TO: MSPM Distribution
FROM: M. Turnquist
SUBJ: BG.8.02
DATE: 02/07/68

The attached revision of BG.8.02 reflects various changes which have been made to several of the general user primitives that come under the directory supervisor.

status and list_dir: the notion of the "protection list" has been radically altered and the "gate list" removed from the branch, leaving only the "ring brackets" (access bracket and call bracket) in the branch; therefore, the declaration of the arrays allocated in the caller's area has been changed slightly.

readacl and writeacl: the order of implementation has been modified. Also, the structure which is returned to the user from readacl, and input by the user to writeacl, is given.

appendb: Appendb has been reorganized. Appendb itself now just sets up the defaults for current length, bit count, ring brackets and usercode. Then appendbx is called.

Appendbx does all that appendb used to do, but has an extended argument list to include the arguments listed above. The code has been expanded to handle these arguments.

This change only affects those users who wish to specify current length, bit count, ring brackets or usercode other than the defaults. Most users will only reference appendbx through appendb.

Published: 02/07/68
(Supersedes: BG.8.02, 07/14/67,
BG.8.02, 03/01/67,
BG.8.02, 05/24/66)

Identification

Directory Supervisor, General User Primitives
C. A. Cushing, M. C. Turnquist

Purpose

The directory supervisor provides the primitives for manipulating directory entries and decides the permission needed to carry out the requested operation from the intent of the caller.

Primitive

The primitives of the directory supervisor which are callable by the general user have one of four intents (read, execute, write, append) with respect to the directory containing the given entry and possibly to the entry itself. The following is a list of the primitives with their normal intent, i.e., the mode needed by the process on whose behalf the primitive is being invoked.

1. list_dir (read)
2. status (read or execute)
3. chname (write)
4. delentry (write)
5. readacl (read)
6. writeacl (write)
7. set$bc (execute; write in branch)
8. set$consistsw (write)
9. a. set$copysw (write)
   b. set$relatesw (write)
10. set$rd (write; write in branch)
11. appendb, appendbx (append)
12. appendl (append)
13. setml (write)
14. movefile (execute; read and write in old branch, append in new branch)

1. list dir

The primitive list dir itemizes the contents of each branch and link in a directory.

```
call list_dir (dir, user_area, branchp, branchct, linkp, linkct,
            code);

     dcl dir char(*),         /* symbolic path name of the directory
                                 to be listed */

     user_area area((*)),     /* an area of storage provided by the
                                 caller into which the information from
                                 each branch and link in dir is put
                                 by directory supervisor */

     (branchp,linkp)ptr,      /* pointer to an array in the area
                                 containing selected information from
                                 each branch (link) in dir, returned
                                 by directory supervisor */

     (branchct,linkct) fixed bin(17), /* number of branches
                                 (links) in dir, i.e., size of the
                                 array pointed to by branchp (linkp),
                                 returned by directory supervisor */

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

The functions of directory supervisor for the primitive list_dir are to decide the permission needed (read) by the user in dir and if the user has this permission, to go through the branch and link slot tables to get at each branch and link, and to call the directory maintainer primitive packer to store selected information from each entry into an array allocated by list_dir in the area, user_area. After all branches and links are listed control is returned to the caller.

2. status

The primitive status itemizes the contents of one specifically named entry in a given directory.

```
call status (dir, entry, chase, type, user_area, entryp, code);

     dcl dir char(*),         /* symbolic path name of the
                                 directory */

     entry char(*),           /* symbolic name of the entry */
```

```
type fixed bin(2),     /* a two-bit flag indicating whether
                       entry is a directory (2) or non-
                       directory (1) branch or a link
                       (0), returned by directory supervisor */

chase fixed bin(1),    /* a switch when =1 indicates that
                       the status of the branch effectively
                       pointed to by entry is desired */

user_area area((*)),   /* an area of storage given by
                       caller in which contents of entry
                       are put by directory supervisor */

entryp ptr,            /* pointer to an array in the area
                       containing selected information
                       from entry, returned by directory
                       supervisor */

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

The caller needs the read permission if entry contains the slot number of the entry to be itemized else the read or execute permisssion if entry contains the name of the entry to be itemized. Directory supervisor calls the primitive findentry in directory maintainer to find entry in directory dir and calls packer to store selected information from entry into an array which was allocated by status in the area, user_area. If this area is not big enough, the call to packer is omitted. In this case, the caller merely ascertains whether entry exists or not and what type it is. Directory supervisor then unlocks entry (it had been locked by findentry) and returns control to the caller.

The following is the declaration of the arrays allocated in the caller's area by list_dir and status:

```
/* array of branches */

dcl  1 branches (branchct) ctl (branchp),
     2 padl bit (2), /* padding to prevent straddling word
                    boundary */
     2 uid bit (70/*uidsize*/), /* unique id of branch */
     2 (dtu, dtm, dtd, dtbm, rd) bit (72),
     2 dirsw bit (1), /* if =1, branch is a directory branch */
     2 optsw bit (2), /* value of copy and relate switches */
     2 bc bit (24), /* count of no. of bits in seg */
     2 consistsw bit (2), /* value of consistency variable */
     2 mode bit (5), /* value of TREWA for current user */
```

```
     2 usage bit (2), /* current usage of seq:read, write, data-
                     share, unused */
     2 usagect bit (17), /* count of the current no. of users
                     of the seg */
     2 nomore bit (1), /* value of no-more-users switch */
     2 cl bit (9), /* current length of segment in 1024 blocks */
     2 ml bit (9), /* max length of segment in 1024 blocks */
     2 acct bit (36), /* account to which storage for seg is
                     charged */
     2 (hlim, llim) bit (17), /* hi and lo multi-level limits */
     2 pad2 bit (2),
     2 (rb1, rb2, rb3) bit (6), /* ring brackets */
     2 pad5 bit (18),
     2 pad3 bit (18),
     2 namerp bit (18), /* rel ptr to names */
     2 pad4 bit (19),
     2 nnames bit (17), /* number of names for this branch */

/* array of links */

dcl  1 links (linkct) ctl (linkp),
     2 pad bit (1), /* padding to prevent straddling word
                      boundary */
     2 uid bit (70),
     2 (dtu, dtm, dtd) bit (72),
     2 pathnamerp bit (18),/* rel ptr to path name*/
     2 namerp bit (18),/* rel ptr to array of link names*/
     2 nnames bit (17); /* number of names */

/* array of names for each branch and link - array of names for
the gates of each branch (if any) */

dcl  1 namelist (nnames) ctl (nlistptr),
     2 size bit (17),
     2 string char (511);

/* path name of the entry to which each link points */

dcl  1 pathname ctl (pathnameptr),
     2 size bit (17),
     2 string char (pathnameptr->pathname.size);
```

For a more detailed explanation of each piece of the above structures see BG.7.00, Directory Data Base.

3. chname

The primitive chname modifies the names of an entry in a directory by adding one name to and deleting another from the list of names of the entry.

```
call chname (dir, entry, oldname, newname, code);

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

     newname char(*),     /* name to be added to the list
                             of names of entry */

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

It is possible to only delete or to only add a name if the newname or oldname argument is a zero-length character string.

The user needs the write permission in dir to modify the names of entry. The directory supervisor calls the findentry primitive to find entry in dir and then checks the list of names in entry to be sure oldname is in the list, newname isn't and, in the case where only oldname is to be deleted, to be sure that at least one name will be left in the list after oldname is deleted. The hash$out primitive of directory maintainer is invoked to vacate the location in the hash table used for oldname and the hash$in primitive to fill in an empty location in the hash table for newname with the pointer to entry. If hash$in is unsuccessful, e.g., newname is a name for another entry in dir and therefore an empty location cannot be found for it, then entry is unlocked and an error is reflected to the caller. Otherwise, oldname is deleted from and newname is added to the list of names of entry.

If entry is a link, the date-time-modified item is updated to the current date and time. If entry is a branch the date-and-time-branch-modified item is updated to the current date and time. This is to indicate to the backup system that the branch has been modified, not the segment. In any case, segment control is notified of the modification to the contents of dir through the primitive dirmod (see BG.3.00), the entry is unlocked and control is returned to the caller.

4. delentry

The primitive delentry deletes a specified entry from a given directory. If the entry is a branch, the contents of the segment to which it points are deleted first.

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

```
     dcl csw fixed bin(1);/* courtesy switch indicating whether
                          or not the caller wishes to delete
                          a segment while someone else is
                          using it
                          if = 1, give an error return to
                          caller if segment in use
                          if = 0, delete segment even if it
                          is in use */
```

The user needs the write permission in dir to delete entry. If the entry is a branch, the user needs the write permission in the branch also. First, the findentry primitive is invoked to find entry in dir. If entry is a link, the removel primitive in directory maintainer is called to remove all traces of entry from dir (e.g., vacate locations in hash table for its names, decrease the count of the number of links in dir by one) and control is then returned to the caller. If entry is a branch and if the current length of the segment to which it points is non-zero, then the contents of the segment are deleted through a set of calls to segment control; makeknown, deleteseg, makeunknown.

If csw = 1 and the segment is in use or if entry is a directory branch and the directory segment to which it points has entries in it, then the segment is not deleted, entry is unlocked and an error is reflected to the caller. If the current length of the segment is zero, or if the segment was successfully deleted, the primitive removeb of directory maintainer is called to remove all traces of entry from dir and control is returned to the caller.

5. readacl

The primitive readacl returns the Access Control List (ACL) of a specified entry on the Common Access Control List (CACL) of a specified directory. The calling sequence is as follows:

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

```
dcl dir char (*),           /* directory path name */

entry char (*),             /* entry name. If this argument
                               is a zero-length character
                               string the CACL of dir
                               is returned */

user_area area ((*)),       /* an area provided by the caller
                               in which readacl returns the
                               acl information */

aclptr ptr,                 /* pointer to a structure
                               allocated by Directory Super-
                               visor in user_area which is
                               filled in with the contents
                               of the requested access
                               control list */

aclct fixed bin (17),       /* count of the number of user
                               names in the access control
                               list, returned by the
                               Directory Supervisor */
```

The structure to which aclptr points is set up in the following manner:

```
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 trapproc based (tp),
    2 size fixed bin (17),
    2 string char (tp->trapproc.size);
```

Note that the structure output from readacl is identical to the structure input into writeacl.

The normal procedure (and that used by the command setacl) for modifying an ACL or CACL is as follows. Use readacl to get the current ACL or CACL from the branch or directory. Modify it in the array form given as output from readacl. Input the modified array into writeacl. Writeacl reformats the structure into a threaded list and enters the revised ACL or CACL into the branch or directory.

Read permission is needed in the directory containing the requested access control list in order to read it. If entry is a link, the execute permission is needed in dir and in each directory containing the links in the path which goes from entry to the branch.

Implementation

If the entry argument is specified (i.e. entry is a non-zero length character string) then the findbranch primitive of Directory Maintainer is called to find the branch effectively pointed to by the entry. The ACL is copied into the stack_area and the branch is unlocked. (The branch was locked by findbranch.) The ACL is then copied from the stack_area into the area provided by the user. The double copying is to assure that this primitive will not incur an access violation while it has the directory or a branch locked in the case where the area provided by user cannot be accessed. Control is returned to the caller.

If the entry is a zero-length character string then the directory is found by getdirseg. The directory is locked for reading and the CACL pointer is found. The CACL is locked and the directory unlocked. The CACL is written into the stack area in order to avoid access problems. The CACL is then unlocked. The CACL structure is read from the stack area into the user area and control is returned to the caller.

6. writeacl

The primitive writeacl replaces the ACL of the specified entry or the CACL of the specified directory. The calling sequence is as follows:

```
call writeacl (dir, entry, aclptr, aclct, code);
```

where the arguments are declared the same as in readacl except that now aclptr and aclct are input arguments rather than return arguments.

The structure to which aclptr points is the same as in readacl.

Write permission is needed in the directory containing the requested access control list in order to replace it. If entry is a link, the execute permission is needed in dir and in each directory containing the links in the path which goes from entry to the branch. The ring brackets must conform to the following conditions: $rb1 \leq rb2 \leq rb3$ and rb1 must be greater than or equal to the current validation ring number. If ring number or the ring brackets supplied do not meet these conditions then the write operation is not performed and a bad_ring_brackets code is returned.

Implementation

The new ACL or CACL is written into the current stack area. This is to assure access before locking the directory.

If the entry argument is specified, i.e., a non-zero length character string, then the findbranch primitive of Directory Maintainer is called to find the branch effectively pointed to by the entry. (Bear in mind that findbranch returns the branch locked.) The directory is locked for modification. Write the new ACL from the stack into the directory. Free the old ACL. Unlock both the directory and the branch. Tell the directory that the branch has been updated by calling segment control at branchmod. Control is returned to the caller.

If the entry argument is a zero-length character string then dir is located through a call to segment control at getdirseg. Lock the directory for modification. Find the CACL pointer and lock the CACL. Read the new CACL from the stack_area into the directory. Free the old CACL. Unlock both the directory and the CACL and return control to the caller.

7. set$bc

The primitive set$bc is provided for the use of the file system interface module (FSIM) for replacint the bit-count item in the branch to which a given entry effectively points.

call set$bc (dir, entry, bitct, code);

```
     dcl bitct bit(24),    /* count of the number of bits
                           in the segment to which entry
                           points, given by caller */
```

The execute permission is necessary in the directory containing the branch to which entry points. The primitive findbranch is called to locate the desired branch. The effmode primitive in the access control module is then called to find the effective mode of the user with respect to the branch.

If the mode does not indicate the write or append permission for this user, then the branch is unlocked and this error is reflected to the caller. Otherwise the bit-count item is replaced, the current date and time is entered into the date-and-time-branch-modified item and segment control is notified of the modification to the directory containing the branch through the primitive dirmod. Then the branch is unlocked and control is returned to the caller.

8. set$constw

The primitive set$constsw changes the value of the consistency variable in a branch to a given value. The setting of the consistency variable specifies to the backup system that the user does or doesn't wish the subtree beneath this branch to be dumped consistently, (see BH.2.00). The setting of the variable tells the user that either 1) the subtree is consistent, i.e., the subtree was successfully dumped or reloaded in a consistent state or no consistence dump was requested, 2) the subtree is consistent but waiting to be dumped in a consistent state, or, 3) the subtree is inconsistent, i.e., dump aborted while in subtree or entire subtree was not reloaded.

```
call set$constsw (dir, entry, const, code);

     dcl const bit(2);     /* new value for the consistency
                              variable, given by caller (see BH.2.00
                              for meaning of various values for
                              this variable) */
```

The user needs the write permission in the directory containing the branch pointed to by entry.

The primitive findbranch is called to find the branch pointed to by entry.

The value of the consistency variable in this branch is changed to const. The date-and-time-branch-modified item is not updated in this case.

The branch is then unlocked and control is returned to caller.

9. set$copysw

set$relatesw

The primitives set$copysw and set$relatesw change the setting of the copy and relate switches respectively, in the branch to which a given entry effectively points. These consist of the copy and relate switches which are interpreted by the Segment Management Module. If the copy switch is ON, each user of the segment will get his own copy. If the relate switch is ON, the segment contains information about a set of segments which are dependent upon one another.

```
call set$copysw (dir, entry, copy, code);

call set$relatesw (dir, entry, relate, code);

     dcl copy bit(1),       /* new setting of copy switch */

     relate bit(1);         /* new setting of relate switch */
```

The write permission is needed in the directory containing the branch to which entry points. The primitive findbranch is called to find the desired branch. The value of the option switches in the branch is changed as specified and the date-and-time-branch-modified item is updated to the current date and time. Segmemt control is notified of the modification to the contents of the directory containing the branch through the primitive dirmod, the branch is unlocked and control is returned to the caller.

10. set$rd

The primitive set$rd changes the setting of the retention date in the branch to which a given entry effectively points.

```
call set$rd (dir, entry, rdate, code);

     dcl rdate bit(72);     /* new date and time after which the
                               branch to which entry points and its
                               segment are to be deleted, given by
                               caller */
```

The write permission is needed in the directory containing the branch pointed to by entry.

The primitive findbranch is called to locate the requested branch. The primitive effmode in the access control module is then called to determine the effective mode of the user with respect to this branch. If the effective mode does not indicate write permission or if rdate is greater than the default time lapse added to the current date, the branch is unlocked and the error is reflected to the caller. Otherwise the value of the retention date in the branch is changed to rdate.

The date-and-time-branch-modified item is updated to the current date and time. Segment control is notified through the primitive dirmod that the directory containing the branch has been modified, the branch is unlocked and control is returned to the caller.

11. appendbx and appendb

The primitive appendbx creates a new branch in the file system hierarchy by appending it to a given directory.

Appendbx may be called directly if there is need, however, it will usually be accessed through appendb.

```
call appendbx (dirname, name, dirsw, usermode, ringbrack,
               usercode, optionsw, maxl, curl, bitcnt,
               code);

dcl dirname char (*),       /* path name of the directory in
                               which to append the new branch */

name char (*),              /* name for the new branch */

dirsw bit (1),              /* switch indicating whether the
                               branch is to be a directory
                               (1) or a non-directory (0)
                               branch */

usermode bit (5),           /* access mode of creator
                               (current user) with respect
                               to this new branch trap,
                               read, execute, write, append */

ringbrack (3) bit (6),      /* the three ringbrackets */

usercode char (50),         /* the user's code number made up
                               of his name, project, and
                               instance_tag */

optionsw bit (2),           /* setting of the copy and relate
                               switches for this new branch
                               (if the branch is a directory
                               branch, these switches must be
                               off) */

maxl bit (9),               /* maximum length of the segment
                               to which this branch will
                               point (in units of 1024 words) */
```

```
curl bit (13),              /* current length of the segment
                               to which the branch will
                               point (in units of 64 words) */

bitcnt bit (24);            /* the number of bits in the
                               new segment */
```

All arguments are given by the caller.

The primitive appendb sets up the defaults for current length, bit count, the three ringbrackets and usercode for appendbx.

```
call appendb (dir, name, dirsw, usermode, optionsw,
              maxl, code);
```

The defaults provided by appendb are as follows:

```
curl = "0"b;

bitcnt = "0"b

usercode = current user's name, project and instance_tag.

for i = 1 to 3
     ringbrack (i) = validation level ring number
```

Appendbx is then called.

The user needs the append permission in dir to create a branch in dir. The segment dir is located through a call to getdirseg in segment control.

The first major task for appendbx to find a slot number for this new branch. If there is a vacant branch in dir, it is locked and its slot number and structure are used for this new branch. Its contents are changed as listed below. Otherwise, a new branch structure is allocated in dir and a new slot number (equal to plus the current largest branch slot number) is given to this new branch.

It is locked and its contents are set as listed below. The hash$in primitive is then called to put the slot number of this new branch into the hash table location found for name.

If this new branch is a directory branch, the primitive makeknown in segment control is called to make this new directory segment known. Then the segment can be initialized (e.g., count of total number of links in the directory is set to zero). The primitive makeunknown is then called for this segment.

The new branch is then unlocked. Segment control is notified of the modification to dir through dirmod and control is returned to the caller.

The following values are given to the new branch items:

uid =	value of the function unique_bits (BY.15.01)
dirsw =	dirsw
dtbm =	current date and time
dtd, dtu =	0
dtm =	0 (if dirsw = non-directory) else current date and time
usage, usagect, nomore =	0
ml =	maxl
bc =	bitcnt
optsw =	optsw (if dirsw = non-directory) else 0
rd =	default time lapse + current date and time
cl =	curl
hlim, llim =	default setting
acct =	current user's account number
names =	name

access control list = { userid - usercode; mode = usermode; rb1 = ringbrack (1), rb2 = ringbrack (2), rb3 = ringbrack (3); trap procedure = empty }

access control list =	userid - usercode
	mode = usermode
	rb1 = ringbrack (1)
	rb2 = ringbrack (2)
	rb3 = ringbrack (3)
	trap procedure = empty

12. appendl

The primitive appendl creates a new link in a given directory.

```
call appendl (dir, name, pathname, code);

     dcl pathname char(*);/* pathname of the entry to which
                          this new link will point, given by
                          caller */
```

The user needs the append permission in dir to create a link in it. The segment dir is located through a call to getdirseg in segment control.

If there is a vacant link in dir, it is locked and its slot number and structure are used for this new link. Its contents are changed as listed below. Otherwise, a new link structure is allocated in dir and a new link slot number is assigned to this link. It is locked and its contents are set as listed below. The hash$in primitive is called to put the slot number of this new link in the hash table location found for name.

The new link is then unlocked. Segment control is notified of the modification to dir through dirmod and control is returned to the caller.

The following values are given to the new link items:

uid =	value of the function unique_bits
dtu, dtm =	current date and time
dtd =	0
names =	name
pathname =	pathname

13. setml

The primitive setml changes the maximum length of the segment to which a given entry effectively points.

```
call setml (dir, entry, maxl, code);

     maxl bit(9);    /* new maximum length of the segment to
                        stored in the branch to which entry
                        effectively points, given by the caller */
```

The write permission is needed in the directory containing the branch to which entry effectively points. The primitive findbranch finds the desired branch. If maxl is less than the current length of the file then the branch is unlocked and an error is reflected to the caller. Otherwise maxl is stored in the maximum length item in the branch. If the segment is active, the segment control primitive unloadseg is called to unload the segment if loaded, i.e., to place directed faults in all descriptors currently pointing to the segment. Then, when one of these faults occurs, the AST entry will be updated or constructed for this segment and will contain the new maximum length setting.

To indicate to the backup system that the branch has been modified but not the file, the date-and-time-branch-modified item is updated to the current date and time. Segment control is notified of the modification to the directory containing the branch through the primitive dirmod, the branch is unlocked and control is returned to the caller.

14. movefile

The primitive movefile effectively moves a segment from one section of the hierarchy to another. An existing branch is modified to point to the segment and the branch which originally pointed to the segment is modified to point to no segment.

```
call movefile (dir, entry, csw, newdir, newentry, code);

     dcl dir char(*),      /* path name of a directory */

     entry char(*),        /* name of an entry in dir which
                              effectively points to the segment
                              to be moved */

     csw fixed bin(1),     /* as described for delentry */
```

newdir char(*),	/* path name of a directory */
newentry char(*);	/* name of an entry in newdir which will effectively point to the segment */

All arguments are given by caller.

The user needs the write permission in the directory containing the branch to which <u>entry</u> points and in the directory containing the branch to which <u>newentry</u> points. The primitive effmode in access control is then called to determine if this user also has the write permission with respect to the segment to which entry effectively points. If not, the branch is unlocked and an error is reflected to the caller.

If the user has the write permission in dir the following conditions are also necessary for the segment to be moved:

1. if the segment is a directory segment, it must have no entries in it,
2. csw = 0 or the segment is not in use (if the segment is in use and csw = 1 then the branch is unlocked and the caller is returned an error), and
3. if the segment is active, it must be deactivated through a call to unloadseg in segment control since the AST entry for it contains invalid information such as the slot number of the branch pointed to by <u>entry</u>.

The primitive findbranch is called again to find the branch pointed to by newentry. If newentry points to an inactive zero-length segment then the items in the branches pointed to by newentry and entry are changed as follows:

<u>entry</u>

did, file_map =	0
retrievesw, retrieve trap =	0
dtbm, dtm, dtu =	current date and time

<u>newentry</u>

dtu, dtm, dtbm =	current date and time
did, file_map, ml, cl, bc, dirsw =	corresponding items in branch pointed to by entry

actind, actime = 0

retrievesw, retrieve trap = 0

Both branches are unlocked and segment control is notified, through calls to dirmod, of the modifications to the directory containing the branch to which entry points and the directory containing the branch to which newentry points. Control is then returned to the caller.