Identification

Command, file system interface
ufo
R. J. Feiertag

Purpose

This routine has three main objectives:

1. Provide a set of file system primitives to accept both
    fixed and varying character strings,

2. To handle the convention of using a trailing ">" on a
    name to indicate a directory branch,

3. To test the usefulness of proposed file system primitives
    by simulating them in ufo.

Usage

The reader should be familiar with the new command system
(BX.8.00). Each of the entry points acts as a file system
primitive. They differ from the ring 0 primitives in
that they accept both fixed and varying character strings
as arguments. A name with a trailing ">" is treated as
a directory branch name.

The entries are:

    1. chname-change an entry name

        call ufo$chname(dir, entry, oldname, newname, code);

    dcl dir char(*),   /*or char(*) varying,
                                           symbolic pathname of
                                           directory*/

        entry char(*),   /*or char(*) varying,
                                           symbolic name of
                                           entry*/

        oldname char(*),   /*or char(*) varying,
                                           name to be deleted from
                                           list of entry names*/

```
        newname char(*),              /*or char(*) varying,
                                        name to be added to list
                                        of entry names*/

        code fixed bin(17);           /*if non-zero, it repre-
                                        sents the code of an error
                                        detected by the file
                                        system*/
```

oldname is deleted from and newname is added to the list
of entry names of the branch indicated by dir and entry.
If oldname is "" nothing is deleted and if newname is
"" nothing is added. The write permission in dir is needed
to change an entry name.

chname first calls the internal procedure off which removes
trailing blanks and a trailing ">" if they exist and converts
to fixed character strings by calls to cv_string (BY.10.03).
chname then calls hcs_$chname (BG.8.02) to perform the
name changes.

   2. appendb-creates a branch

   call ufo$appendb(dir, name, usermode, maxl, code);

```
 dcl name char(*),               /*or char(*) var, name
                                    for new branch*/

     usermode bit(5),            /*access mode of current
                                    user with respect to this
                                    branch*/

     maxl bit(9);                /*maximum length of segment
                                    to which this branch points
                                    (in units of 1024 words)*/
```

The user needs the append permission in dir to create
a branch in dir. If so, a branch with entry name name
is created in dir.

appendb first calls off and then calls hcs_$appendb (BG.8.02).

   3. appendl-create a link

   call ufo$appendl(dir, name, pathname, code);

```
dcl name char(*),                    /*or char(*) var, name
                                        for new link*/

    pathname char(*);                /*or char(*) var, path-
                                        name of the entry to
                                        which this new link
                                        will point*/
```

The user needs append permission in <u>dir</u> to create a link
in it. If so, a link with entry name <u>name</u> will be created
which points to the entry <u>pathname</u>.

append1 first calls <u>off</u> and then calls hcs_$append1 (BG.8.02).

4. <u>chase</u>-determine the entry to which a link effectively
          points and distinguish between directory
          branches and non-directory branches.

```
    call  ufo$chase(dir, entry, newdir, newentry, nlinks, code);

dcl newdir char(511) var,            /*directory containing
                                        entry being chased*/

    newentry char(*),                /*or char(*) var, name
                                        of entry being chased*/

    nlinks fixed bin(17);            /*maximum number of
                                        links to be gone through*/
```

Read permission is necessary in each directory containing
a branch or link to be chased. <u>newdir</u> and <u>newentry</u> specify
a branch effectively pointed to by <u>dir</u> and <u>entry</u>. If
<u>newentry</u> is a directory branch it will contain a trailing
">". If the number of links gone through in search of
<u>newdir</u> and <u>newentry</u> exceeds <u>nlinks</u> an error will be returned.
Whenever an error occurs the name of the link currently
being processed will be returned as the value of <u>newdir</u>
and <u>newentry</u>. If <u>nlinks</u> is 0 the maximum number of links
will be set to the system maximum (currently 10).

chase calls the internal entry chase_ with the same calling
sequence except that if nlinks is 0 it is changed to the
system maximum number of links. chase_ calls hcs_$status
(BG.8.02). If the entry is a non-directory branch then
its name is returned. If the entry is a directory branch
then its name is returned with a trailing ">". If entry
is a link and <u>nlinks</u> is greater than 0 then chase_ is
called recursively with the pathname of the link, and
<u>nlinks</u> reduced by one.

5. delentry-delete an entry

    call ufo$delentry (dir, entry, csw, code);

    dcl csw fixed bin(1);   /*courtesy switch*/

Write permission is necessary in the directory of the
entry to be deleted and if the entry is a branch, write
permission is needed in the branch. The entry specified
by dir and entry is deleted. If the entry is a non-directory
branch the segment is deleted. If the entry is a directory
branch the directory and its subtree are deleted. If
the entire subtree cannot be deleted then as much is deleted
as is possible and an error code is returned. If csw
is 1 then the branch will be deleted only if it is not
in use.

delentry first calls off. If entry contains a trailing
">" then hcs_$del_dir_tree (BG.9.06) is called to delete
the subtree. Then hcs_$delentry (BG.8.02) is called.

6. copier-create a copy of a segment

    call ufo$copier(dir, entry, newdir, newentry, code);

A copy of the segment in the branch defined by dir and
entry is created in the directory newdir, with name newentry.
Read permission is necessary in dir and entry and write
and append permission are necessary in newdir. The segment
defined by dir and entry can not be a directory.

First off is called to prepare the arguments for the file
system. Then initiate is called to get a pointer to the
segment to be copied. A new segment is created by a call
to make seg and a copy of the old segment is placed in
the new segment by a call to move.

7. movebr-move a branch

    call ufo$movebr(dir, entry, newdir, newentry, csw, code);

The branch indicated by dir and entry is moved to the
directory, newdir, and is given the name newentry. This
branch can not be a directory. The old branch indicated
by dir and entry will no longer exist. The access control
list is also moved. Read permission is necessary in dir
and entry and write permission is necessary in newdir.
If csw is 1 and entry is being used then nothing is done
and an error code is returned.

First ufo$copy is called to create a copy of <u>entry</u> in
<u>newentry</u>. ufo$readacl and ufo$writeacl are called to
move the ACL. Then ufo$delentry is called to delete <u>entry</u>.

8. <u>readacl</u>-get access control list

call ufo$readacl(dir, entry, user_area, aclptr, aclct, code);

dcl user_area area((*)),   /*an area provided by
                                           the caller in which readacl
                                           returns the acl informa-
                                           tion*/

aclptr ptr,   /*pointer to a structure
                                           allocated by Directory
                                           Supervisor in user_area
                                           which is filled with
                                           the contents of the
                                           requested acl*/

aclct fixed bin(17);   /*count of the number of
                                           user names in the acl,
                                           returned by Directory
                                           Supervisor*/

The access control list of the entry effectively pointed
to is returned. If <u>entry</u> is the null string the common
access control list of the specified directory, <u>dir</u>, is
returned. Read permission is necessary in the directory
of the entry. The structure returned by readacl is:

dcl 1 acl (aclct) based (aclptr),

     2 userid,

        3 name char(24),

        3 project char(24),

        3 instance_tag char(2),

     2 packbits

        3 mode bit(5),

        3 pad13 bit(13),

        3 (rb1, rb2, rb3) bit(6),

        3 traprp bit(18),

        3 pad18 bit(18);

```
dcl  1 trappoc based (tp),

    2 size fixed bin(17),

    2 string char (tp->trapproc.size);
```

After calling off, ufo$readacl calls the primitive readacl
(BG.8.02).

9. writeacl-write access control list

    call ufo$writeacl(dir, entry, aclptr, aclct, code);

The ACL of the entry effectively pointed to or CACL or
the specified directory dir, is replaced with the structure
pointed to by aclptr. Write permission is needed in the
directory of the entry pointed to. The structure of the
ACL or CACL is the same as that shown for readacl.

After calling off, ufo$writeacl calls the primitive writeacl
(BG.8.02).

10. status type-determine if entry is a non-directory
                branch, directory branch, or link.

    call ufo$status_type(dir, entry, chase, type, code);

```
dcl chase fixed bin(1),          /*switch to determine if
                                    links are to be chased*/

    type fixed bin(2);           /*indicates type of entry,
                                    as returned by status_
                                    type*/
```

type is set to 0 if entry is a link, 1 if entry is a non-directory
branch, 2 if entry is a directory and 3 if there is an
error. If chase is non-zero then type will return values
for the branch effectively pointed to by entry. Read
permission is needed in the directory of the entry.

First off is called to prepare arguments for the file
system, then a call is made to entry_status$type (BY.2.10)
to determine the type of the entry.