Mjølner Informatics Report
MIA 90−09
March 2004

Table of Contents

1 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.

1.1 Attributes

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

1.2 Using the unixFile Fragment

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

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

1.2.1 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:

Program 1: unixentry.bet

```
ORIGIN '~beta/unixlib/unixfile';
INCLUDE '~beta/sysutils/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;*)
       ' We skipped that, since it messes up later tests, if we don\'t own the file!' ->
    // false then
       ' does not exist.' -> putline;
   if);
#)
```

1.2.2 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.:

Program 2: splitexe.bet

```
ORIGIN '~beta/unixlib/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 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;
#)
```

2 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.

2.1 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.

2.2 Using the unixDirectory Fragment

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

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

2.2.1 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:

Program 3: listunix.bet

```
ORIGIN '~beta/unixlib/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;
#)
```

3 The PassWdHome Library

⚠**Notice**: The PassWdHome library can only be used on unix−systems.

The PassWdHome library defines only one pattern, getPassWdHome that expand ~username to a absolute path.

3.1 Using the PassWdHome Fragment

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

```
INCLUDE '~beta/unixlib/passwdhome'
--- program: descriptor ---
(# ...
do ...
   '~bart' -> getpasswdhome -> putline;
   ...
#)
```

3.1.1 GetPassWdHome example

```
ORIGIN '~beta/unixlib/passwdhome';
-- PROGRAM: descriptor --
(# t: ^text
do
   'Enter text strings for home-directory expansion.' -> putline;
   'E.g. '\'~beta\'' will be expanded to the home directory' -> putline;
   'of the user called beta.' -> putline;
   'Stop expansion by entering an empty string' -> putline;
   loop: cycle
     (#
     do 'Homedir to get: ' -> putText;
        keyboard.getLine -> t[];
        (if t.length // 0 then leave loop
         else t[] -> getPasswdHome -> putLine;
        if);
     #);
#)
```

4.1 Unixfile Interface

```
ORIGIN '~beta/basiclib/file';
LIB_DEF 'unixfile' '../lib';
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
        *)
       (# ... #);
  #)
```

4.2 Unixdirectory Interface

```
ORIGIN '~beta/basiclib/directory';
LIB_DEF 'unixdirectory' '../lib';
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(##)
```

4.3 Passwdhome Interface

```
ORIGIN '~beta/basiclib/betaenv';
LIB_DEF 'passwdhome' '../lib';
BODY 'private/passwdhomebody'
(*
 * COPYRIGHT
 *       Copyright Mjolner Informatics, 1992-94
 *       All rights reserved.
 *)
--- LIB: attributes ---

getPassWdHome:
  (* Expands usernames like ~cn to real home directories like  /users/cn *)
  (# t: ^text;
  enter t[]
  ...
  exit t[]
  #)
```

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

O

P

R

S

U

W