TO: Distribution

FROM: Gary C. Dixon

DATE: November 28, 1973

SUBJECT: Some Thoughts on the Handling of Files

Currently, two classes of files are implemented by Multics: those which can be stored in a single Multics segment (single-segment files or SSFs); and those which must be stored in a group of Multics segments because a single segment is too small to hold all of the data (multi-segment files or MSFs).

MSFs were implemented under Multics by extending four functional areas in the user ring of the system. These areas provide for: the creation, expansion, and truncation of MSFs (the file_ IOSIM, msf_manager_, tssl_, etc); the printing and punching of MSFs (the IO Daemon); the deletion of MSFs (the delete command and delete_ subroutine); and listing MSF attributes (the list command). Only one of these four areas (the list command) differentiates between SSFs and MSFs. The routines in the other three areas provide external interfaces to the user which can manipulate SSFs and MSFs interchangeably.

Our past experience with files has shown that treating SSFs and MSFs alike is beneficial, because it simplifies the code in many system and user programs which manipulate files. Extending this policy into other functional areas of the system will further facilitate the handling of files. Areas of interest include:

- listing file attributes
- file access control
- copying and moving files
- truncating files
- setting other file attributes

Experience has also shown that our implementation of MSFs has changed in the past and is likely to change in the future as new storage system attributes or other facilities are added to the system. In order to insulate users from these changes, it seems wise to implement a new user ring interface to the storage system which is independent of the particular implementation of files.

This MTB proposes some changes which can be made in each of the above areas to unify the handling of files. It attempts to bring together the ideas proposed in recent MTBs and MCRs to solve different aspects of what is really a single, large problem, the file problem. The purpose is to generate thought and discussion on the file problem as a whole. This hopefully will lead to better solutions for the various aspects of the problem.

Your comments on this MTB would be appreciated, preferably in writing. Comments may be sent by IPC Courier to:

GDixons bin, Bldg 39

or by Multics mail, using the command:

```
mail comment GDixon PDO
```

I. A Description of an MSF

The following is a brief description of the properties of an MSF, including mappings of the new directory control attributes onto MSFs.

A. Organization of an MSF

An MSF is a group of segments or links to segments. These segments and links must reside in a single directory. The segments and links are known as MSF components. (2) The MSF components contain the data which resides within the file. The directory is known as the MSF directory. It contains the attributes of each of the MSF components, as well as some of the attributes of the MSF itself. The *n* MSF components are named by a sequence of integer names beginning with 0 (i.e., 0, 1, 2, ..., *n*-1). All MSF components (except perhaps the last) have the same maximum segment length. Usually, this length is equal to the value of sys_info$max_seg_size, but it may be another number. The bit count on MSF components indicates the number of bits of data contained in each component. Usually, the bit counts on all MSF components but the last have the same value, but this may not be the case.

B. Attributes of an MSF

- ACL
- author
- bit count
- bit count author
- current length (in records)
- date dumped
- date entry modified
- date modified
- date used
- entry type ("file")
- file type (unstructured, sequential, or indexed)
- maximum segment length (max length of each MSF component)
- maximum file length (maximum length of entire MSF)
- MSF indicator
- names
- records used
- ring brackets (r1, r2, r3)
- safety switch setting
- unique identifier

---

(2) We shall see that many of the attributes of MSF components which are links are stored on the target segment of the link.

The copy switch should not be settable for MSF components or for directories of any kind.

C. How the MSF Attributes are Stored Under a User Ring Implementation of MSFs

ACL: the ACL of an MSF specifies how all users may access the data in the file. Therefore, it must control access to each of the MSF components, and to the MSF directory. An MSF ACL is a segment ACL. It is stored as the initial ACL for segments of the MSF directory, in the ring of the validation level at which the MSF was created. In addition, the modes in the MSF ACL are mapped into directory modes (as described below), and the mapped ACL is placed on the MSF directory. An MSF ACL should not contain a mode of "e", since MSFs can never be executed.

author: the author of an MSF is the author on the MSF directory.

bit count: the bit count of an MSF is the sum of the bit counts on all of the MSF components.

bit count author: the bit count author of an MSF is the bit count author of the MSF directory (i.e., the author of the MSF indicator).

current length: the current length of an MSF is the sum of the current lengths of all of the MSF components.

date dumped: the date dumped of an MSF is the date dumped of the MSF directory.

date entry modified: the date entry modified of an MSF is the date entry modified of the MSF directory.

date modified: the date modifed of an MSF is the date modified of the MSF directory. (Remember that dtm for the MSF directory is updated whenever the bit count of any MSF component is set.)

date used: the date used of an MSF is the date used of the MSF directory.

entry type: the entry type of an MSF is "file". (The "segment" entry type returned by the status command should be renamed to "file".)

file type: the file type of an MSF may be any of the file types defined in MSB-113 (iox_). These include "unstructured", "sequential", and "indexed". This file type is stored as an attribute of the MSF directory.

maximum segment length: this attribute controls how large MSF component segments can grow. The maximum segment length attribute of an MSF is stored on the MSF directory and on each of the MSF component segment, with the exception that a smaller value may be stored on the final MSF component (see maximum file length below).

maximum file length: this attribute controls how large the entire MSF can grow. It is stored as an attribute of the MSF directory. When adding a new component $\underline{i}$ to an MSF, the component's maximum segment length is computed by the algorithm:

maximum_file_length - (maximum_segment_length * ($\underline{i}$-1))

If this value is less than or equal to zero, then component $\underline{i}$ cannot be created, and the out_of_bounds condition is signalled; component $\underline{i}$-1 is effectively the last component of the MSF. The maximum file length is also a new attribute of an SSF which can be set to prevent an SSF from ever being converted into an MSF. The MSF component segments would have a maximum file length setting of sys_info$max_seg_size to prevent them from being converted into MSFs.

MSF indicator: the MSF indicator is a count of the number of MSF components. It is stored as the bit count of the MSF directory.

names: the names of an MSF are the names on the MSF directory.

records used: the count of records used by an MSF is the sum of the records used by each of the MSF components plus the records used by the MSF directory. This sum is equivalent to the quota used by the MSF directory plus the records used by the MSF directory.

ring brackets: the ring brackets of an MSF are the ring brackets on each of the MSF components. The ring brackets on all MSF components must be the same. If the ring brackets of the MSF are r1,r2,r3, then the ring brackets on the MSF directory must be r1,r2. Since an MSF cannot be executed, it cannot be a gate: therefore, r2 should always equal r3.

safety switch setting: the safety switch setting of an MSF is the safety switch setting on each of the MSF components and on the MSF directory. The system should maintain identical safety switch settings on the MSF directory and all of the MSF components.

unique identifier: the unique identifier of an MSF is the unique identifier of the MSF directory.

D. Rules for Mapping MSF Attributes Into MSF Directory Attributes and Vice Versa

```
access modes:  rw -> sma     sma -> rw
               r  -> s       s   -> r

ring brackets:  r1,r2,r3 -> r1,r2    r1,r2 -> r1,r2,r2
```

The fact that an MSF ACL cannot contain a mode of "e" allows us to define the one-to-one mapping between the segment access modes of the MSF ACL and the directory access modes of the MSF directory's ACL. This one-to-one mapping, in turn, allows us to determine a given user's access to the data stored within the MSF components by examining the ACL of the MSF directory.

Similarly, the fact that an MSF cannot be a gate implies that an MSF's second and third ring brackets (r2 and r3) must be equal, and this allows us to determine the ring brackets of the MSF components from the ring brackets of the MSF directory.

E. Implications of the User Ring Implementation of MSFs

ACL: Since the ACL of an MSF is stored as the initial ACL for segments of the MSF directory in the ring of the validation level at which the MSF was created, the MSF ACL will automatically be placed on new MSF components, only when the new components are created at the validation level at which the MSF was created. If proper access control lists are to be maintained on new MSF components, then the use of the initial ACL implies that an MSF should only be written at a single validation level, the level at which it was created. Furthermore, it implies that an MSF's ACL should only be modified at the validation level at which the MSF was created.

author: since the author of an MSF is the author of the MSF directory, we can use hcs_$get_author as a uniform interface to obtain the author of any (SSF or MSF) file or directory; however, the author of a file may change as an SSF is expanded into an MSF, or as an MSF is truncated into an SSF.

bit count: since the bit count of an MSF is the sum of bit counts of the MSF components, and since each component may have a different bit count, it is necessary to obtain the bit count of each MSF component in order to compute the bit count of the MSF.

bit count author: since the bit count author of an MSF is the bit count author of the MSF directory (the MSF indicator author, if you will), the MSF indicator must be set (or reset) whenever the bit count of any component is set. This allows us to use hcs_$get_bc_author as a uniform interface to obtain the bit count author of any (SSF or MSF) file or directory.

current length: as with the bit count, the current length of an MSF cannot be computed without obtaining the current lengths of all of the MSF components.

maximum segment length: this attribute of MSF directories and components must be preserved when moving or copying an MSF, and it should be preserved when converting an SSF to an MSF, or vice versa.

maximum file length: this attribute of an SSF or an MSF must be preserved when copying or moving files, and when converting an SSF to an MSF, or vice versa.

safety switch setting: the safety switch setting must be preserved when creating new MSF components, when converting an SSF to an MSF or vice versa, and when moving an MSF. Having the same safety switch setting on the MSF directory as on the MSF components allows us to use the hcs_$get_safety_switch to obtain the safety switch setting for any (SSF or MSF) file in a uniform manner.

unique identifier: since the unique identifier of an MSF is the unique identifier of the MSF directory, the unique identifier of a file will change when that file is converted from an SSF to an MSF, or vice versa.

F. MSF Attributes Already Implemented

ACL: storage of the ACL attribute of an MSF, as described above, has already been implemented by msf_manager_$acl_replace.

author: the author of an MSF is set by the storage system when the MSF directory is created.

date dumped: is updated automatically by the storage system when the MSF directory is dumped.

date entry modified: is updated by the storage system whenever the ACL of an MSF is replaced by msf_manager_$acl_replace or msf_manager_$replace_acl (obsolete), whenever the ring brackets of the MSF directory are changed, and whenever the MSF indicator is changed.

date time modified: is updated by the storage system whenever the ACL, ring brackets, bit count or contents of one of the MSF components is modified.

date used: is modified whenever one of the MSF components is used.

maximum length: is maintained now by msf_manager_.

MSF indicator: is maintained now by msf_manager_.

names: are maintained by msf_manager_ and by storage control, and are reported by the status commands and subroutines (including list).

records used: are maintained in the directory entry of each MSF directory and component, and are properly totalled and reported by the list command.

II. New or Changed Interfaces for Handling Files

As mentioned in the introduction of this MTB, new user interfaces which treat files in a uniform manner are needed in the areas of: listing file attributes; file access control; copying and moving files; truncating files; and setting other file attributes. Since most of these user interfaces will be implemented in the user ring, they will have the additional advantage of insulating system and user programs which manipulate files from the particular strategy for implementing those files, whether that strategy is implemented in the user ring or in the hardcore ring. This section of the MTB briefly outlines the facilities which should be provided by these interfaces.

A. Listing File Attributes

A fundamental component of any solution to the file problem is an interface which provides status information for (both SSF and MSF) files, directories, and links in a uniform manner. Therefore, I propose a status_ subroutine which might provide the following information. (3)

---

(3) Note that the information is presented as a structure declaration to simplify its presentation. What is important is the content of the structure, and not its organization.

```
     dcl 1 branch_status          aligned,
          (2 entry_type           fixed bin(17),
           2 file_type            fixed bin(17),
           2 Nnames               fixed bin(17),
           2 Onames               fixed bin(17)) unaligned,
           2 dtm                  bit(36),
           2 dtu                  bit(36),
           2 dtem                 bit(36),
           2 dtd                  bit(36),
          (2 mode                 bit(3),
           2 rb (3)               fixed bin(3)) unaligned,
           2 length               fixed bin(35),
           2 records              fixed bin(35),
           2 max_seg_length       fixed bin(35),
           2 max_file_length      fixed bin(35),
           2 bitcount             fixed bin(35),
          (2 msf_indicator        fixed bin(17),
           2 copy_sw              bit(1),
           2 safety_sw            bit(1)) unaligned,
           2 uid                  bit(36);

     dcl 1 link_status            aligned
                                  based (addr (branch_status)),
          (2 entry_type           fixed bin(17),
           2 file_type            fixed bin(17),
           2 Nnames               fixed bin(17),
           2 Onames               fixed bin(17),
           2 Lpath                fixed bin(17),
           2 Opath                fixed bin(17)) unaligned,
           2 pad                  bit(36),
           2 dtem                 bit(36),
           2 dtd                  bit(36);
```

status_ would have a calling sequence similar to hcs_$status_long, and would return the information above, which is similar to that returned by hcs_$status_long. The major differences are that the status information would report file status (i.e., SSF attributes for an SSF, and MSF attributes, as defined above, for an MSF), rather than segment status, and that the status_ subroutine would insulate user programs from knowing how MSFs are implemented. Having such an interface will greatly facilitate the implementation of the other new interfaces described below, and will be useful to user's also.

A get_max_length_ subroutine should be created to allow the user to get the maximum segment length of (SSF and MSF) files in a uniform manner. status_ might call this subroutine to obtain the max_length for the branch_status block shown above, or the get_max_length_ subroutine might be implemented as an entry point of status_ with an internal subroutine which could be called by status_.

Similarly, a get_max_file_length_ subroutine should be created to provide a uniform interface for obtaining maximum file lengths.

The status command should be modified to call status_. For files, it should report the "file" entry type, rather than "segment" or "directory". It should also report both the MSF bit count and MSF indicator values for MSFs.

The list command should be modified, as proposed in MCR 131, to accept a -file (-fl) option which will cause it to list both SSFs and MSFs in a single group. The existing options -segment and -msf should continue to work as they do today, but -file should become the default option, instead of -segment.

A new subroutine, list_, should be created which would return an array of branch_status structures for the entries matching the star name. This would seem to provide the cleanest interface for the list command, and would be useful to other system and user programs as well, as we shall see below.

An entry point should be added to the files active function which treats SSFs and MSFs alike. One possible change which would avoid conflicts in terminology is to change the meaning of the files entry point to return the names of files (SSFs and MSFs) which match a star name, and to create a new entries entry point which acts like the current files entry point (i.e., returns all entries which match a star name). Although this change would require some user conversion at MIT, I think the advantages of avoid the terminology conflict outweigh the short term user inconvenience of such a conversion.

A "file" control argument should also be added to the exists active function to allow it to identify a file.

The entry points hcs_$get_author_, hcs_$get_bc_author_, and hcs_$get_safety_sw_ should be added to hcs_. These entries would be identical to their counterparts not ending in underscore. We can then create an implementation-independent interface for these status primitives without impairing their efficiency by adding the names get_author_, get_bc_author_, and get_safety_sw_ to hcs_.

The listacl, list_inacl_dir and list_inacl_seg commands are discussed in the next section which deals with the entire question of access control for files.

B. File Access Control

Just as a user needs a uniform interface for obtaining the status of files, so he needs uniform interfaces for listing and setting the ACL and ring brackets of files, interfaces which insulate him from the particular implementation of file access control. I propose that we create add_acl_entries_, delete_acl_entries_, list_acl_, and replace_acl_ subroutines which manipulate SSF and MSF ACLs uniformly. These subroutines would operate by calling the appropriate hcs_ entry points for manipulating segment ACLs. If an error is returned indicating that the file is not a segment, then status_ would be called, and the appropriate action would be taken for an MSF. For list_acl_, the action would be to call hcs_$list_inacl. For the other entries, it would involve manipulating the MSF component and directory ACLs and the initial ACL for segments of the MSF directory.

The listacl, setacl, and deleteacl commands should call list_ to determine entries type of entries matching their input star name, and then call the subroutines above for files, and the directory ACL entry points in hcs_ for directories.

The copy_acl_ subroutine should be changed to call these new subroutine interfaces. This would enable copy_ to copy ACLs from the source file onto the target file without undue difficulty, and it would facilitate the proposed changes to make the copy and move commands handle MSFs.

The list_iacl_dir, list_iacl_seg, set_iacl_dir, set_iacl_seg, delete_iacl_dir, and delete_iacl_seg commands should be changed to call list_ to identify MSF directories. The commands should refuse to manipulate the initial ACLs of MSF directories unless the user has specified a -force option indicating that he knows what the effects of changing the initial ACL of an MSF directory are. In general, the contents of the directory initial ACL of the MSF directory should be considered undefined; the contents of the segment initial ACL of the MSF directory should be considered part of the MSF ACL.

A new set_ring_brackets_ interface should be created to manipulate the ring brackets of files and directories. This subroutine should implement the ring bracket strategy described above for MSFs. The set_ring_brackets command should call this new subroutine.

In addition, perhaps a command should be written which changes the validation level at which an MSF can be written or its ACL be modified. The need for such an interface has not been well established.

C. Copying/Moving Files

The changes to the copy and move commands (proposed in MCR 030 by Tom Casey and in MCR 131 by Max Smith) would allow these commands to copy/move MSFs. I feel that these changes should be implemented by dividing the commands into an argument-processing command interface, and a strategy-defining subroutine, copy_ and move_, which would be called by the commands and by other system and user programs. These subroutines could be called much as delete_ is today, with switches to control which type of file is to be copied/moved. MSFs. Some of the special requirements of the copy_ and move_ subroutines with respect to MSFs include preserving the maximum segment and file length setting on the MSF components when they are moved or copied, and preserving the safety switch setting on the MSF directory and components when an MSF is moved.

D. Truncating Files

As I pointed out in MTB-008, we need a way for user programs to truncate files. Since publishing MTB-008, I have revised truncate_'s proposed method of operation. When called with a directory path, and entry name, and a bit length (BL), truncate_ calls status_ to determine the entry type. For directories, it would return error_table_$dirseg. For SSF's, it converts the specified bit length to a word length (WL):

```
     WL = divide (BL+36, 36, 0, 0);
```

and calls hcs_$truncate_file with this word length. If no error code is returned, then if:

```
     mod (BL, 36) > 0
```

it zeroes the right-most bits of the single word, WL, which are beyond the bit length BL. Finally, it calls hcs_$set_bc to set the bit count. For MSFs, it loops through the MSF components summing their bit counts until:

```
     MSF_bit_count(i-1) < BL <= MSF_bit_count(i)
```

where MSF_bit_count(i) is the sum of the bit counts of MSF components 0 through i. It then truncates MSF component i to a bit length of:

```
     BL - MSF_bit_count(i-1)
```

in the manner described above for truncating SSFs, and deletes MSF components i+1 through n-1, if they exist. If MSF component i is not found, error_table_$badarg is returned.

The truncate command should be modified to call the truncate_ subroutine. For the sake of compatibility, the truncation length given to the truncate command should be in decimal words, with options to specify the length in octal words or decimal bits.

E. Setting Other File Attributes

A new interface, set_bc_, should be implemented which sets the bit count of any file or directory. This subroutine calls status_ to determine the entry type of the target, and calls hcs_$set_bc for SSFs and directories. For MSFs, it loops through the MSF components summing up the MSF bit count until:

MSF_bit_count(i-1) < BL <= MSF_bit_count(i)

where MSF-bit-count(i) is the sum of the bit counts of MSF components 0 through i, and BL is the bit length specified by the caller to set_bc_. The bit count of MSF component i is then set to:

BL - MSF_bit_count(i-1)

and the bit counts of MSF components i+1 through n-1 are set to zero. If the ith MSF component is not found, error_table_$bigarg is returned.

The set_bit_count command should be changed to call set_bc_, to allow it to be used on MSFs.

A set_safety_switch_ subroutine should be created to set the safety switch in a uniform manner for all files and directories. The safety_switch_on and safety_switch_off commands should be changed to call these subroutines.

A set_max_length_ subroutine should be created to set maximum segment length values for all files in a uniform manner. The set_max_length command should be changed to call this subroutine.

Similarly, a set_max_file_length_ subroutine and set_max_file_length command should be created to set bounds on the size of files in a uniform manner.

F. Summary

Appendix A lists the new subroutines proposed in Section II of this MTB. Appendix B lists the changes proposed to existing commands and subroutines in Section II, and proposes changes to several other commands and subroutines. Appendix C lists the commands and subroutines which handle files correctly. Appendix D lists the commands and subroutines which reference caller-supplied segments, and which should not be modified to handle files. Appendix E lists commands and subroutines which do not reference caller-supplied segments. In all appendices except Appendix A (new things), the commands and subroutines surveyed were chosen from the union of MPM Parts II and III, and the commands and subroutines in system_library_standard.

III. Implementing MSFs in the Hardcore Ring

The thoughts on files expressed above are based on the assumption that we continue implementing MSFs in the user ring, rather than in the hardcore ring. However, as noted above, this has some undesirable and sometimes obscure implications on the way MSFs must be handled. This section discusses the implications on the thoughts above of implementing MSFs in the hardcore ring. It does not propose any specific hardcore implementation of MSFs, but it does identify some of the requirements of such an implementation.

A. Limitations of a User Ring Implementation of MSFs

First, I will briefly recap the limitations which implementinging MSFs in the user ring places on the way MSFs are handled. User ring implementation of MSFs:

1) forces the user to have sma access to the directory which contains an SSF in order to convert that SSF to an MSF. The same applies when converting an MSF to an SSF. The user should be able to perform this conversion with null access to the containing directory, just as he can set the bit count on a segment with null access on its containing directory.

2) prevents the user from safely writing into an MSF and from setting access to the MSF, except at the validation level at which the MSF was created. Restrictions on writing into an MSF should be controlled only by its ACL and by its write ring bracket. Restrictions on setting access to an MSF should be controlled only by the ACL and ring brackets of the directory containing the MSF.

3) allows user programs to store MSF attributes in an inconsistent manner (e.g., not resetting the bit count author on the MSF directory when changing the bit count of an MSF component; not storing uniform ACLs, maximum segment lengths, or safety switch settings on MSF components, etc). If users are to depend upon the reliability of these attributes, then they should be maintained by the hardcore ring without the possibility of user tampering.

4) prevents retention of the author and unique identifier across SSF to MSF conversions or vice versa. At least the author, and probably the unique identifier should be retained across such conversions.

5) requires accessing the directory entries for each MSF component to compute the current length or bit count of the MSF. These attributes could be propogated up to the MSF directory just as quota (records used) is now.

B. Implications of a Hardcore Implementation of MSFs

All of the limitations listed in the previous subsection can be removed by implementing some MSF attributes in the hardcore ring, and by extending the function of some existing hardcore gate entries to handle MSFs. This would allow many of the system and user programs which call these gate entries to handle MSFs without further conversion.

The savings from not having to convert many user ring programs would seem to outweigh the added hardcore complexity caused by implementing MSFs in the hardcore ring. Therefore, hardcore implementation of MSFs seems like the most desirable course of action. However, a hardcore implementation is more complicated to construct than a user ring implementation. The Multics Project probably does not have the resources at this time to build a hardcore implementation. We may have to adopt an interim plan of extending the current user ring implementation of MSF attributes in some of the areas indicated in the previous section, while planning in the future to reimplement some of these attributes in the hardcore ring to remove the user ring implementation's limitations. This approach is practical if we create implementation-independent interfaces to the storage system to insulate system and user programs from such reimplementations.

Appendix A

New Subroutines for Handling Files

add_acl_entries_
copy_
delete_acl_entries_
get_max_file_length_
get_max_length_
list_
list_acl_
move_
replace_acl_
set_bc_
set_max_file_length
set_max_file_length_
set_max_length_
set_ring_brackets_
set_safety_switch_
status_
truncate_

Appendix B

Changes to Existing Commands and Subroutines for Handling Files

adjust_bit_count_: handle MSFs.

code: handle MSFs.

compare_ascii_: compare two MSFs (or an MSF composed of short components with a long SSF, etc).

convert_characters: convert contents of MSFs.

copy: call list_; then, for each entry matching star name, call copy_.

copy_acl_: call status_; for files, call list_acl_ and add_acl_entries_; for directories, call hcs_$list_dir_acl and hcs_$add_dir_acl_entries.

copy_seg_: call status_.

decode: decode MSFs.

delete_iacl_dir: identify MSF directories (by calling list_) and reject them.

delete_iacl_seg: identify MSF directories (by calling list_) and reject them.

deleteacl: call list_; for files, call delete_acl_entries_: for directories, call hcs_$delete_dir_acl_entries.

dump_segment: dump portions of MSFs; rename dump_segment to dump_file.

edm: edit MSFs.

exists: add "file" control this argument to extend this active function's operation; call status_.

files: rename files entry point to entries; create a new files entry point; call list_.

getquota: treat MSF directories as segments when reporting the quota on directories matching a star name; this can be done by calling list_.

locall: accept MSFs as output or input segments for "locall read" and "locall write" commands, respectively.

list: add "-file" option, which becomes the default option; call list_.

list_iacl_dir: identify MSF directories (by calling list_) and reject them.

list_iacl_seg: identify MSF directories (by calling list_) and reject them.

listacl: call list_; for files, call list_acl_; for directories, call hcs_$list_dir_acl.

listnames: add "file" option, which becomes the default option; call list_.

listtotals: add "file" option, which becomes the default option; call list_.

move: call list_; then, for each entry matching star name, call move_.

msf_manager_: preserve values of copy switch, max length, safety switch, bit count, bit count author when converting from SSF to MSF or vice versa, or when adding another MSF component (call status_ to help do this); zero the high-order bits of the last word of a truncated MSF component, those which lie beyond the bit count of the segment.

parse_file_: parse MSFs by calling msf_manager_.

print: print MSFs.

qedx: edit MSFs.

safety_switch_off: call set_safety_switch_.

safety_switch_on: call set_safety_switch_.

set_bit_count: call set_bc_.

set_iacl_dir: identify MSF directories (by calling list_) and reject them.

set_iacl_seg: identify MSF directories (by calling list_) and reject them.

set_max_length: call set_max_length_.

set_ring_brackets: call status_; for files, call set_ring_brackets_; for directories, call hcs_$set_dir_ring_brackets.

setacl: call list_; for directories, call hcs_$add_dir_acl_entries; for files, call add_acl_entries_.

sort_file: sort MSFs by calling msf_manager_.

status: call list_ if star name given as input; otherwise, call status_.

truncate: call truncate_.

walk_subtree: call list_ to avoid walking through MSF directories.

Appendix C

Commands and Subroutines Which Handle Files Correctly

delete
delete_
delete_dir
deleteforce
dl_handler_
dprint
dprint_
dpunch
file_
file_output
nd_handler_
runoff
tssi_

Appendix D

Commands and Subroutines Which Need Not be Modified to Handle Files

abbrev
alloc_
alm (input, object)
alm_abs
archive
archive_sort
area_
area_assign_
basic
basic_run
basic_system
bind
create
createdir
debug
enter_abs_request
error_table_compiler
exec_com
fortran (input, object)
fortran_abs
freen_
get_system_free_area_
hcs_$add_acl_entries
hcs_$add_dir_acl_entries
hcs_$add_dir_inacl_entries
hcs_$add_inacl_entries
hcs_$append_branch
hcs_$append_branchx
hcs_$del_dir_tree
hcs_$delentry_file
hcs_$delentry_seg
hcs_$delete_acl_entries
hcs_$delete_dir_inacl_entries
hcs_$delete_inacl_entries
hcs_$fs_get_mode
hcs_$fs_move_file
hcs_$get_author
hcs_$get_bc_author
hcs_$get_dir_ring_brackets
hcs_$get_max_length
hcs_$get_ring_brackets
hcs_$get_safety_sw
hcs_$initiate
hcs_$initiate_count
hcs_$list_acl_entries
hcs_$list_dir_acl_entries
hcs_$list_dir_inacl_entries
hcs_$list_inacl_entries
hcs_$make_ptr
hcs_$make_seg
hcs_$quota_get
hcs_$quota_move
hcs_$replace_acl
hcs_$replace_dir_acl
hcs_$replace_dir_acl_entries
hcs_$replace_dir_inacl
hcs_$replace_inacl
hcs_$set_bc
hcs_$set_bc_seg
hcs_$set_dir_ring_brackets
hcs_$set_max_length
hcs_$set_max_length_seg
hcs_$set_ring_brackets
hcs_$set_safety_sw
hcs_$set_safety_sw_seg
hcs_$star_
hcs_$star_list_
hcs_$status_
hcs_$status_long
hcs_$status_minf
hcs_$status_mins
hcs_$terminate_file
hcs_$terminate_name
hcs_$terminate_noname
hcs_$terminate_seg
hcs_$truncate_file
hcs_$truncate_seg
help
indent
initiate
lisp
mail
make_commands
make_peruse_text
object_info_
peruse_text
pl1 (input, object)
pl1_abs
print_bind_map
print_link_info
print_motd
profile_data_
reorder_archive
runoff_abs

send_message	terminate
send_message_	trace_stack
set_search_rules	v1pl1 (input, object)
stu_	v1pl1_abs
term_	v5basic

Appendix E

Commands and Subroutines Which Do Not Reference Caller-Supplied Segments

ALL ACTIVE FUNCTIONS
 EXCEPT exists & files
active_fnc_err_
add_search_rules
addname
answer
broadcast_
calc
cancel_abs_request
cancel_daemon_request
change_error_mode
change_wdir
change_wdir_
check_info_segs
clock_
com_err_
command_query_
condition_interpreter_
conditon_
console_output
convert_binary_integer_
convert_date_to_binary_
convert_status_code_
copy_names_
cpu_time_and_paging_
cu_
cv_acl_
cv_bin_
cv_dec_
cv_dir_acl_
cv_dir_mode_
cv_float_
cv_mode_
cv_oct_
cv_userid_
date_time_
decam
decode_clock_value_
decode_descriptor_
decode_entryname_
default_handler_
delete_search_rules
deletecacl
deletename
discard_output_
display_component_name
endfile
enter
enterp
equal_
establish_cleanup_proc_
expand_path_
find_command_
fs_chname
get_at_entry_
get_com_line
get_default_wdir_
get_group_id_
get_pdir_
get_process_id_
get_wdir_
hcs_$append_link
hcs_$chname_file
hcs_$chname_seg
hcs_$fs_get_path_name
hcs_$fs_get_ref_name
hcs_$fs_get_seg_ptr
hcs_$get_process_usage
hcs_$initiate_search_rules
hcs_$reset_working_set
hcs_$set_search_rules
hcs_$wakeup
hold
how_many_users
ioa_
iomode
ios_
ipc_
link
link_length
list_abs_requests
list_daemon_requests
list_ref_names
listcacl
listen_
login
logout
lss_login_responder_
make_object_map_
move_names_
new_proc
nstd_
page_trace
plot_

prepare_mc_restart_
print_attach_table
print_dartmouth_library
print_default_wdir
print_linkage_usage
print_search_rules
print_wdir
program_interrupt
progress
random_
read_list_
ready
ready_off
ready_on
release
rename
reprint_error
resource_usage
reversion_
revert_cleanup_proc_
set_com_line
set_dartmouth_library
set_lock_
set_search_dirs
setcacl
signal_
standard_default_handler_
start
start_governor_
suffixed_name_
syn
system_info_
tape_
terminate_process_
timer_manager_
total_cpu_time_
transform_command_
tw_
unique_bits_
unique_chars_
unlink
unpack_system_code_
unwinder_
user_info_
user_info_
where
who
write_list_