System Utilities − Reference Manual

Mjølner Informatics Report
MIA 94−28
March 2004

Table of Contents

Table of Contents

List of Programs

1 Introduction

The System Utilities is a collection of small libraries. The libraries contains functionality for doing lowlevel operations. These operations should be used with care and are not for use in most programs.

Some of the libraries is designed for use in very special situations, and is therefore not neccesarily general purpose libraries and can be tricky to use.

Most of the libraries is in the form of one pattern with a set of attributes. This manual is short in its description. Please refer to the interface files for accurate description of patterns.

2 Binary Stream

The Binary Stream library allows the user to create an stream in a repetition. The stream is kept in memory and can be saved in a file and restored again.

Binary Stream allows the user to mix char/shorts/integers in a single repetition. Alignment is handled automatically, but the repetition contains no type information. This means that you can write chars, and later read shorts!

Every operation updates the current position automatically. Binary Stream has the following attributes:

- put: appends a char at the end of the stream
- putShort: appends a short
- putLong: appends a long
- putText: appends a text
- get: reads a char at the front of the stream
- getShort: reads a short
- getLong: reads a long
- getText: reads a text, (zero terminated)
- save: stores the repetition in a file.
- restore: reads the repetition from a file. Sets position to start.
- fragRestore: Like restore, but byte 2 to 5 is the length to read.

Binary stream has the following exceptions:

- AccessError: raised when file is unaccesable (eg. does not exist)
- NoSpaceError: disk full
- WriteError: unable to write to file.

In addition to this, the patterns save,restore,fragRestore also has local exceptions which is also raised.

See the Interface file for details.

3 C−String Library

The C−String library can be used when you need to exchange strings with program parts that is not written in BETA, eg. system−calls.

A C−string is a null−terminated array of chars.

The strings can be exchanged both ways, that is, you can create strings (texts) in BETA and give the to a C−function or you can create a string in C and use it later in BETA.

The C−String pattern has the following attributes:

- charptr: an integer, which value points to the string
- init: allocates space for this string. Default is 256 bytes
- set: if necessary expands the string and copies the chars from a BETA text object
- get: creates a BETA text object, and copies the c−string.
- getChars: returns the string as a char repetition.
- length: returns length (zero terminated)
- inxPut: puts a char at a specific position
- inxGet: receives a char
- toLocal: converts all newlines to system standard.
- fromLocal: conterts system standard end−of−line to newlines
- free: frees the allocated space. Must be used if init is called twice.

The charPtr attribute can also be changed/updated by using the enter and exit parts of the cstring pattern.

In addition, the library contains declarations for standard C−routines to manipulate strings. Note that cstring.charptr can be used as argument to these C−routines.

4 Endian Library

The Endian Library contains a few patterns to test for endianess and to convert data.

- isBigEndian: returns true if this machine is big endian
- isLittleEndian: returns true if this maching is little endian
- ntohl: converts a integer to host byteorder
- ntonl: converts a integer to network byteorder
- ntohs: converts a short to host byteorder
- ntons: converts a short to network byteorder

See the Interface file for details.

5 Environment Library

The Environment Library contains a single pattern expandEnvVar. ExpandEnvVar enables the user to look up variables in the process environment.

ExpandEnvVar is used like this:

Program 1: Using expandEnvVar

```
(# var:^text;
do 'TMP'->var[];
   var->expandEnvVar
        (# defaultvalue::
           (# do 'c:\windows\temp'->envvarvalue[]
            #)
         #)->screen.putline
#)
```

In this example, the environment variable TMP is looked up in the environment. If TMP was not defined the value 'c:\windows\temp' is used as default value. The value found is printed using screen.putline.

Generaly, if the environment variable is not found in the environment defaultvalue can be used to specify a value. Note that the defaultvalue is not set in the process environment.

6 ErrorScreen Library

The ErrorScreen library defines a new pattern "errorscreen". Errorscreen is a subpattern of pattern stream.

Errorscreen can be used just like the stream instance **screen** but instead of writing output to standard output errorscreen writes to standard−error.

ErrorScreen defines no new attributes.
Note: you can not read from ErrorScreen.

7 Heap Info Library

The Heap Info Library gives the programmer access to information about the memory heaps of the program. BETA programs uses 3 different heaps:

- Infant−Object−Area (IOA): A dobbelt heap for newly instantiated object. Garbage collected using a stop−and−copy algorithm. When a object reachs a certain age is it copied to AOA.
- Adult−Object−Area (AOA): A large heap garbage collected using a freelist. This heap is also used for repetitions.
- CBFA (callbacks): A area used for callback structures.

The library defines two patterns, getHeapInfo and PrintHeapUsage.

PrintHeapUsage prints information on heap usage on standard output. It is used like this:

NAME 'MyProgram'
INCLUDE '~beta/sysutils/heapinfo'
PROGRAM: descriptor
(# do ... 'Heap Info at program point a'−>PrintHeapUsage; ... #)

getHeapInfo gives access to the individual data. The use is like this:

Program 2: Getting Heap Informations

```
(# ioasize:@integer;
do HeapInfoIOASize−>getHeapInfo−>ioasize;
   'my IOA−size is %d\n'
     −>putformat(# do ioasize−>d #);
#)
```

The argument to getHeapInfo is predefined constants. getHeapInfo evaluates to an integer. The library defines the following constants to be used with getHeapInfo:

- HeapInfoIOA
- HeapInfoIOALimit
- HeapInfoIOATop
- HeapInfoIOASize
- HeapInfoIOAActive
- HeapInfoIOAspace
- HeapInfoCBFA
- HeapInfoCBFALimit
- HeapInfoCBFATop
- HeapInfoCBFABlockSize
- HeapInfoCBFAsize
- HeapInfoAOABaseBlock
- HeapInfoAOATopBlock
- HeapInfoAOABlockSize

- HeapInfoAOATotalSize
- HeapInfoAOATotalFree

8 Object Interface Library

The Object Interface Library is a collection of patterns that forms a lowlevel runtime interface to objects. It allows the user to read and manipulate the runtime state of objects and prototypes.

This is a short description of the functionality. Please refer to the interface file for parameters etc.

8.1 ProtoTypes

Each object−descriptor in a BETA fragments gives rise to a prototype.
The library defines the following patterns for dealing with prototypes:

- protoType: Prototype is a highlevel model of the lowlevel prototype. It has the following attributes:
  - ♦ address: is the memory address of the actual prototype.
  - ♦ GCtab: is the distance to allocation tables, used by garbage collector.
  - ♦ OriginOff: is the offset of the origin field.
  - ♦ GPart: is the address of the generation entry point corresponding to the prototype.
  - ♦ Size: is the size of the objects with this prototype.
  - ♦ Prefix: is the address of the prefix (super) pattern.
  - ♦ LabId: is the name of objects used in dumps.
  - ♦ GroupId: is the name of the file, where the pattern is declared.
  - ♦ ScanRefs: scans through references in this prototype. See later.
  - ♦ ScanSimples: scans through simple attributes in this prototype.
  - ♦ AstIndex: is the ast−index corresponding to this prototype. Must be doubled before used with indexToNode. Eg. pt.astIndex*2−>ff.indexToNode;
  - ♦ formIndex: this is the index of the form containing the descriptor corresponding to this prototype.
- getProtoTypeForStruc: Returns the prototype corresponding to a pattern−variable.
- getProtoType: Returns the prototype corresponding to a object.
- getProtoField: Returns the prototype−field of a object. Will not work on components.
- putProtoField: Sets the prototype−field of a object.
- IsPrototypeOfProcess: Returns true if the address is a prototype in the program.

ScanRefs: The ScanRefs scans a prototype for references. You can use it to investigate the references in a object. It calls INNER for every reference. IsStatic is true if the reference is static. thisProto is the prototype address of static inlined objects. refType is the type of the dynamic reference.

Here is an example from the sysutils/demo/objectinterface/objint.bet demo:

```
proto.scanRefs
  (#
  do (if isStatic then
         '\tStatic reference at offset  ' −> puttext
      else
         '\tDynamic reference at offset ' −> puttext
     if);
     thisOffset −> putint; newline
  #);
```

8.2 BETA Components

The library defines the following patterns for

- isComponent: Determines if a object is a BETA component.
- objectToComponent: Used to convert a object reference to a component reference.
- componentToObject: Used to convert a component reference to a object reference.

ObjectToComponent is often needed in situation where the THIS construct is used on components. Eg:

```
(# mySystem:System
   (# ...
      myfork:
        (#
        do THIS(system)[]->objectToComponent->fork;
                 (* fork is in basicsystemenv.bet *)
        #);
   #)
#);
```

"THIS(System)" is a object reference and fork needs a component reference.

8.3 Objects

The library defines the following patterns for

- getPatternName: Return the name of the pattern this object is instance of.
- getOrigin: Returns the origin (enclosing) object.
- getGCField: Return the GC field of the object.
- addressToObject: Converts a address (integer) to a object reference.
- printObject: Prints a low−level textual form of a object.
- assignRef: Assigns a object reference to a object field using integer addresses.
- getOIDforObject: Returns a unique integer object ID.
- getObjectByOID: Return a object reference from a integer object ID.

8.4 Groups

The library defines the following patterns for handling groups.
These are only for very advanced use.

- group_header: is a highlevel description of objectcode−groups.
- NextGroup: returns the next/first group in the executable.
- NameOfGroup: returns address of the groups groupname.
- AddGroup: Appends a new groups to the list.
- IsPrototypeOfGroup: returns true if a prototype is in a group.

8.5 Miscellaneous

The library also contains this pattern:

- extGetCstring: converts a address of a c−string to a BETA char repetition.

9 Pathhandler Library

The Pathhandler library is used to convert directory names. It contains the following patterns:

- DirectoryChar: Returns the char which the operating system uses as delimiter in path names.
- CurrentDirectory: Returns the pathname of the process working directory.
- scanSearchPath: Scan the search path of the process calling inner for each path. The user can specify a new path and a new delimiter if needed.

9.1 Pathhandler

The Pathhandler is used to convert paths with the ~beta notation to absolut path names. It has the following attributes:

- BetaLib: returns the path to the installation of the Mjølner System.
- CurrentDirectory: returns the working directory.
- ConvertFilePath: converts a "relative path" relative to a basic and returns the absolut path.
- LocalPath: Given a path and a basis, exit the local path relative to the basis, if possible.

The FilenameConverter is a subpattern of the pathhandler. This pathhandler is capable of keeping track of files which have allready been "converted". ConvertFilePath will return the same pathname every time the same file is accessed even if the absolute path−names differs.

See the Interface file for details.

10 Repetition Object Library

The Repetition Library contains a single pattern, RepetitionObject. This pattern can be used to pass serialized objects around, eg. through sockets. It has attributes that make it simple to pack different simple data into a single repetition.

It has the following attributes:

- initialRange: Virtual which specifies the initial size of the repetition.
- size: The size of the used space.
- pos: The next position to be used for get or put.
- get: Reads a long from the repetition a the current position.
- getLong: Like get, bug swapping to big endian on little endian machines.
- Peek: Like get but does not update position.
- PeekLong: Like getLong but does not update position.
- put: Puts a integer and updates position.
- putLong: Does not update position.
- cheapPut: Like put, but the user must ensure that there is room enough in the repetition.
- cheapPutLong: Like cheapput.
- makeSpace: Expands the repetition.
- checkSize: Updates the size to be at least the position − 1.
- getText: Reads a text from the repetition.
- putText: Writes a text to the repetition. Also expands if neccessary.
- reset: Resets the repetition. Sets pos to the start, but does not clear or frees the space.
- init: Must be called initially.
- checkGet: Checks that the position is legal, ie. less that size.
- checkGetText: Checks that a text can be legally read.

11 Scan Object Library

This Library contains support for scanning all objects at a point of time in the execution of any BETA program.

The main pattern is "scanPrefix:"

```
scanPrefix:
  (# callback:< scanCallback;
     root: ##object;
     printVisited,printOrigin,printSize: @boolean;
  do INNER
  #);
```

If you specialize the 'callback' virtual, it will be executed once for each object visited, with 'obj' referring to the object being visited. It will never be executed for a non−object, such as a repetition etc.

The pattern variable 'root' is used to select which objects to visit: an object is visited iff it is an instance of a specialization of the pattern denoted by 'root'. If 'root' is NONE, all objects and non−objects are visited. If (root## = object##), all objects are visited.

If 'printVisited' is true, a message is printed describing the pattern of each visited object, and giving the category of each visited non−object, such as '[Value Repetition]'.

When 'printVisited' is true, other booleans can be used to enhance the output for objects: If 'printOrigin' is true, the origin of each visited object is printed. If 'printSize' is true, the size in bytes of each visited object is printed.

If a garbage collection runs during a scan, it will be aborted with an error message, but the program continues otherwise unaffected. To avoid this, try to create as few objects as possible in callback. (If callback is not specialized, it will not get called, and no objects will be created during the scan.)

12 Time Library

The Time Library contains a few patterns concerning the system time:

SystemTime This return the system time in a integer. This is system dependent.

PreciseTime As SystemTime but this also returns the millisecond resolution of time. The accurracy is system dependent.

CpuTime This returns the amount of CPU time in milliseconds used since the first call of Cputime in this program. May not work on all platforms.

FormatTime This converts a system time integer to a readble text format.

See the Interface file for details.

13 Time History

The Time History Library contains a single pattern:
TimeHistory: this pattern keeps track of a history of CPU time. If called with true current time and used CPU time is printed on a stream or the screen.

See the Interface file for details.

14.1 Binarystream Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'sysutilsbinstream' '../lib';
(*
 * COPYRIGHT
 *   Copyright Mjolner Informatics, 1995-98
 *   All rights reserved.
 *
 * For constructing an in-memory byte stream in a repetition
 * that can be saved and/or restored on a file.
 * Used by the compiler and valhalla for generating
 * and reading '.db' files.
 *)
BODY 'private/binarystreambody';

---LIB:attributes---

BinaryStream:
  (# init:< (# ... #);
     put: @
       (# ch: @char
       enter ch
       ...
       #);
     putShort: @
       (# V: @integer
       enter V
       ...
       #);
     putLong: @
       (# V: @integer
       enter V
       ...
       #);
     putText: @
       (# T: ^text
       enter T[]
       ...
       #);
     get: @
       (# ch: @char
       ...
       exit ch
       #);
     getShort:  @
       (# V: @integer
       ...
       exit V
       #);
     getLong: @
       (# V: @integer
       ...
       exit V
       #);
     getText: @
       (# T: ^text
       ...
       exit T[]
       #);
     save:
       (# openFailed:< Exception;
          FN: ^text
       enter FN[]
```

```
      do ...
      #);
    restore:
      (# notFound:< Exception;
         FN: ^text
      enter FN[]
      do ...
      #);
    fragRestore:
      (# notFound:< Exception;
         FN: ^text
      enter FN[]
      do ...
      #);
    (* The folowing exceptions may not be sufficient;
     * the exception model is clumsy since the current version
     * is a quick merge of the two binarystream versions used by
     * the compiler and valhalla. The compiler uses the exceptions below
     * but they are not used by valhalla; continue is therefore
     * set to true. Valhalla uses the openFailed/notFound execptions
     * in save,restore and fragRestore
     *)
    AccessError:<
      Exception(# FN: ^text enter FN[] do true -> continue; INNER #);
    NoSpaceError:<
      Exception(# FN: ^text enter FN[] do true -> continue; INNER #);
    WriteError:<
      Exception(# FN: ^text enter FN[] do true -> continue; INNER #);

    rep: @ ...
#)
```

14.2 Cstring Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'sysutilscstring' '../lib';
BODY 'private/cstringbody';

(*
 * COPYRIGHT
 *       Copyright Mjolner Informatics, 1992-2000
 *       All rights reserved.
 *)

-- LIB: attributes --

CString:
  (* BETA interface to standard NULL-terminated C-strings, externally
   * allocated, e.g., using malloc.
   *)
  (# charptr: @integer
       (* The pointer to the external string *);
     init:
       (* Allocate length bytes (default 256) using malloc, and assign the
        * result to charptr. You may want to explicitly free THIS(CString)
        * before a reinitialization.
        *)
       (# length: @integer;
       enter length
       do ...
       #);
     set:
       (* Initializes THIS(CString) to be large enough to hold the chars of
        * T, and then copies the chars of T to THIS(CString). You may want to
        * explicitly free THIS(CString) before this operation, if this is a
        * reinitialization.
        * For convenience, it also exits the char pointer.
        *)
       (# T: ^text;
       enter T[]
       do ...
       exit charptr
       #);
     get:
       (* Return the chars of THIS(CString) copied into the text T *)
       (# T: ^text;
       do ...
       exit T[]
       #);
     getChars:
       (* Return a repetition, with the chars of THIS(CString) copied to it *)
       (# R: [0]@char;
       do ...
       exit R
       #);
     length: IntegerValue
       (* Return the string-length if THIS(CString), i.e., the number of chars
        * before the NULL-char in the externally allocated string.
        *)
       (# ... #);
     inxPut:
       (* Put ch at index inx in the externally allocated CString. The first char
        * is at index 0. NOTICE: There is no index check, it is  the programmer to
        * ensure, that it is allowed to put at the given index.
        *)
       (# inx: @integer;
```

```
            ch: @char;
          enter (inx, ch)
            ...
          #);
        inxGet:
          (* Obtain the char at the given index in the externally allocated string.
           * The first char is at index 0.
           *)
          (# inx: @integer;
            ch: @char;
          enter inx
            ...
          exit ch
          #);
        toLocal:
          (* Convert a BETA CString with newlines to a string containing system
           * line terminators.
           *)
          (# new: ^CString;
          ...
          exit new[]
          #);
        fromLocal:
          (* Convert a CString containing system line terminators to a BETA
           * string with the proper newline character
           *)
          (# new: ^CString
          ...
          exit new[]
          #);
        free:
          (* Free the externally allocated string. The chars of the externally
           * allocated string will be unaccessible after this operation.
           *)
          (# ... #);

        <<SLOT CStringLib: attributes>>;

    enter charptr
    do INNER
    exit charptr
    #);

(* Standard C−library operations to perform on CStrings *)

strlen: External
    (# charptr: @integer;
       result: @integer;
    enter charptr
    exit result
    #);

strcat: External
    (# charptr1, charptr2: @integer;
       result_ptr: @integer;
    enter (charptr1, charptr2)
    exit result_ptr
    #);

strncat: External
    (# charptr1, charptr2: @integer;
       n: @integer;
       result_ptr: @integer;
    enter (charptr1, charptr2, n)
    exit result_ptr
    #);
```

```
strdup: External
  (# charptr: @integer;
     result_ptr: @integer;
  enter (charptr)
  exit result_ptr
  #);

strcmp: External
  (# charptr1, charptr2: @integer;
     result: @integer;
  enter (charptr1, charptr2)
  exit result
  #);

strncmp: External
  (# charptr1, charptr2: @integer;
     n: @integer;
     result: @integer;
  enter (charptr1, charptr2, n)
  exit result
  #);

strcpy: External
  (# charptr1, charptr2: @integer;
     result_ptr: @integer;
  enter (charptr1, charptr2)
  exit result_ptr
  #);

strncpy: External
  (# charptr1, charptr2: @integer;
     n: @integer;
     result_ptr: @integer;
  enter (charptr1, charptr2, n)
  exit result_ptr
  #);

strchr: External
  (# charptr: @integer;
     c: @integer;
     result_ptr: @integer;
  enter (charptr, c)
  exit result_ptr
  #);

strrchr: External
  (# charptr: @integer;
     c: @integer;
     result_ptr: @integer;
  enter (charptr, c)
  exit result_ptr
  #);

strpbrk: External
  (# charptr1, charptr2: @integer;
     result_ptr: @integer;
  enter (charptr1, charptr2)
  exit result_ptr
  #);

strspn: External
  (# charptr1, charptr2: @integer;
     result: @integer;
  enter (charptr1, charptr2)
  exit result
```

```
  #);

strcspn: External
  (# charptr1, charptr2: @integer;
     result: @integer;
  enter (charptr1, charptr2)
  exit result
  #)
```

14.3 Endian Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'sysutilsendian' '../lib';
(* Operations for testing whether or not the host system uses
 * big endian or little endian data representation.
 * In addition there are operations for converting a long (4bytes)
 * and a short (2 bytes) memory location from big/little to
 * little/big endian.
 *)
MDBODY linux   'private/endian_littleBody'
       nti     'private/endian_littleBody'
       x86sol  'private/endian_littleBody'
       default 'private/endian_bigBody';

---lib:attributes---

(* Predicates indicating the endian-type of the host system. *)

isBigEndian:
  (# b: @boolean
    ...
  exit b
  #);
isLittleEndian:
  (# b: @boolean
    ...
  exit b
  #);

(* BSD like conversion routines. These operations are safe. *)

ntohl:
  (# i: @Integer;
  enter i
  ...
  exit i
  #);
htonl:
  (# i: @Integer;
  enter i
  ...
  exit i
  #);

ntohs:
  (# i: @Integer;
  enter i
  ...
  exit i
  #);
htons:
  (# i: @Integer;
  enter i
  ...
  exit i
  #)
```

14.4 Envstring Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'sysutilsenvstring' '../lib';
BODY 'private/envstringbody';
(*
 * COPYRIGHT
 *       Copyright Mjolner Informatics, 1992-96
 *       All rights reserved.
 *)
--- LIB: attributes ---
expandEnvVar:
  (* Expands text containing substrings of the form $(VAR) to the value
   * of the corresponding unix-environment variable.
   *
   * E.g., if the unix-command printenv says
   * HOME=/users/cn
   * SHELL=/bin/csh
   * USER=cn
   *
   * then                  is expanded to
   *
   *  $(SHELL)               /bin/csh
   *  $(HOME)!$(USER)        /users/cn!cn
   *
   * (if an environment variable is specified, which is not set, then the
   * virtual pattern defaultvalue is called. This allows the programmer to
   * specify default-values for not-set environment variables.
   *)
  (# t: ^text;

     defaultValue:< (# envvar,envvarvalue: ^text
       enter envvar[]
       do INNER
       exit envvarvalue[]
       #);
     expand: @...
  enter t[]
  do expand
  exit t[]
  #)
```

14.5 Errorscreen Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'sysutilserrscr' '../lib';
BODY 'private/errorscreenbody';

-- LIB: attributes --
ErrorScreen: Stream
  (* Pattern corresponding to the Screen pattern of betaenv,
   * but which outputs to the standard error instead of to the
   * standard output, as Screen does
   *)
  (# Eos::< (# do true->value #);
     OtherError::< (# ... #);
     Get::< (# ... #);
     Put::< (# ... #);
     PutText::< (# ... #);
  #)
```

14.6 Heapinfo Interface

```
ORIGIN '~beta/basiclib/betaenv'

--lib:attributes--

HeapInfoIOA:           (# exit 0 #);
HeapInfoIOALimit:      (# exit 1 #);
HeapInfoIOATop:        (# exit 2 #);
HeapInfoIOASize:       (# exit 3 #);
HeapInfoIOAActive:     (# exit 4 #);
HeapInfoIOAspace:      (# exit 5 #);
HeapInfoCBFA:          (# exit 10 #);
HeapInfoCBFALimit:     (# exit 11 #);
HeapInfoCBFATop:       (# exit 12 #);
HeapInfoCBFABlockSize: (# exit 13 #);
HeapInfoCBFAsize:      (# exit 15 #);
HeapInfoAOABaseBlock:  (# exit 30 #);
HeapInfoAOATopBlock:   (# exit 31 #);
HeapInfoAOABlockSize:  (# exit 33 #);
HeapInfoAOATotalSize:  (# exit 35 #);
HeapInfoAOATotalFree:  (# exit 36 #);
(* get information from runtimesystem about heap usage.
 * use constants from above.
 *)

getHeapInfo: external
  (# infoId: @integer;
     heapAmount: @integer
  enter infoId
  do callC
  exit heapAmount
  #);

(* print heap information on stdout.
 * Text marked with prompt.
 *)
PrintHeapUsage: external
  (# prompt:[1]@char;
  enter prompt
  do CallC;
  #)
```

14.7 Objinterface Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'sysutilsobjint' '../lib';
BODY 'private/objinterfacebody';

(* The lowlevel BETA interface to BETA objects
 *
 * COPYRIGHT
 *       Copyright Mjolner Informatics, 1992-96
 *       All rights reserved.
 *)
--- LIB: attributes ---

(* Possible values of protoType.scanRefs.refType if isStatic is FALSE: *)
REFTYPE_DYNAMIC: (# exit 0 #);
REFTYPE_OFFLINE: (# exit 1 #);
REFTYPE_ORIGIN: (# exit 2 #);

(* Possible values of protoType.scanSimples.simpleType.
 * Must be disjoint from the REFTYPEs, as they're
 * assigned to the same repetitions. *)

SIMPLETYPE_INT16: (# exit 64 #);
SIMPLETYPE_INT32: (# exit 65 #);
SIMPLETYPE_REAL64: (# exit 66 #);

(* The following are used by the objectserializer, not by scanSimples,
 * but they are intermixed in the ObjectServer.  Therefore their valueset
 * must be disjoint.
 *)
SIMPLETYPE_HANDLED: (# exit 67 #); (* handled by previous entry in memory *)
SIMPLETYPE_BIN32:   (# exit 68 #); (* 4 bytes binary data *)
SIMPLETYPE_INT16BIN16: (# exit 69 #);
SIMPLETYPE_BIN16INT16: (# exit 70 #);
SIMPLETYPE_INT16INT16: (# exit 71 #);

(* no need yet: SIMPLETYPE_BIN16INT16: (# exit 71 #); *)

COMProtoTypes: (# exit true #);
DISP_OFF: (# exit 24 #);

protoType:
  (* Pattern used to access various attributes of the BETA prototype located
   * at address 'address'
   *)
  (# beta: @...;
     address: @integer; (* The memory address of THIS(prototype) *)

     GCtab: IntegerValue
       (* Value of the GCtab field, i.e. distance to the allocation
        * tables in THIS(Prototype), used by the garbage collector.
        *)
       (# ... #);
     OriginOff: IntegerValue
       (* Value of the OriginOff field, i.e. the offset of the Origin field
        * in having THIS(Prototype).
        *)
       (# ... #);
     GPart: IntegerValue
       (* Address for generation entry point corresponding to THIS(Prototype)
        *)
       (# ... #);
     Size: Integervalue
```

```
  (* The size in longs of objects having THIS(prototype) *)
  (# ... #);
Prefix: IntegerValue
  (* The prototype of the prefix *)
  (# ... #);
LabId:
  (* exit the name of used in 'beta.dump' for the pattern corresponding
   * to THIS(Prototype)
   *)
  (# id: ^text;
     doIt: @...
  do doIt
  exit id[]
  #);
GroupId:
  (* exit the name of the file, the pattern corresponding to
   * THIS(Prototype) is declared in. This is also used in 'beta.dump'.
   * This method is pretty slow as it involves a search through a
   * linked list.
   *)
  (# id: ^text;
  do ...
  exit id[]
  #);
scanRefs:
  (* Scan through references in THIS(Prototype). The boolean isStatic
   * indicates whether the current reference is static or dynamic.
   * If isStatic is TRUE, thisProto contains the address of the
   * prototype of the static inlined partobject.
   * If isStatic is FALSE, refType indicates whether the dynamic reference
   * is an origin reference, a reference to an offline allocated static
   * part, or an ordinary dynamic reference. Possible values are
   * REFTYPE_DYNAMIC, REFTYPE_OFFLINE and REFTYPE_ORIGIN.
   * thisOffset tells at what offset the reference is placed in objects
   * having THIS(prototype).
   * thisAddr is the memory address thisOffset is found in.
   *)
  (# isStatic: @boolean;
     thisOffset,thisAddr,thisProto,thisOriginOff: @integer;
     refType: @Integer;
  do ...
  #);
scanSimples:
  (* Scan through "simple values" in THIS(Prototype).
   * thisOffset tells at what offset the value is placed in objects
   * having THIS(prototype).
   * simpleType holds the type of the value. Possible types are
   * SIMPLETYPE_INT32, SIMPLETYPE_INT16, SIMPLETYPE_REAL64.
   *)
  (# thisOffset, simpleType: @Integer;
  do ...
  #);
astIndex: IntegerValue
  (* exit the astIndex corresponding to this prototype.
   * Please note, that the value returned is not immediately useful
   * for accessing the ast through MPS.  The value returned _must_
   * be multiplied with 2 (two) in order to be useful for MPS.
   * I.e. you should write something like:
   *          pt.astindex*2->ff.indexToNode
   * where pt: @prototype and ff: @fragmentForm
   *)
  (# ... #);
formIndex: IntegerValue
  (* exit the index of the form containing the objectDescriptor
   * corresponding to this prototype. The value returned is an index
   * into the fragmentList of the group. *)
```

```
      (# ... #);
  enter address
  exit address
  #); (* Prototype *)

getProtoTypeForStruc:
  (* exit the prototype corresponding to structure## *)
  (# structure: ##object;
     protoAdr: @Integer;
  enter structure##
  ...
  exit protoAdr
  #);

getProtoType:
  (* exit the prototype for obj *)
  (# obj: ^object;
     protoAdr: @Integer;
  enter obj[]
  ...
  exit protoAdr
  #);

isComponent: BooleanValue
  (* Exits true if obj is a component. *)
  (# obj: ^Object;
  enter obj[]
  ...
  #);

objectToComponent:
  (* Exits the component part of the object entered *)
  (# comp: ^|Object;
     obj: ^Object;
  enter obj[]
  do ...
  exit comp[]
  #);

componentToObject:
  (* Exits the item part of the object entered *)
  (# comp: ^|Object;
     obj: ^Object;
  enter comp[]
  do ...
  exit obj[]
  #);

getPatternName:
  (* exit the name of the pattern obj is an instance of *)
  (# obj: ^object;
     name: ^text
  enter obj[]
  do ...;
  exit name[]
  #);

getOrigin:
  (* exit the object that is the origin of obj. The origin of a BETA object is
   * the object statically enclosing the *pattern* the object is an instance
   * of.
   *)
  (# obj: ^object;
     doIt: @...
  enter obj[]
  do doIt
```

```
  exit obj[]
  #);

getProtoField: IntegerValue
  (* exit the value of the protoType Field for obj *)
  (# obj: ^object;
  enter obj[]
  ...
  #);

putProtoField:
  (* Put 'value' into the prototype field of 'obj *)
  (# value: @integer;
     obj: ^Object;
  enter (obj[], value)
  ...
  #);

getGCField: IntegerValue
  (# obj: ^object;
  enter obj[]
  ...
  #);

addressToObject:
  (* Given a memory address, exit the BETA object at that address.
   * You must have a static instance of addressToObject when using it.
   *)
  (# addr: @integer;
     obj: ^object;
  enter addr
  ...
  exit obj[]
  #);

printObject:
  (* Print obj in textual form on the stream s. If s is NONE, Screen is used *)
  (# obj: ^object;
     s: ^stream;
  enter (obj[],s[])
  do ...
  #);

extGetCstring: external
  (* Given the address of a zero−terminated string, returns the string. *)
  (# stringAdr: @Integer;
     s: [1]@Char;
  enter stringAdr
  do 'copyInput' −> callC;
  exit s
  #);

assignRef: external
  (* Given the address of an object in objAdr, and the address of a reference
   * field in another object in fieldAdr, assignRef puts the object reference
   * into the field.
   * Use this instead of directly assigning the object address to the field
   * as this will sooner or later give GC problems. *)
  (# fieldAdr, objAdr: @Integer;
  enter (fieldAdr, objAdr)
  #);

group_header: data
  (# data_start: ^group_header;
     protoTableAdr: @Integer;
     data_end: ^group_header;
```

```
      code_start: @Integer;
      code_end: @Integer;
      groupNameAddr: @integer;
      unique_group_id_hash: @Integer;
      unique_group_id_modtime: @Integer;
      ptr: @Integer
  #);

NextGroup: external
  (* current is a pointer to a group header. NextGroup returns the
   * group after current in the executable. If current is NONE, the
   * first group is returned.
   *)
  (# current: ^group_header;
  enter current[]
  exit current[]
  #);
NameOfGroup: external
  (* Returns the groupname address of the group whose header is
   * given as parameter.
   *)
  (# group: ^group_header;
     nameAdr: @Integer;
  enter group[]
  exit nameAdr
  #);
AddGroup: external
  (* Add new_group to list of known data−segment headers
   * in runtime system − used by e.g. NextGroup.
   *)
  (# new_group: @integer (* address of start of data segment *)
  enter new_group
  #);
IsPrototypeOfGroup: External
  (* Exits true if data_addr is the address of
   * a prototype in the fragment group corresponding
   * to gh.
   *)
  (# gh: ^group_header;
     data_addr: @integer;
     is_proto_in_group: @boolean;
  enter (gh[], data_addr)
  exit is_proto_in_group
  #);
IsPrototypeOfProcess: External
  (* Exits true if data_addr is the address of
   * a prototype in the fragment groups constituting
   * the current program.
   *)
  (# data_addr: @integer;
     is_proto_in_group: @boolean;
  enter data_addr
  exit is_proto_in_group
  #);

(* Prototype constants for special object types *)
ComponentPTValue:   (# exit −1 #);
StackObjectPTValue: (# exit −2 #);
StructurePTValue:   (# exit −3 #);
RefRepPTValue:      (# exit −4 #);
ValRepPTValue:      (# exit −5 #);
ByteRepPTValue:     (# exit −6 #);
WordRepPTValue:     (# exit −7 #);
DoubleRepPTValue:   (# exit −8 #);
DopartObjectPTValue:(# exit −9 #);
DynItemRepPTValue:  (# exit −10 #);
```

```
DynCompRepPTValue:  (# exit −11 #);

getOIDforObject:
  (* exits a unique Object Identifier, that is valid for the
   * lifetime of this program run.  The object may die, though,
   * if no references to it exist.
   * Having an OID does NOT count as a root for the Garbage Collector.
   *)
  (# obj: ^Object;
     OID: @Integer;
  enter obj[]
  ...
  exit OID
  #);

getObjectByOID:
  (* Finds the object associated with the Object Identifier.
   * Note! Having an OID does NOT count as a root for the Garbage Collector.
   *)
  (# obj: ^Object;
     OID: @Integer;
  enter OID
  ...
  exit obj[]
  #)
```

14.8 Pathhandler Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'pathhandler' '../lib';
BODY 'private/pathhandlerbody';
(*
 * COPYRIGHT
 *       Copyright Mjolner Informatics, 1992-96
 *       All rights reserved.
 *)
-- LIB: Attributes --
(* idx=2 *) (* Utility patterns *)
DirectoryChar: charValue
(* The character which is used in this operating system to merge local
 * names to full path-names. Placed here to allow use without having
 * an instance of pathhandler.
 *) (# ... #);
DirectoryCharRegExp:
(* The character(s) which are used in this operating system to merge local
 * names to full path-names.  On DOS/Windows this is a regexp that matches
 * \ or /
 *)
  (# t: ^text;  ... exit t[] #);
IsDirectoryChar: booleanValue
(* The difference between this and the above is that on DOS/Windows
 * both / and \ are valid directory chars, but the canonical one is
 * \.
 *) (# c: @char;  enter c ... #);
CurrentDirectory:
(* The current working directory of this process.
 * Placed here to allow use without having an instance of pathhandler.
 *)
  (# cwd: ^text;  ... exit cwd[] #);
searchPath:
  (* Default program search path on this platform *)
  (# path: ^text;
  ...
  exit path[]
  #);
searchPathDelimiter: charValue
  (* Default separator between directories in search paths on this platform *)
  (# ... #);
scanSearchPath:
(* Scan through a list of paths separated by the delimiter
 * specified. INNER is called for each path found in
 * the list.
 * If pathlist is none, the default searchPath is used.
 * If delimiter is 0, the default searchPathDelimiter is used.
 *)
  (#
     pathlist: ^text;
     (* list if paths seperated by delimiter *)
     delimiter: @char;
     path: ^text;
     (* pathlist component set before each call of INNER *)

  enter (pathlist[],delimiter)
  ...
  #);
applDirectory:
  (# path: ^text
  ...
  exit path[]
  #);
```

```
PathHandler:
(*
 *  This pattern converts the Mjolner path notation to and from full native paths.
 *
 *  The following abbreviations are handled:
 *
 *      ~    is expanded into the home directory of the user
 *      ~foo is expanded into the home directory of foo
 *      .    is interpreted relatively to another path
 *           (see ConvertFilePath below)
 *      ..   is interpreted relatively to another path
 *           (see ConvertFilePath below)
 *
 *      Environment variables like $(HOME) are expanded within paths.
 *
 *  PathHandler examines the environment variable
 *
 *      BETALIB. If set, this determines the home directory of the Mjolner
 *               BETA system.
 *
 *      The value of BETALIB may, but need not, be terminated by '/'.
 *)
  (#
     <<SLOT PathHandlerLib:Attributes>>;
     BetaLib:
     (* Mjolner BETA System home directory *)
       (# value: ^text;  ... exit value[] #);
     CurrentDirectory: (* The current working directory of this process *)
       (# value: ^text;  ... exit value[] #);
     ConvertFilePath:<
     (* Convert the 'path' relative to 'basis'.
      *
      * Within 'path' the conversion expands the abbreviations mentioned
      * above.  '..' is a relative path designating the parent of a directory
      * (if any) whereas '.' is a relative path designating the directory
      * itself.  If path starts with ~, this is expanded to the home
      * directory of the user, and if it starts with ~foo, this is expanded
      * to the home directory of the user foo. These expansions ignore basis.
      * These abbreviations are only expanded if they are the first
      * component(s) of 'path'.
      *
      * The only meaningful abbreviation in 'basis' is that it may start with
      * '.' Using this, 'basis' may be specified as the relative path '.'
      * instead of the absolute path THIS(PathHandler).CurrentDirectory.
      *
      * If both the path and the basis are relative paths, the result will be
      * a relative path, otherwise it will be an absolute path.  Path and
      * basis may be terminated by '/', but need not be.
      *
      * The resulting path will contian the native DirectoryChar as separators.
      *
      *)
       (#
          path,basis,convertedpath: ^text;
          NoFather:< Exception
          (* Raised if there are too many '..' in the beginning of 'path'
           * compared to the given (absolute) basis.
           * If basis is relative, excess '..'s will be converted to '..'
           *) (#  ... #);
          convert:
            @...
        enter (path[],basis[])
        do convert; INNER
        exit convertedpath[]
        #);
     DirectoryChar: @char
```

```
      (* The character which is used in this operating system to merge local
       * names to full path−names; '/' in unix, ':' on Macintosh, '\' on Windows
       *) ;
      LocalPath:
      (* Given a path and a basis, exit the local path relative to the basis,
       * if possible.
       *
       * The local path will be a path relative to basis, if basis is a prefix
       * of path. If this is not the case, it is attempted to use some of the
       * paths allready converted by THIS(PathHandler) as prefix to path. In
       * this case the result may be a relative path, or it may be a path in
       * the form ~foo.  If it is not possible to find any legal prefix, the
       * exception LocalPathException is raised and the original path is
       * exited. LocalPathException has the continue attribute set to true by
       * default; set it to false in a further binding, if you want to catch
       * this error.
       *
       * The resulting path will contain '/' as separators.
       *)
        (#
           path,basis: ^text;
           localName: ^text;
           LocalPathException:< Exception
             (#  ... #);
           localize:
             @...;

        enter (path[],basis[])
        do localize
        exit localName[]
        #);
      init:< (#  ... #);
      Private:
        @...;

  #);
FilenameConverter: PathHandler
  (#
  (* This pathhandler is capable of keeping track of files which have
   * allready been "converted".
   *
   * convertFilePath will return the same pathname every time the same
   * file is accessed even if the absolute path−names differs.
   *
   * E.g. if /users/beta/basiclib/current is a link to the directory
   * ~cn/beta/basiclib then
   *       '/users/beta/basiclib/current/file.bet' −> convertFilePath −> ...
   * and
   *       '~cn/beta/basiclib/file.bet' −> convertFilePath −> ...
   *
   * will give the same result, and the result will be the exit−parameter of
   * the first executed call of "convertFilePath".
   * Notice that if the file itself is a link to another file, these two
   * disk entries are considered different.
   * Notice also that in this special pathhandler, it is significant whether
   * the path are terminated by '/' or not: If it is, the path is considered
   * a directory.
   *)
     <<SLOT FileNameConverterLib:Attributes>>;
     convertFilePath::<
       (#
          fileNotFound:<
            (# do INNER #);
          convert:
            @...
       do convert
```

```
      #);
   init::<  (#  ... #);
   FCPrivate: @...
#)
```

14.9 RepetitionObject Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'sysutilsrepobj' '../lib';
(*
 * $RCSfile: RepetitionObject.bet,v $ $Revision: 1.9 $ $Date: 2000/01/11 15:32:35 $
 *
 * COPYRIGHT
 *       Copyright Mjolner Informatics, 1992-94
 *       All rights reserved.
 *)

BODY 'private/RepetitionObjectBody';

--- lib:attributes ---
memcpy: external
  (# adrTarget: @Integer;
     adrSource: @Integer;
     byteCount: @Integer;
     out: @Integer;
  enter (adrTarget, adrSource, byteCount)
  exit out
  #);

RepetitionObject:
  (* Used to pass serialized objects around.
   * Can also be used to pass simple datastructures through sockets.
   *)
  (# r: [InitialRange]@Integer;
     initialRange:< (# value: @Integer;
       do 10 -> value; INNER
       exit value
       #);

     <<SLOT RepetitionObjectAttributes:attributes>>;

     size: @ (# enter r[1] exit r[1] #);
     (* Index of last used entry in r. (Not the same as r.range *)

     pos: @(# enter r[2] exit r[2] #);
     (* next position to get or put *)

     get: @(# do r[2] + 1 -> r[2] exit r[r[2] - 1] #);

     getLong: @ (* Get a long, byteswapping it, if on little-endian. *)
       (# l: @Integer;
       ...
       exit l
       #);

     (* Like Get, except it does not update position *)
     Peek: @(# exit r[r[2]] #);

     PeekLong: @
       (# l: @Integer;
       ...
       exit l
       #);

     put: @
       (# enter r[r[2]]
       ...
       #);
```

```
    (* Put a long, byteswapping it on little−endian machines. *)
    putLong: @
      (# enter r[r[2]]
         ...
      #);

    (* The user of cheapPut must ensure that space is available
     * (using makeSpace) and that size is updated (using checkSize). *)
    cheapPut: @ (# enter r[r[2]] do r[2]+1 −> r[2] #);

    (* Put a long, byteswapping it on little−endian machines.
     * Comment on cheapPut applies, as they share the allocated space. *)
    cheapPutLong: @
      (# l: @Integer;
      enter l
      ...
      #);

    makeSpace: @
      (# nlongs: @Integer;
      enter nlongs
      ...
      #);

    checkSize: @(# ... #);

    getText: @
      (# t: ^Text; len, wsize: @Integer;
      ...
      exit t[]
      #);

    putText: @
      (# t: ^Text;
         len, wsize: @Integer;
      enter t[]
      ...
      #);

    firstPos: (# exit 3 #);
    initialSize: (# exit 2 #);
    reset: @(# ... #);

    init: @(# ... #);

    (* Safety checking procedures.
     * As the operations above do NO parameter checking at all,
     * the following functions may be used when the origin of
     * a repetitionObject is uncertain.
     * They are also quite usefull to catch programming errors.
     *)

    checkGet: BooleanValue
      (# ... #);

    checkGetText: BooleanValue
      (# maxLen: @Integer;
      enter maxLen
      ...
      #);
  #)
```

14.10 Scanobjects Interface

```
ORIGIN '~beta/basiclib/betaenv';
BODY 'private/scanobjectsbody';

(* Support for scanning all objects at a point of time
 * in the execution of any BETA program.
 *
 * If you specialize the 'callback' virtual, it will be
 * executed once for each object visited, with 'obj'
 * referring to the object being visited. It will
 * never be executed for a non-object, such as a
 * repetition etc.
 *
 * The pattern variable 'root' is used to select
 * which objects to visit: an object is visited iff
 * it is an instance of a specialization of the
 * pattern denoted by 'root'. If 'root' is NONE,
 * all objects and non-objects are visited. If
 * (root## = object##), all objects are visited.
 *
 * If 'printVisited' is true, a message is printed
 * describing the pattern of each visited object,
 * and giving the category of each visited non-object,
 * such as '[Value Repetition]'.
 *
 * When 'printVisited' is true, other booleans can be
 * used to enhance the output for objects: If
 * 'printOrigin' is true, the origin of each
 * visited object is printed. If 'printSize'
 * is true, the size in bytes of each visited
 * object is printed.
 *
 * If a garbage collection runs during a scan, it
 * will be aborted with an error message, but the
 * program continues otherwise unaffected. To
 * avoid this, try to create as few objects as
 * possible in callback. (If callback is not
 * specialized, it will not get called, and no
 * objects will be created during the scan.)
 *
 * COPYRIGHT
 *       Copyright Mjolner Informatics, 1994-96
 *       All rights reserved.
 *)

--- LIB: attributes ---

scanCallback:
  (# obj: ^object;
  enter obj[]
  do INNER
  #);

scanPrefix:
  (# callback:< scanCallback;
     root: ##object;
     printVisited,printOrigin,printSize: @boolean;
  do INNER
  #);

scanIOA:  scanPrefix(# do ... #);
scanAOA:  scanPrefix(# do ... #);
scanLVRA: scanPrefix(# do ... #);
```

```
ScanRefsToPrefix:
  (# target: ^Object;
     callback:< scanCallback;
     printName: @Boolean;
     skipGC: @Boolean; (* Set to true when using within the other scans *)
  enter target[]
  do INNER
  #);

(* Scans IOA only for references to the given target *)
scanRefsToObject: ScanRefsToPrefix(# do ... #)
```

14.11 Time Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'sysutilstime' '../lib';
BODY 'private/timebody';

(*
 * COPYRIGHT
 *       Copyright Mjolner Informatics, 1992-96
 *       All rights reserved.
 *)

-- LIB: attributes --

systemTime:
  (* Exits the current time in a system-dependent format *)
  (# t: @integer;
  ...
  exit t
  #);

preciseTime:
  (* Same as SystemTime, except that it also returns the millisecond resolution
   * of time - note that the accuracy of the millisecond resolution is
   * system-dependent
   *)
  (# time, millitm: @integer;
     gettime: @...
  do gettime;
  exit (time, millitm)
  #);

cpuTime:
  (* Exits the the amount of CPU time (in microseconds) used since the first
   * call to cpuTime (in the same program).  May not be supported on all
   * platforms.
   *)
  (# t: @integer;
  ...
  exit t
  #);

formatTime:
  (* Translates the entered system time into a text readable by humans *)
  (# t: @integer;
     readable: ^text;
  enter t
  ...
  exit readable[]
  #)
```

14.12 TimeHistory Interface

```
ORIGIN '~beta/basiclib/betaenv';
INCLUDE '~beta/sysutils/time';
---lib:attributes---
timeHistory:
  (* pattern to keep track of history of CPU times;
   * if called with TRUE, current time and used CPU
   * time is printed on the stream S (default screen[])
   *)
  (# t: [4] @ integer;
     top: @integer;
     S: ^stream;
     display: @boolean
  enter display
  do (if (top+1 -> top) > t.range then t.range -> t.extend if);
     cpuTime -> t[top];
     (if display then
         (if S[] = NONE then screen[] -> S[] if);
         systemtime -> formattime -> S.putline;
         (if top > 1 then
             'Time this slot: \t' -> S.puttext;
             ((t[top] - t[top-1]) div 1000000) -> S.putint;
             ' sec. \nAccumulated time: \t' -> S.puttext;
             ((t[top] - t[1]) div 1000000) -> S.putint;
             ' sec.\n' -> S.puttext
     if)if)
  #)
```

Index

The entries in the alphabetic index consists of selected words and symbols from the body files of this manual − these are in **bold** font − as well as the identifiers defined in the public interfaces of the libraries − set in regular font.
In the manual, the entries, which can be found in the index are typeset like this. This can help localizing the identifier, when the link from the index if followed − especially in the case where the browser does not scroll the line to the top, e.g. because there is less than a page of text left.
In the small table of letters and symbols below, each entry links directly to the section of the index containing entries starting with the corresponding letter or symbol.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W