

2.0 only

ACTION_CHANGE_MODE **1028** **ChangeMode(type,obj,mode)**
 ARG1: LONG Type of object to change - either CHANGE_FH or CHANGE_LOCK
 ARG2: BPTR object to be changed
 ARG3: LONG New mode for object - see ACTION_FINDINPUT, and
 ACTION_LOCATE_OBJECT

 RES1: BOOL Success/Failure (DOSTRUE/DOSFALSE)
 RES2: CODE Failure code if RES1 is DOSFALSE

This action requests that the handler change the mode of the given file handle or lock to the mode in ARG3. This request should fail if the handler can't change the mode as requested (for example an exclusive request for an object that has multiple users).

2.0 only

ACTION_COPY_DIR_FH **1030** **DupLockFromFH(fh)**
 ARG1: LONG fh_Arg1 of file handle

 RES1: BPTR Lock associated with file handle or NULL
 RES2: CODE Failure code if RES1 = NULL

This action requests that the handler return a lock associated with the currently opened file handle. The request may fail for any restriction imposed by the file system (for example when the file handle is not opened in a shared mode). The file handle is still usable after this call, unlike the lock in ACTION_FH_FROM_LOCK.

2.0 only

ACTION_PARENT_FH **1031** **ParentOfFH(fh)**
 ARG1: LONG fh_Arg1 of File handle to get parent of

 RES1: BPTR Lock on parent of a file handle
 RES2: CODE Failure code if RES1 = NULL

This action obtains a lock on the parent directory (or root of the volume if at the top level) for a currently opened file handle. The lock is returned as a shared lock and must be freed. Note that unlike ACTION_COPY_DIR_FH, the mode of the file handle is unimportant. For an open file, ACTION_PARENT_FH should return a lock under all circumstances.

2.0 only

ACTION_EXAMINE_ALL **1033** **ExAll(lock,buff,size,type,ctl)**
 ARG1: BPTR Lock on directory to examine
 ARG2: APTR Buffer to store results
 ARG3: LONG Length (in bytes) of buffer (ARG2)
 ARG4: LONG Type of request - one of the following:
 ED_NAME Return only file names
 ED_TYPE Return above plus file type
 ED_SIZE Return above plus file size
 ED_PROTECTION Return above plus file protection
 ED_DATE Return above plus 3 longwords of date
 ED_COMMENT Return above plus comment or NULL
 ARG5: BPTR Control structure to store state information. The control
 structure **must** be allocated with AllocDosObject()!

 RES1: LONG Continuation flag - DOSFALSE indicates termination
 RES2: CODE Failure code if RES1 is DOSFALSE

This action allows an application to obtain information on multiple directory entries. It is particularly useful for applications that need to obtain information on a large number of files and directories.

This action fills the buffer (ARG2) with partial or whole ExAllData structures. The size of the ExAllData structure depends on the type of request. If the request type field (ARG4) is set to ED_NAME, only the ed_Name field is filled in. Instead of copying the unused fields of the ExAllData structure into the buffer, ACTION_EXAMINE_ALL truncates the unused fields. This effect is cumulative, so requests to fill in other fields in the ExAllData structure causes all fields that appear in the structure *before* the requested field will be filled in as well. Like the ED_NAME case mentioned above, any field that appears after the requested field will be truncated (see the ExAllData structure below). For example, if the request field is set to ED_COMMENT, ACTION_EXAMINE_ALL fills in all the fields of the ExAllData structure, because the ed_Comment field is last. This is the only case where the packet returns entire ExAllData structures.

```
struct ExAllData {
    struct ExAllData *ed_Next;
    UBYTE  *ed_Name;
    LONG    ed_Type;
    ULONG   ed_Size;
    ULONG   ed_Prot;
    ULONG   ed_Days;
    ULONG   ed_Mins;
    ULONG   ed_Ticks;
    UBYTE  *ed_Comment;    /* strings will be after last used field */
};
```

Each ExAllData structure entry has an ead_Next field which points to the next ExAllData structure. Using these links, a program can easily chain through the ExAllData structures without having to worry about how large the structure is. *Do not examine the fields beyond those requested* as they certainly will not be initialized (and will probably overlay the next entry).

The most important part of this action is the ExAllControl structure. It *must* be allocated and freed through AllocDosObject()/FreeDosObject(). This allows the structure to grow if necessary with future revisions of the operating and file systems. Currently, ExAllControl contains four fields:

Entries - This field is maintained by the file system and indicates the actual number of entries present in the buffer after the action is complete. Note that a value of zero is possible here as no entries may match the match string.

LastKey - This field *must* be initialized to 0 by the calling application before using this packet for the first time. This field is maintained by the file system as a state indicator of the current place in the list of entries to be examined. The file system may test this field to determine if this is the first or a subsequent call to this action.

MatchString - This field points to a pattern matching string parsed by ParsePattern() or ParsePatternNoCase(). The string controls which directory entries are returned. If this field is NULL, then all entries are returned. Otherwise, this string is used to pattern match the names of all directory entries before putting them into the buffer. The default AmigaDOS pattern match routine is used unless MatchFunc is not NULL (see below). Note that it is not acceptable for the application to change this field between subsequent calls to this action for the same directory.

MatchFunc - This field contains a pointer to an alternate pattern matching routine to validate entries. If it is NULL then the standard AmigaDOS wild card routines will be used. Otherwise, MatchFunc points to a hook function that is called in the following manner: