Open a Directory (DR:OPDR)

Input parameter:
IP1 C: Directory (OB.OPEN)

No return parameters.

Directories are implemented as 1-level disk files, and so the directory system must open a directory-file before manipulating it. If several consecutive actions are to be performed on a directory, it can be held open to reduce the overall real time, by using this action. Note that the directory system always calls the disk system to open a directory file, so it is never mandatory for a user to open a directory.

An open 1-level disk file requires fixed ecs space, so a directory should never be held open longer than necessary. The disk system "close all locally open files" action (DF:CHOF) affects directories, since they are disk files.

Possible errors while opening a directory:

Same as opening a disk file (quad vide).

Close a Directory (DR:CLDR)

Input parameter:

IPI C: Directory (OB.CLOSE)

No return parameters.

Closing a directory performs a close action on the underlying disk file used to implement the directory, and fixed ecs space occupied by this file is released. Of course, the directory may still be manipulated since all the directory actions automatically open the directory for the duration of the action.

Possible errors while closing a directory:

Same as for closing a disk file (q.v.).

Return Directory Subsystem Clocks (DR:CLKS)

No input parameters

Returned parameters:

RDAT 0: system time
1: swap time
2: directory system time

The cumulative time expended by the directory system (including time spent by the disk system in response to calls by the directory system) is returned to the caller.

Access an Object from a Directory (DR:UACC)

Input Parameters:

IP1 C: A directory (OB.ACC, and OB.IMPL if IP3 is the implicit ~~null~~ access key)

IP2 BD: A name (≤5 words)

IP3 C: An access key

Return parameter:

RCAP 0: The accessed object

DR: UACC - 1/2

This action calls for the search of one or more directories to access an entry with name IP2. [Insert A, following] Although only one directory is included among the parameters to this action, every directory has a successor link which may specify another directory. Thus this action searches down this chain of directories until an entry with name IP2 is found, (hereafter referred to as "the" entry) or an empty successor link is encountered. In the latter case, the action performs an freturn.

If the search described above ends successfully, then the access procedure continues by searching the access list (of the entry which was just found) for an access pair with a lock matching the access key IP3. It is an error if no such pair exists, otherwise the options from this pair are used as described below.

If the entry is not a soft link (i.e., it is an ownership entry or a hard link), a capability for the object it contains is fabricated, possessing the options selected above, and returned via the return parameter mechanism. In case of a soft link, however, the entry specifies an object indirectly by giving a directory, a name, and an access key. Thus the whole look up procedure is applied to these new parameters,

[INSERT A]

The name is specified by a sequence of ~~words containing~~ 7-bit characters* packed 8 to a word. The rightmost 56 bits of each word are significant, and the leftmost 4 bits are ignored. If the last word contains ~~fewer~~ than 8 characters, say n, then the last $8-n$ character positions of this word, as well as all character positions in any subsequent words of the block datum must contain the blank character ($=000_8$). Furthermore, only printable characters (codes 001_8 thru 136_8 inclusive) are allowed in names.

---

*In the TSS rotated ASCII code.

resulting ultimately in a capability for some object. This capability, modified by turning off any options not set both in its original option field and the options selected from the access list of the soft link, is returned via the return parameter mechanism.

Both soft links and successor links may result in a circular structure, which would cause the access procedure to loop infinitely unless some precaution is taken. Thus chains of successor links are never followed more than k deep.* Similarly chains of soft links are never followed more than l deep.*

Possible errors while accessing an object from a directory:

Class	Number	Modifier	Description
EC.DIRCT	EN.BADNM		IP2 is null, or contains non printing character (including Mark).
E.OPER	E.CAPTY		IP3 (is) the null access key, but IP1 locked OB.IMPL
EC.DIECT	EN.IMPOT		The accessed entry had no lock matching IP3
EC.DIRECT	EN.LOOP2		More than 5 successor pointers were followed in the search for a name.
EC.DIRCT	EN.LOOP1		More than 5 softlink entries were followed in the search for a hardlink or ownership entry.

Also, any file opening error.

* k and l are assembly parameters, easily increased, and currently both have the value 5.

Access an Object from a Scan List (DR:ASCL)

Input parameters

IP1 C: A capability list, taken as a scan list (see below.)
IP2 BD: A name (≤5 words)

Return parameter:

RCAP 0: The accessed object.

A scan list is just a capability list used to hold (directory, access key) capability pairs. This action steps through the scan list doing a regular directory access with the name IP2 and the directory and access key from the current scan list pair. If the directory search succeeds, this action returns the resultant object, otherwise the procedure continues with the next scan list pair. If the end of the scan list, marked by the actual end of the capability list or an empty entry where a directory capability is expected, is reached the action performs an freturn.

Possible errors while accessing an object from a scan list.

Class	Number	Modifier	Description
EC.DIRCT	EN.NTDIR	i	The $(2 \cdot i)^{th}$ entry of IP1 contained an object other than a directory.
EC.DIRCT	EN.NTKey	i	The $(2 \cdot i+1)^{th}$ entry of IP1 did not contain an access key.

Also, any directory accessing error.

-Type

Create a Disk File Ownership Entry in a Directory (DR:CRFI)

Input parameters:

IP1 C: A directory (OB.CREIL)
IP2 BD: A name (≤ 5 words)
IP3 BD: Shape numbers (≤ 11 words)

Return parameters

KCAP 0: A capability for the new disk file
1: A capability for the ecs incarnation of the new disk file.

This action creates a new disk file and an ownership entry for the new file. The entry is placed in the directory IP1 with the name IP2 (for the form of IP2, see the description of DR:UACC), provided that the name isn't already in use in that directory. After the disk and the new ownership entry has been completed, file subsystem has been called to actually create the file (q.v.), a disk file capability (with all options on) and an ecs file capability (with options permitting read, write, etc.) are returned.

Possible errors while creating a disk file ownership entry:

Class	Number	Modifier	Description
EC.DIRCT	EN.BADNM		IP2 contains non printing character
EC.DIRCT	EN.DUPNM		IP2 duplicates name of existing entry
EC.DIRCT	EN.NOSPC		No room for new ownership entry in IP1

Also, any file opening or file creating error (see ——).

Create a Directory-Type Ownership Entry in a Directory

Input parameters:

IP1	C: A directory	(OB.CRDR)
IP2	BD: A name	(≤ 5 words)
IP3	D: Size of new directory in words.	
IP4	D: Funding directory flag	

This action creates a new directory object and places an ownership entry for it into the directory IP1 under the name IP2, which must not duplicate the name of an existing directory. Directories are implemented as one level disk files so the size of the new directory, IP3, must be within the allowable size for one level disk files (currently ~ 500 words).

A funding directory is one which contains a disk accounting record (DAR); accounting for objects owned by non-funding directories is goes through their nearest funding ancestor (for a definition of ancestor, see the writeup of DA:MVSP). If IP4 is nonzero (using a 60-bit zero test), the new directory is created as a funding one, otherwise not.

Possible errors while creating a directory:

Class	Number	Modifier	Description

Move Space Up/Down the Funding Directory Tree (DA:MVSP)

Input parameters:

IP1 C: Donor directory (@B.GIVE)

IP2 C: Donee directory (@B.GET)

IP3 D: Number of sectors of disk space to move.

Returned parameters:

[A single datum is sometimes returned in the caller's register X6; see below.]

Disk space is moved from the disk accounting record (DAR) contained in one funding directory to the DAR in another one, provided one directory is the nearest funding ancestor of the other, in either order. [If α is any directory and $\beta, \gamma, \cdots, \omega$ are all the directories whose own ownership entries are contained by α, then the descendants of α are just α itself together with all the descendants of $\beta, \gamma, \cdots$, and ω. If α is a descendant of α, then α is an ancestor of β.] A DAR contains a reserved space field and an occupied space field. The surplus space held by a DAR is defined to be the difference: reserved minus occupied. This action moves the minimum of (IP3, donor's surplus) from the donor to the donee. This involves increasing the donor's occupied field and the donee's reserved field and charge rate field by the amount moved, if the donor is the ancestor of the donee. Otherwise, if the move is from descendant to ancestor, the donor's reserved and charge rate fields, as well as the donee's occupied field, are decreased by the amount

moved. In case less space than requested was moved (IP3 > donor's surplus), the amount actually moved is returned in the caller's register X6, and the action returns. Otherwise X6 is not changed, and the action returns normally.

Possible errors while moving disk space:

Class	Number	Modifier	Description
E.PARMS	E.NEGPAR		Amount to move was negative.
EC.DIRCT	EN.RSRV	(1 or 2)	(Donor or donee) directory was exclusively open.
EC.DIRCT	EN.NDAR	(1 or 2)	(Donor or donee) is not a funding directory (contains no DAR).
E.ABLOCK	E.NOABLK		One of the DAR's does not exist.

Also, any file-opening error (such as file gone, too many open files, etc.) may be returned.

Increment Funding Directory Charge Meter (DA:INCM)

Input parameters:

IP1 C: Funding directory (.OB.INMTR).

IP2 D: A sign-plus-59-bit integer.

No returned parameters.

If the directory is not a funding directory (does not contain a DAR), an error is returned. Otherwise the discontinuous charge meter in the DAR is incremented (using a 60-bit integer add) by IP3. The units of this meter are sector-milliseconds.

Possible errors while incrementing a charge meter:

Class	Number	Modifier	Description
EC.DIRCT	EN.RSRV		Directory was exclusively open.
EC.DIRCT	EN.NDAR		Directory contains no DAR.
E.ABLOCK	E.NOABLK		No such DAR.

Also, any file opening errors (see ——).

Set Accounting Tag in Funding Directory (DA:STAG)

Input parameters:

IP1 C: Funding directory (OB.SETAG).
IP2 D: Value for tag.

The disk accounting record (DAR) contained by a funding directory has an 18-bit field provided for the use of a higher-level accountant. This action sets the tag field to the value of IP2, whose high-order 42 bits must be zero.

Possible errors while setting tag field:

Class	Number	Modifier	Description
E.PARMS	E.BIGPAR		Not all 42 high-order bits of IP2 were zero.
EC.DIRCT	EN.RSRV		Directory was exclusively open.
EC.DIRCT	EN.NDAR		Directory contains no DAR.
E.ABLOCK	E.NOABLK		No such DAR.

Also, any file opening errors (see ——).

Display Disk Accounting Record of Funding Directory (DA:DSP)

Input parameter:

IP1 C: Funding directory

Returned parameters:

RD(1) - RD(8): The eight words of a DAR.

If this directory contains a disk accounting record (DAR), it is returned through the return parameter mechanism.

Possible errors while displaying a DAR:

Class	Number	Modifier	Description
EC.DIRCT	EN.RSRV		Directory was exclusively open.
EC.DIRCT	EN.NDAR		Directory contains no DAR.
E.ABLOCK	E.NOABLK		No such DAR.

Also, any file opening error (see —).

9 Feb 71

Display the N^{th} entry of a directory (DR:DSPN in OPERCL)

Parameters:

IP1) C: A directory

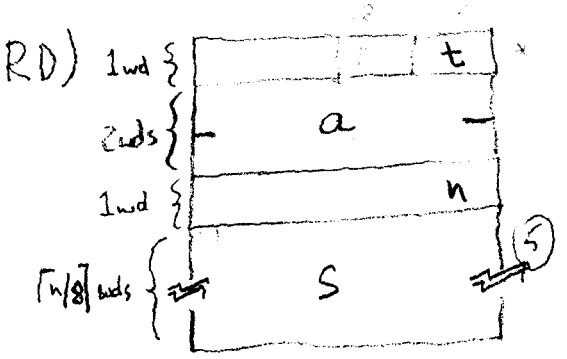
IP2) D: A positive integer, [(2,0) error if IP2 ≤ 0]

Action:

If the directory contains fewer than IP2 entries, FRETURN. Otherwise, information describing the $IP2^{th}$ entry is returned in the format shown below.

Returns:

RD)

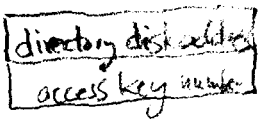


where t = entry type = { 0 ≡ soft link, 2 ≡ hard link, 3 ≡ ownership }

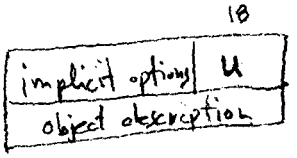
n = character count, $1 \le n \le 40$ *

s = a character string of length n, left justified in the rightmost 56 bits of each word. [Left most 4 bits=0.]

a = additional information:



if t = 0



if t = 2, 3

where u = 0, 1, 2, ..., 5 specifies File, directory, ..., access key

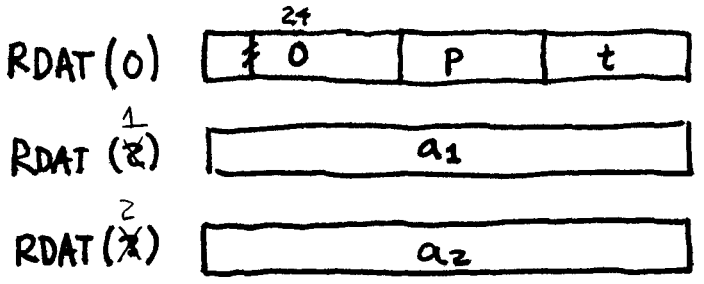
(* the number 40 is an assembly parameter of the directory system)

Display a Directory Entry (DR:DSPE)

Input parameters:

IP1 C: A directory
IP2 BD: A name

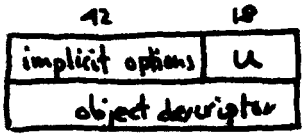
Return parameters:



p = number of access pairs this entry

t = entry type = { 0 if softlink; 2 if hardlink; 3 if ownership }

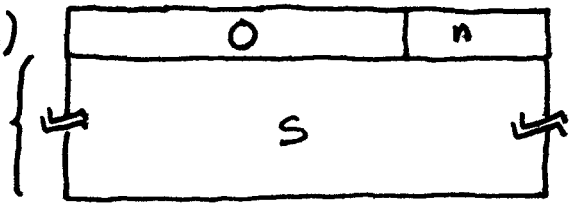
a_1, a_2 = { directory descriptor; access key descriptor } if t=0



if t=2,3

and, if t=0,

RDAT(3, ... 3+⌈n/8⌉)



and u = { 0 if file; 1 if directory; 2 if subprocess descr.; 3 if static name tag; 4 if dynamic name tag; 5 if access key }

n = character count, $1 \le n \le 40$

S = a character string of length n, LJZF in the rightmost 56 bits of each word [Leftmost 4 bits = 0]. } S = softlink name

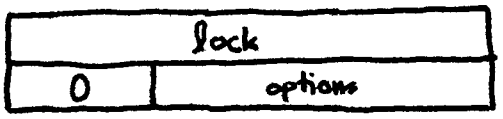
Display Directory Entry Access Pairs (DR:DAP)

Input parameters:

IP1 C: A directory
IP2 BD: An entry name
IP3 D: An integer i, $i \geq 1$
IP4 D: An integer n, $n \geq 1$

Return parameters:

RDAT($2(j-1)$)
RDAT($2(j-1)+1$)



} $(i+j-1)^{th}$ access pair

$j = 1, 2, \cdots, \min(n, k)$

where k is an assembly parameter.

Action:

The i^{th} through the $(i+n-1)^{th}$ access pairs of the given directory entry are returned as shown. The action never FRETURNs; if $(i+n-1)$ is greater than the total number of pairs, as many as are present (but never more than k) are returned.

Display Directory Successor Pointer (DR:DSPS)

Input parameter:

IP1 C: A directory

Return parameters:

Action:

If the directory IP1 has no successor pointer, this action FRETURNs. Otherwise, two words are returned as shown above.

DR:MOWN

Access Multiple Owned Objects in a Directory

Input parameters:

IP1 C: A directory (with OB.UACC, OB.IMPL)
IP2 C: A capability list
IP3 D: A starting index i, $i \geq 1$
IP4 D: A count k, $k \geq 1$

Return parameters:

RDAT(0) A count m of objects actually placed in the c-list, $m \geq 0$
RDAT(1) A count n of owned objects remaining in the directory, $n \geq 0$

Action:

Capabilities for objects appearing in ownership entries of a given directory (IP1) are copied into a given c-list (IP2). The index i (IP3) controls where the copying process begins -- $i = 1$ means the first directory ownership entry, $i = 2$ means the second, etc. Slots in the c-list are always filled starting with the first. The number m of capabilities moved (returned as RDAT(0)) is equal to the minimum of the following three quantities:

(i) the parameter k (IP4);
(ii) the length of the c-list;
(iii) $(p-i+1)$, where p is the number of ownership entries in the directory, or zero if $p < i$.

Returned as RDAT(1) is the quantity $n = p-m$ = the number of ownership entries in the directory after the ones accessed. Note that this action never freturns.

Errors:

No such (directory, c-list); non-positive (starting index, count).

PMcJ
27 Oct 71

DR:GMDA

Get Multiple Disk Adresses from a Directory File

Input parameters:

IP1 C: A disk file, in the format of a directory
IP2 C: An ecs file, to be filled with (disk address, unique name) words
IP3 D: A starting index i, $i \geq 1$
IP4 D: A count k, $k \geq 1$

Return parameters:

RDAT(0) A count m of disk addresses actually returned, $m \geq 0$
RDAT(1) A count n of disk addresses following those returned, $n \geq 0$

Action:

The file IP1 is assumed to represent a directory -- presumably one with no-longer-valid disk addresses. This action finds (disk address, unique name) words in the directory file and copies them into the ecs file IP2. Copying begins with the i-th disk address encountered, while the ecs file is always filled up starting at its file address zero. The number of disk-address words copied is m, the minimum of:

(i) the count k (IP4);
(ii) the length of the ecs file;
(iii) $(p-i+1)$, where p is the total number of disk addresses contained in the directory file, or zero if $i > p$.

~~Missing blocks in the ecs file will be created as needed.~~ Returned parameters include m and $n = p-m$ = the number of disk addresses in the directory file after the ones accessed. The action never freturns.

Errors:

No such (disk file, ecs file); incorrect format for directory file; non-positive (starting index, count).

PMcJ
27 Oct 1971

DR:PMDA

Put Multiple Disk Addresses into a Directory File

Input parameters:

IP1 C: A disk file, in the format of a directory
IP2 C: An ecs file, containing (disk address, unique name) words
IP3 D: A starting index i, $i \geq 1$
IP4 D: A count k, $k \geq 1$

Return parameters:

RDAT(0) A count m of disk addresses actually replaced, $m \geq 0$
RDAT(1) A count n of disk addresses yet to be replaced, $n \geq 0$

Action, errors:

Same as DR:GMDA (Get Multiple Disk Address), except the disk-address words are copied from the ecs file to the directory file, rather than the other way around.

PMcJ
27 Oct 1971