfer this integer into the `ptr` attribute of the `externalRecord` object. If it is necessary to transfer this address back to the external language, it can be done by transferring the `ptr` attribute back through some external language routine.

13.1.3.3 Example Using cStruct

The following example shows how to interface to the C language using `cStruct` and `external`. The BETA pattern `myStruct` describes a BETA object to be transferred to the `foo` C function. The BETA pattern `foo` describes the interface to a

[1] Note, that `cStruct` is defined in `betaenv`, and that the `external` library defines additional attributes to this `cStruct` pattern. `ByteSize` is defined in `betaenv`, whereas the rest of the attributes mentioned here, are described in `external`.

C function called `foo`. It is important that there exist a C function with the name `foo` and exactly the same parameters and result.

```
(# myStruct: cStruct
     (* myStruct describes a cStruct consisting of 8 bytes
      * 'a' denotes byte[0]
      * 'b' denotes byte[1]
      * 'c' denotes byte[2-5]
      * 'd' denotes byte[6-7]
      *)
     (# byteSize ::< (# do 8 -> value #);
        a: byte (# pos ::< (# do 0 -> value #) #);
        b: byte (# pos ::< (# do 1 -> value #) #);
        c: long (# pos ::< (# do 2 -> value #) #);
        d: short (# pos ::< (# do 6 -> value #) #)
     #);
   foo: external
     (* This pattern describes the interface to the following
      * C function, called 'foo':
      *
      * int foo(int i, short si, char a, char *t, myStruct *r)
      *)
     (# i: @integer; si: @shortint;
        a: @char; t: [1] @char;
        r: ^myStruct;
        status: @ integer;
     enter (i, si, a, t, r[])
     exit status
     #);
   theStruct: @myStruct;
   m, n, status: @integer;
   c: @char;
do ...
   m -> theStruct.a; (* overflow is not detected *)
   17 -> theStruct.b; ...
   (n, 117, c, 'smith', theStruct[]) -> foo -> status;
   (if status
    //117 then
      theStruct.d -> m;
    ...
   if);
   (11, m, 'x', 'smith', NONE) -> foo -> status;
#)
```

13.1.3.4 Example Using externalRecord

Here `TCSbuffer` is the interface into some data structure in some image processing software. `TCAlloc` is the interface into the C routine in that software, allocating this data structure, and `allocate` is an example of using this routine for getting access to the externally allocated data structure. Finally, `update` illustrates how to transfer the memory address back into the external language.

```
TCSbuffer: externalRecord
  (# display:        @long(# pos ::< (# do 0 -> value #)#);
     window:         @long(# pos ::< (# do 4 -> value #)#);
     visual:         @long(# pos ::< (# do 8 -> value #)#);
     colormap:       @long(# pos ::< (# do 12 -> value #)#);
     depth:     @long(# pos ::< (# do 16 -> value #)#);
     gc:             @long(# pos ::< (# do 20 -> value #)#);
     colorLookup:    @long(# pos ::< (# do 24 -> value #)#);
     width:     @long(# pos ::< (# do 56 -> value #)#);
     height:         @long(# pos ::< (# do 60 -> value #)#);
     data:           @long(# pos ::< (# do 64 -> value #)#);
     xOffset:        @long(# pos ::< (# do 68 -> value #)#);
     yOffset:        @long(# pos ::< (# do 72 -> value #)#);
```

```
        zoom:             @long(# pos ::< (# do 76 -> value #)#);
        updateTile: @long(# pos ::< (# do 80 -> value #)#);
     #);

   TCAlloc: External
     (# width, height: @integer;
        buffer: @integer
     enter (width, height)
     exit buffer
     #);

   allocate:
     (* allocates a true color buffer of resolution
      * width x height
      *)
     (# width, height: @integer; buffer: ^TCSbuffer
        noMemoryError :< TCSnoMemoryError;
     enter(width, height, buffer[])
     do (width, height) -> TCALLOC -> buffer.ptr;
        (if ptr //-1 then noMemoryError if);
        INNER
     #);

   update:
     (* Draws a region of a true color buffer on the window it's
      * associated with.  The x, y, width and height arguments
      * give the location and size of the region in BUFFER
      * coordinates, NOT window coordinates.
      *)
      (# x, y, width, height: @integer; buffer: ^TCSbuffer
         noBufferError :< TCSnoBufferError;
         internError :< TCSinternError;
      enter (buffer, x, y, width, height)
      do (if (buffer.ptr, x, y, width, height) -> TCUPDATE
          // 0 then 'update' -> NoBufferError
          //-4 then 'update' -> internError
         if);
         INNER
      #);
```

13.2 Interface Description for the external Library

```
ORIGIN 'betaenv';
-- CStructLib: attributes---
(*
 * COPYRIGHT
 *       Copyright Mjolner Informatics, 1992-96
 *       All rights reserved.
 *
 ****** Patterns for external interface *****
 *
 * In CStructLib, the operations on a cStruct are defined.
 * The pattern ExternalRecord is an interface to e.g. CStruct objects
 * allocated from C or other external languages.
 *)
```

```
GetByte:
  (# byteoffset: @integer
  enter byteoffset
  do byteoffset->ChkBounds
  exit (R,byteoffset)->TOS'%inxGetByte'
  #);
PutByte:
  (# val,byteoffset: @integer
  enter(byteoffset,val)
  do byteoffset->ChkBounds; (R,byteoffset,val) ->TOS'%inxPutByte'
  #);
GetShort:
  (# byteoffset: @integer
  enter byteoffset do byteoffset->ChkBounds exit (R,byteoffset)-
>TOS'%inxGetShort'
  #);
PutShort:
  (# val,byteoffset: @integer
  enter(byteoffset,val)
  do byteoffset->ChkBounds; (R,byteoffset,val) ->TOS'%inxPutShort'
  #);
GetSignedShort:
  (# byteoffset: @integer
  enter byteoffset
  exit @@R[1]+byteoffset->TOS'%adrGetSignedShort[0]'
  #);
GetLong:
  (# byteoffset: @integer
  enter byteoffset
  do byteoffset->ChkBounds
  exit (R,byteoffset)->TOS'%inxGetLong'
  #);
PutLong:
  (# val,byteoffset: @integer
  enter(byteoffset,val)
  do byteoffset->ChkBounds;
     (R,byteoffset,val) ->TOS'%inxPutLong'
  #);
Byte: (* Used for declaring CStruct fields *)
  (# p: @pos; pos:< IntegerObject; val: @integer;
     set: @(# enter val do (R,p,val)->TOS'%inxPutByte' #);
  enter set
  exit (R,p)->TOS'%inxGetByte'
  #);
Short: (* Used for declaring CStruct fields *)
  (# p: @pos; pos:< IntegerObject; val: @integer;
     set: @(# enter val do (R,p,val)->TOS'%inxPutShort' #);
  enter set
  exit (R,p)->TOS'%inxGetShort'
  #);
SignedShort:
  (# p: @pos; pos:< IntegerObject; val: @integer;
     set: @(# enter val do (R,p,val)->TOS'%inxPutShort' #);
  enter set
  exit @@R[1]+p->TOS'%adrGetSignedShort[0]'
  #);
```

```
Long: (* Used for declaring CStruct fields *)
  (# p: @pos; pos:< IntegerObject; val: @integer;
     set: @(# enter val do (R,p,val)->TOS'%inxPutLong' #);
  enter set
  exit (R,p)->TOS'%inxGetLong'
  #);

--LIB: attributes--
ExternalRecord:
  (* Super-pattern for describing externally allocated record-structures.
   * A call to e.g. a C routine may often return a pointer to a CStruct.
   * By assigning such a pointer to the ptr-field of an externalRecord
object
   * it is possible to interface to such a CStruct.
   *)
  (# ptr: @integer; (* pointer to the external record *)
     GetByte:
       (# byteoffset: @integer
       enter byteoffset
       exit ptr + byteoffset -> TOS'%adrGetByte'
       #);
     PutByte:
       (# val,byteoffset: @integer
       enter(byteoffset,val)
       do (ptr+byteoffset,val) -> TOS'%putByte[0]'
       #);
     GetShort:
       (# byteoffset: @integer
       enter byteoffset
       exit ptr + byteoffset -> TOS'%adrGetShort'
       #);
     GetSignedShort:
       (# byteoffset: @integer
       enter byteoffset
       exit ptr + byteoffset -> TOS'%adrGetSignedShort'
       #);
     PutShort:
       (# val,byteoffset: @integer
       enter(byteoffset,val)
       do (ptr+byteoffset,val) -> TOS'%putShort[0]'
       #);
     GetLong:
       (# byteoffset: @integer
       enter byteoffset
       exit ptr + byteoffset -> TOS'%adrGetLong'
       #);
     PutLong:
       (# val,byteoffset: @integer
       enter(byteoffset,val)
       do (ptr+byteoffset,val) -> TOS'%putLong';
       #);
     Byte: (* For declaring field *)
       (# pos:< IntegerValue; val: @integer; p: @pos;
          set: @(# enter val do (ptr+p,val) -> TOS'%putByte[0]' #)
       enter set
       exit ptr+p -> TOS'%adrGetByte'
```

```
        #);
      Short: (* For declaring field *)
        (# pos:< IntegerValue; val: @integer; p: @pos;
           set: @(# enter val do (ptr+p,val) -> TOS'%putShort[0]' #);
        enter set
        exit ptr+p -> TOS'%adrGetShort'
        #);
      SignedShort: (* For declaring field *)
        (# pos:< IntegerValue; val: @integer; p: @pos;
           set: @(# enter val do (ptr+p,val) -> TOS'%putShort[0]' #);
        enter set
        exit ptr+p -> TOS'%adrGetSignedShort'
        #);
      Long: (* For declaring field *)
        (# pos:< IntegerValue; val: @integer; p: @pos;
           set: @(# enter val do (ptr+p,val) -> TOS'%putLong' #);
        enter set
        exit ptr+p-> TOS'%adrGetLong'
        #);
      DoubleLong: (* For declaring field *)
        (# pos:< IntegerValue; v1,v2: @integer; p: @pos;
           set: @(# enter(v1,v2) do (ptr+p,v1) -> TOS'%putLong';
(ptr+4+p,v2) -> TOS'%putLong' #);
        enter set
        exit (ptr+p-> TOS'%adrGetLong',ptr+4+p-> TOS'%adrGetLong')
        #);
  #) (* ExternalRecord *);

makeCBF: External
  (* Call this external to install a callback and get
   * an integer pointer to it.
   *)
  (# pat: ##External;
     cb: @integer;
  enter pat##
  exit cb
  #);

freeCBF: External
  (* Call this external with an integer pointer to an installed
   * callback (obtained via MakeCBF) when it is certain that the
   * callback will NOT be called again.
   * This will free BETA heap space associated with the callback.
   *)
  (# cbf: @integer;
  enter cbf
  #);
```

14. Index

The entries in the index are the identifiers defined in the public interface of the libraries: The minor level entries refer to identifiers defined local to the identifier of the major level entry. For those index entries referring to patterns with super- or subpatterns within the library, these patterns are specified in special sections of the minor level index for that identifier.