

Subroutine Call  
1/31/73

Name: get\_ring\_

This subroutine returns to the caller the number of the protection ring in which he is executing. For a discussion of rings see the MPM Subsystem Writers Guide section, Interprocess Access Control (Rings).

Usage

```
declare get_ring_ entry returns (fixed bin(6));
```

```
ring_no = get_ring_();
```

- 1) ring\_no is the number of the ring in which the caller is executing. (Output)



Name: get\_system\_free\_area\_

This procedure returns a pointer to the system free area for the ring in which it was called. (namely system\_free\_k where k is the ring in which it was called). Allocations by system programs should be performed in this area.

Usage

```
declare get_system_free_area_ entry returns (ptr);
```

```
area_ptr = get_system_free_area_ ();
```

1) area\_ptr is a pointer to the system free area. (Output)



Subroutine Call  
Development System  
6/29/72

Name: get\_to\_cl\_

This procedure is called to re-establish the standard environment after a quit or unclaimed signal. It is the default procedure called by cu\_\$cl. (See the MPM write-up for cu\_.)

Entry: get\_to\_cl\_\$unclaimed\_signal

This entry is called by the standard system default handler when a quit or unclaimed signal occurs. It throws away any read-ahead data on the stream "user input". It saves the attachments of the I/O streams "user\_input", "user\_output", and "error\_output" and restores them to their standard attachment, namely "user\_i/o". It saves the mode of "user\_i/o" and restores it to the default mode. It reestablishes the standard default condition handler. It then calls listen\_\$release\_stack. If control returns, this means a start command has been typed, so the entry restores the attachments of "user\_input", "user\_output" and "error\_output" and the mode of "user\_i/o" to what they were at the time of the quit (unless the argument passed to listen\_\$release\_stack indicates that it should not), issues a start order call, and returns to its caller.

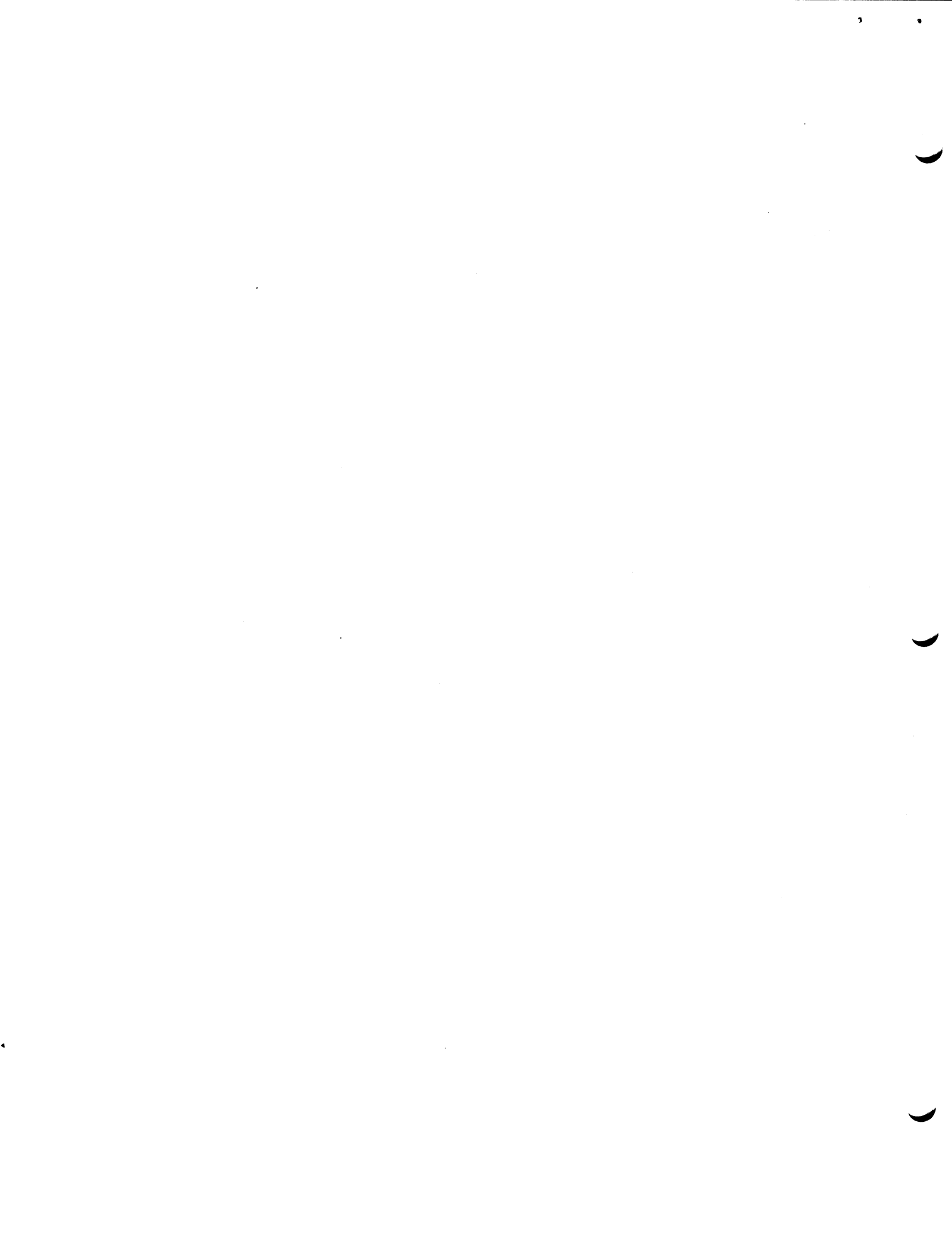
This entry should be called only via cu\_\$cl.

### Usage

```
declare get_to_cl_$unclaimed_signal entry;
```

```
call get_to_cl_$unclaimed_signal;
```

There are no arguments.



Subroutine Call  
2/27/73Name: hcs\_\$add\_dir\_inacl\_entries

This subroutine, given a list of Initial Access Control List (Initial ACL) entries, will add the given Initial ACL entries, or change their directory modes if a corresponding entry already exists, to the Initial ACL for new directories within the specified directory.

Usage

```
declare hcs_$add_dir_inacl_entries entry (char(*), char(*),
      ptr, fixed bin, fixed bin, fixed bin(35));

call hcs_$add_dir_inacl_entries (dirname, ename, acl_ptr,
      acl_count, ring, code);
```

- 1) `dirname` is the path name of the directory superior to the one in question. (Input)
- 2) `ename` is the entry name of the directory in question. (Input)
- 3) `acl_ptr` points to a user-filled `dir_acl` structure. See Notes below. (Input)
- 4) `acl_count` contains the number of entries in the `dir_acl` structure. See Notes below. (Input)
- 5) `ring` is the ring number of the Initial ACL. (Input)
- 6) `code` is a standard status code. (Output)

Notes

The following structure is used:

```
declare 1 dir_acl (acl_count) aligned based (acl_ptr),
      2 access_name char(32),
      2 dir_modes bit(36),
      2 status_code fixed bin(35);
```

- 1) `access_name` is the access name (in the form `person.project.tag`) which identifies the processes to which this Initial ACL entry applies.
- 2) `dir_modes` contains the directory modes for this access name. The first three bits correspond to the

modes status, modify, and append. The remaining bits must be zero.

3) status\_code is a standard status code for this Initial ACL entry only.

If code is returned as error\_table\_\$argerr then the offending Initial ACL entries in the dir\_acl structure will have status\_code set to an appropriate error and no processing will have been performed.



Subroutine Call  
2/27/73

**Name:** hcs\_\$add\_inacl\_entries

This subroutine, given a list of Initial Access Control List (Initial ACL) entries, will add the given Initial ACL entries, or change their modes if a corresponding entry already exists, to the Initial ACL for new segments within the specified directory.

**Usage**

```
declare hcs_$add_inacl_entries entry (char(*), char(*),
ptr, fixed bin, fixed bin, fixed bin(35));
```

```
call hcs_$add_inacl_entries (dirname, ename, acl_ptr,
acl_count, ring, code);
```

- 1) `dirname` is the superior directory portion of the path name of the directory in question. (Input)
- 2) `ename` is the entry name portion of the path name of the directory in question. (Input)
- 3) `acl_ptr` points to a user-filled `segment_acl` structure. See Notes below. (Input)
- 4) `acl_count` contains the number of Initial ACL entries in the `segment_acl` structure. See Notes below. (Input)
- 5) `ring` is the ring number of the Initial ACL. (Input)
- 6) `code` is a standard status code. (Output)

**Notes**

The following structure is used:

```
dcl 1 segment_acl (acl_count) aligned based (acl_ptr),
2 access_name char(32),
2 modes bit(36),
2 zero_pad bit(36),
2 status_code fixed bin(35);
```

- 1) `access_name` is the access name (in the form `person.project.tag`) which identifies the processes to which this Initial ACL entry applies.
- 2) `modes` contain the modes for this access name. The first three bits correspond to the modes read, execute, and write. The remaining bits must be

zero.

- 3) zero\_pad must contain zero. (This field is for use with extended access.)
- 4) status\_code is a standard status code for this Initial ACL entry only.

If code is returned as error\_table\$argerr then the offending Initial ACL entries in segment\_acl will have status\_code set to an appropriate error and no processing will have been performed.

Subroutine Call  
2/28/73

Name: hcs\_\$delete\_dir\_inacl\_entries

This subroutine is used to delete specified entries from an Initial Access Control List (Initial ACL) for new directories within the specified directory. The delete\_acl structure used by this subroutine is described in the MPM write-up for hcs\_\$delete\_inacl\_entries.

Usage

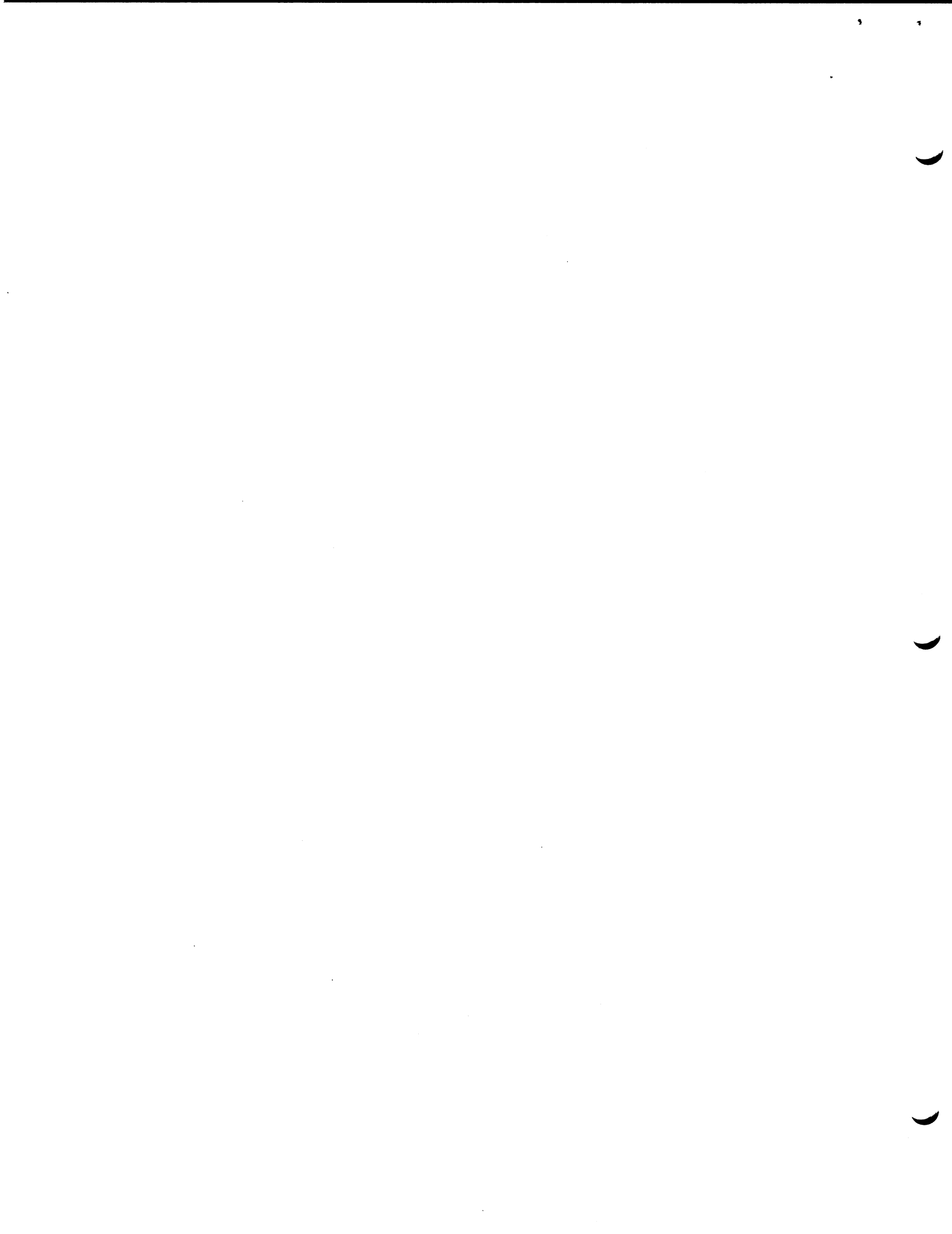
```
declare hcs_$delete_dir_inacl_entries entry (char(*),
      char(*), ptr, fixed bin, fixed bin, fixed bin(35));

call hcs_$delete_dir_inacl_entries (dirname, ename, acl_ptr,
      acl_count, ring, code);
```

- 1) `dirname` is the path name of the directory superior to the one in question. (Input)
- 2) `ename` is the entry name of the directory in question. (Input)
- 3) `acl_ptr` points to a user-filled delete\_acl structure. (Input)
- 4) `acl_count` is the number of Initial ACL entries in the delete\_acl structure. (Input)
- 5) `ring` is the ring number of the Initial ACL. (Input)
- 6) `code` is a standard status code. (Output)

Note

The status code is interpreted as described in hcs\_\$delete\_inacl\_entries.



Subroutine Call  
2/27/73Name: hcs\_\$delete\_inacl\_entries

This subroutine is called to delete specified entries from an Initial Access Control List (Initial ACL) for new segments within the specified directory.

Usage

```
declare hcs_$delete_inacl_entries entry (char(*), char(*),
    ptr, fixed bin, fixed bin, fixed bin(35));
```

```
call hcs_$delete_inacl_entries (dirname, ename, acl_ptr,
    acl_count, ring, code);
```

- 1) `dirname` is the superior directory portion of the path name of the directory in question. (Input)
- 2) `ename` is the entry name portion of the path name of the directory in question. (Input)
- 3) `acl_ptr` points to a user-filled `delete_acl` structure. See Notes below. (Input)
- 4) `acl_count` contains the number of ACL entries in the `delete_acl` structure. See Notes below. (Input)
- 5) `ring` is the ring number of the Initial ACL. (Input)
- 6) `code` is a standard status code. (Output)

Notes

The following structure is used:

```
declare 1 delete_acl (acl_count) aligned based (acl_ptr),
    2 access_name char(32),
    2 status_code fixed bin(35);
```

- 1) `access_name` is the access name (in the form of `person.project.tag`) which identifies the Initial ACL entry to be deleted.
- 2) `status_code` is a standard status code for this Initial ACL entry only.

If `code` is returned as `error_table_$argerr` then the offending Initial ACL entries in the `delete_acl` structure will have `status_code` set to an appropriate error and no processing will have been performed.

If an access name cannot be matched to one existing on the Initial ACL then the status code of that Initial ACL entry is set to `error_table_$user_not_found`, processing continues to the end of the `delete_acl` structure and code is returned as zero.

Subroutine Call  
3/15/73

**Name:** hcs\_\$get\_author

This subroutine returns the author of a segment or a link.

**Usage**

```
declare hcs_$get_author entry (char(*), char(*), fixed
    bin(1), char(*), fixed bin(35));

call hcs_$get_author (dirname, entry, chase, author, code);
```

- 1) **dirname** is the path name of the directory containing entry. The path name can have a maximum length of 168 characters. (Input)
- 2) **entry** is the name of the entry. It can have a maximum length of 32 characters. (Input)
- 3) **chase** if entry refers to a link, this flag indicates whether to return the author of the link or the author of the segment to which the link points:  
  
    0 = return link author;  
    1 = return segment author. (Input)
- 4) **author** is the author of the segment or link in the form of Doe.Student.a with a maximum length of 32 characters. (Output)
- 5) **code** is a standard storage system status code. (Output)

**Note**

The user must have status permission on the parent directory.





Subroutine Call  
3/16/73

Name: hcs\_\$get\_bc\_author

This subroutine returns the bit count author of a segment or directory. The bit count author is the name of the user who last set the bit count of the segment or directory.

Usage

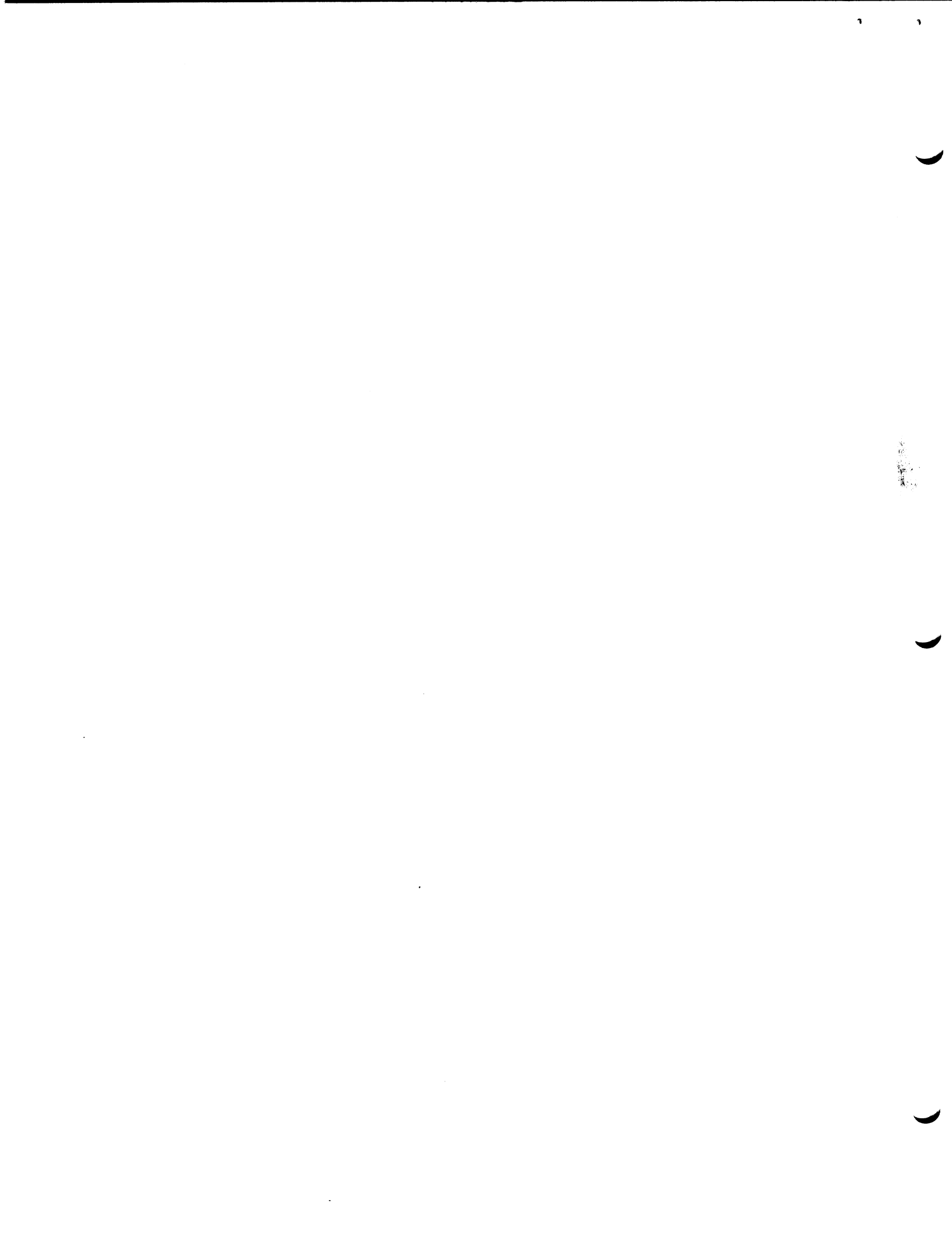
```
declare hcs_$get_bc_author entry (char(*), char(*),  
char(*), fixed bin(35));
```

```
call hcs_$get_bc_author (dirname, ename, bc_author, code);
```

- 1) dirname is the directory name of the segment whose bit count author is wanted. (Input)
- 2) ename is the entry name of the segment whose bit count author is wanted. (Input)
- 3) bc\_author is the bit count author of the segment in the form of Doe.Student.a. (Output)
- 4) code is a standard storage system status code. (Output)

Note

The user must have status permission on the directory containing the segment.



Subroutine Call  
3/1/73

Name: hcs\_\$get\_dir\_ring\_brackets

This subroutine, given the path name of the superior directory and the name of the directory, will return that directory's ring brackets.

Usage

```
declare hcs_$get_dir_ring_brackets entry (char(*), char(*),  
      (2) fixed bin(3), fixed bin(35));
```

```
call hcs_$get_dir_ring_brackets (dirname, ename, drb, code);
```

- 1) `dirname` is the path name of the superior directory. (Input)
- 2) `ename` is the entry name of the directory in question. (Input)
- 3) `drb` is a 2-element array to contain the directory's ring brackets. (Output)
- 4) `code` is a standard status code. (Output)

Notes

The user must have status permission to `dirname` in order to list the directory's ring brackets.

Ring brackets are discussed in the MPM Subsystem Writers' Guide section, Intraprocess Access Control (Rings).



Subroutine Call  
3/16/73

Name: hcs\_\$get\_max\_length

This subroutine returns the max length of a segment given a directory name and entry name. The max length is the length beyond which the segment may not grow.

Usage

```
declare hcs_$get_max_length entry (char(*), char(*),  
fixed bin(18), fixed bin(35));
```

```
call hcs_$get_max_length (dirname, ename, max_length, code);
```

- 1) `dirname` is the directory name of the segment whose max length is wanted. (Input)
- 2) `ename` is the entry name of the segment whose max length is wanted. (Input)
- 3) `max_length` is the max length of the segment in words. (Output)
- 4) `code` is a standard storage system status code. (Output)

Note

The user must have status permission on the directory containing the segment.



Subroutine Call  
4/30/73

Name: hcs\_\$get\_process\_usage

This subroutine returns information about a process's usage of Multics since it was created. It provides data about processor and memory usage.

Usage

```
declare hcs_$get_process_usage entry (ptr, fixed bin(35));  
call hcs_$get_process_usage (info_pointer, code);
```

- 1) info\_pointer is a pointer to the structure in which process information is returned (see Notes below). (Input)
- 2) code is a standard status code. (Output)

Notes

The format of the structure based on info\_pointer is:

```
declare 1 process_usage,  
2 number_wanted fixed bin,  
2 cpu_time_used fixed bin(71),  
2 memory_usage fixed bin(71),  
2 number_of_page_faults fixed bin(35),  
2 amount_of_prepaging fixed bin(35),  
2 process_virtual_time fixed bin(71);
```

- 1) number\_wanted is set by the calling program to specify the number of other entries in the structure to be filled in. The entry itself (the numbers wanted) is not included in this count. The value 5 would cause five entries listed below to be filled in. A smaller number, n, will cause the first n entries to be filled in. (Input)
- 2) cpu\_time\_used is set to the amount of processor time (in microseconds) used by the calling process. (Output)

- 3) memory\_usage is a measure of the primary (core) memory used by this process. The units of memory usage are page-seconds, normalized to account for the size of primary memory actually in use. (Output)
- 4) number\_of\_page\_faults is set to the number of demand page faults this process has taken. (Output)
- 5) amount\_of\_prepaging is the number of pages prepaged for this process. (Output)
- 6) process\_virtual\_time is the amount of processor time (in microseconds) used exclusive of page fault and system interrupt processing time. (Output)



Subroutine Call  
2/27/73

Name: hcs\_\$get\_ring\_brackets

This subroutine, given the directory name and entry name of a nondirectory segment will return that segment's ring brackets.

Usage

```
declare hcs_$get_ring_brackets entry (char(*), char(*),  
    (3) fixed bin(3), fixed bin(35));
```

```
call hcs_$get_ring_brackets (dirname, ename, rb, code);
```

- 1) `dirname` is the directory portion of the path name of the segment in question. (Input)
- 2) `ename` is the entry name of the segment in question. (Input)
- 3) `rb` is a 3-element array to contain the segment ring brackets. (Output)
- 4) `code` is a standard status code. (Output)

Notes

The user must have status permission to `dirname` in order to list a segment's ring brackets.

Ring brackets are discussed in the MPM Subsystem Writers' Guide section, Intraprocess Access Control (Rings).



Subroutine Call  
3/16/73

Name: hcs\_\$get\_safety\_sw

This subroutine returns the safety switch of a directory or a segment, given a directory name and an entry name.

Usage

```
declare hcs_$get_safety_sw entry (char(*), char(*), bit(1),
    fixed bin(35));
```

```
call hcs_$get_safety_sw entry (dirname, ename, safety_sw,
    code);
```

- 1) `dirname` is the directory name of the segment whose safety switch is wanted. (Input)
- 2) `ename` is the entry name of the segment whose safety switch is wanted. (Input)
- 3) `safety_sw` is the value of the segment's safety switch.  
= "0"b if the segment may be deleted.  
= "1"b if the segment may not be deleted. (Output)
- 4) `code` is a standard storage system status code. (Output)

Note

The user must have status permission with respect to the directory containing the segment.



Subroutine Call  
Development System  
06/14/71

Name: hcs\_\$get\_search\_rules

This entry returns the search rules currently in use in the caller's process.

Usage

```
declare hcs_$get_search_rules entry (ptr);  
call hcs_$get_search_rules (search_rules_ptr);
```

- 1) search\_rules\_ptr is a pointer to a user supplied search rules structure. (Input)

Notes

The search rule structure is declared as follows:

```
declare 1 search_rules,  
        2 number fixed bin,  
        2 names (21) char(168) aligned;
```

- 1) number is the number of search rules.  
2) names are the names of the search rules.



Subroutine Call  
Development System  
12/15/71

Name: hcs\_\$initiate\_search\_rules

This is a supervisor entry which is mainly used by the set\_search\_rules and set\_search\_dirs commands. It also provides the user with a means of specifying the search rules which he wishes to use in his process. (For more information on search rules, see the appropriate MPM Reference Guide Section.)

Usage

```
declare hcs_$initiate_search_rules entry (ptr, fixed bin);
```

```
call hcs_$initiate_search_rules (search_rule_pointer,  
code);
```

- 1) search\_rule\_pointer is a pointer to a structure containing the new search rules. (Input)
- 2) code is a standard return status code. (Output)

Notes

The structure pointed to by search\_rule\_pointer is declared as follows:

```
declare 1 sr aligned,  
2 num fixed bin,  
2 names (21) char(168) aligned;
```

- 1) num is the number of entries. The current maximum is 21 but the user need only disclose the maximum that he will use.
- 2) names are the names of the search rules. They may be absolute pathnames or key words.

Search rules may be either absolute pathnames of directories or key words. The allowed search rules are:

pathname the absolute pathname of a directory to be searched;

(key words)

initiated\_segments search for the already initiated segment;

Page 2

referencing_dir	search the parent directory of the module making the reference;
working_dir	search the working directory;
process_dir	search the process directory;
home_dir	search the login or home directory;
default	return to the default search rules;
system_libraries	insert the default system libraries at this point in the search rules;
set_search_directories	insert the following directories after working_dir in the default search rules and make the result the current search rules.

The key word "default" cannot be used with any other code word or pathname as it returns the default rules and exits immediately.

The search rules can be changed when the procedure is called with different rules or the process is terminated.

Errors returned from this routine are:

error\_table\_\$bad\_string (not a pathname or code word)  
error\_table\_\$notadir  
error\_table\_\$too\_many\_sr

Additional file system errors may be returned from other routines which are called from hcs\_\$initiate\_search\_rules.



Subroutine Call  
2/27/73

Name: hcs\_\$list\_dir\_inacl

This subroutine is used to either list the entire Initial Access Control List (Initial ACL) for new directories within the specified directory, or to return the access modes for specified entries. The dir\_acl structure described in hcs\_\$add\_dir\_inacl\_entries is used by this subroutine.

Usage

```
declare hcs_$list_dir_inacl entry (char(*), char(*), ptr,
    ptr, ptr, fixed bin, fixed bin, fixed bin(35));
```

```
call hcs_$list_dir_inacl (dirname, ename, area_ptr,
    area_ret_ptr, acl_ptr, acl_count, ring, code);
```

- 1) dirname is the path name of the directory superior to the one in question. (Input)
- 2) ename is the entry name of the directory in question. (Input)
- 3) area\_ptr points to an area into which the list of Initial ACL entries is to be allocated. (Input)
- 4) area\_ret\_ptr points to the start of the list of the Initial ACL entries. (Output)
- 5) acl\_ptr if area\_ptr is null then acl\_ptr is assumed to point to an Initial ACL structure, dir\_acl, into which mode information is to be placed for the access names specified in that same structure. (Input)
- 6) acl\_count is either the number of entries in the Initial ACL structure identified by acl\_ptr (Input); or if area\_ptr is not null, then it is set to the number of entries in the dir\_acl structure that has been allocated. (Output)
- 7) ring is the ring number of the Initial ACL. (Input)
- 8) code is a standard status code. (Output)

Page 2

Note

If `acl_ptr` is used to obtain modes for specified access names (rather than obtaining modes for all access names on the Initial ACL), then each Initial ACL entry will either have a zero `status_code` and will contain the directory's mode or will have `status_code` set to `error_table_$user_not_found` and will contain a zero mode.

Subroutine Call  
2/27/73

Name: hcs\_\$list\_inacl

This subroutine is used to either list the entire Initial Access Control List (Initial ACL) for new segments within the specified directory, or to return the access modes from specified entries. The segment\_acl structure used by this subroutine is described in the MPM write-up for hcs\_\$add\_inacl\_entries.

Usage

```
declare hcs_$list_inacl entry(char(*), char(*), ptr, ptr,  
ptr, fixed bin, fixed bin, fixed bin(35));
```

```
call hcs_$list_inacl (dirname, ename, area_ptr, area_ret_ptr,  
acl_ptr, acl_count, ring, code)
```

- 1) dirname is the superior directory portion of the path name of the directory in question. (Input)
- 2) ename is the entry name portion of the path name of the directory in question. (Input)
- 3) area\_ptr points to an area into which the list of Initial ACL entries is to be allocated. (Input)
- 4) area\_ret\_ptr points to the start of the allocated list of Initial ACL entries. (Output)
- 5) acl\_ptr if area\_ptr is null then acl\_ptr is assumed to point to an Initial ACL structure, segment\_acl, into which mode information is to be placed for the access names specified in that same structure. (Input)
- 6) acl\_count is the number of entries in the Initial ACL structure identified by acl\_ptr (Input); or is set to the number of entries in the segment\_acl structure allocated in the area pointed to by area\_ptr, if area\_ptr is not null. (Output)
- 7) ring is the ring number of the Initial ACL. (Input)
- 8) code is a standard status code. (Output)

Page 2

Note

If `acl_ptr` is used to obtain modes for specified access names (rather than obtaining modes for all access names on the Initial ACL), then each Initial ACL entry will either have a zero `status_code` and will contain the segment's mode or will have `status_code` set to `error_table_$user_not_found` and will contain a zero mode.

Subroutine Call  
3/19/73

Name: hcs\_\$quota\_get

This subroutine returns the record quota and accounting information for a directory.

Usage

```
declare hcs_$quota_get entry (char(*), fixed bin(18),
    fixed bin(35), bit(36) aligned, fixed bin, fixed
    bin(1), fixed bin, fixed bin(35));
```

```
call hcs_$quota_get (dirname, quota, trp, tup, infqcnt,
    taccsw, used, code);
```

- 1) `dirname` is the path name of the directory for which quota information is desired. (Input)
- 2) `quota` is the record quota in the directory. (Output)
- 3) `trp` is the time-record product charged to the directory. This number is in units of record-seconds. (Output)
- 4) `tup` is the time that the `trp` was last updated in storage system time format (the high-order 36 bits of the 52-bit time returned by `clock_`). (Output)
- 5) `infqcnt` is the number of immediately inferior directories (i.e., directories in this directory) which contain terminal accounts. (Output)
- 6) `taccsw` is the terminal account switch. If the switch is on, the records are charged against the quota in this directory. If the switch is off, the records are charged against the quota in the first superior directory with a terminal account. (Output)
- 7) `used` is the number of records used by segments in this directory and by non-terminal inferior directories. (Output)
- 8) `code` is a standard storage system status code. (Output)

Page 2

Notes

The user must have status permission on the directory.

If the account is currently active, this call will cause the account information in the directory header to be updated from the Active Segment Table (AST) entry before this information is returned to the caller. If the directory contains a non-terminal account, the quota, trp, and tup variables are all zero. The variable used, however, is kept up-to-date and represents the number of pages of segments in this directory and inferior non-terminal directories. If a quota were to be placed in this directory, it should be greater than this used value.

Subroutine Call  
3/19/73

Name: hcs\_\$quota\_move

This subroutine is callable by any user and moves all or part of a quota between two directories, one of which is immediately inferior to the other.

Usage

```
declare hcs_$quota_move entry (char(*), char(*),  
    fixed bin(18), fixed bin(35));
```

```
call hcs_$quota_move (dirname, entry, quota_change, code);
```

- 1) `dirname` is the path name of the parent directory.  
(Input)
- 2) `entry` is the entry name of the inferior directory.  
(Input)
- 3) `quota_change` is the number of 1024-word pages of secondary storage quota to be subtracted from the parent directory and added to the inferior directory. (Input)
- 4) `code` is a standard storage system status code.  
(Output)

Notes

The entry specified by `entry` must be a directory.

The user must have modify permission in both directories.

After the quota change, the remaining quota in each directory must be greater than the number of pages used in that directory.

The argument `quota_change` may be either a positive or negative number. If it is positive, the quota will be moved from `dirname` to `entry`. If it is negative, the move will be from `entry` to `dirname`. If the change results in zero quota left on `entry`, that directory is assumed to no longer contain a terminal quota and all of its used pages are reflected up to the used pages on `dirname`. There is a restriction on quotas such that all quotas in the chain from the root to (but not including) the terminal directory must be nonzero. This restriction means that `hcs_$quota_move` cannot leave behind a quota of zero in the superior directory.





Subroutine Call  
3/1/73**Name:** hcs\_\$replace\_dir\_inacl

This subroutine replaces an entire Initial Access Control List (Initial ACL) for new directories within a specified directory with a user-provided Initial ACL, and can optionally add an entry for \*.SysDaemon.\* with mode sma to the new Initial ACL. The dir\_acl structure described in hcs\_\$add\_dir\_inacl\_entries is used by this subroutine.

**Usage**

```
declare hcs_$replace_dir_inacl entry (char(*), char(*), ptr,
    fixed bin, bit(1) aligned, fixed bin,
    fixed bin(35));
```

```
call hcs_$replace_dir_inacl (dirname, ename, acl_ptr,
    acl_count, no_sysdaemon_sw, ring, code);
```

- 1) **dirname** is the path name of the directory superior to the one in question. (Input)
- 2) **ename** is the entry name of the directory in question. (Input)
- 3) **acl\_ptr** points to a user-supplied dir\_acl structure that is to replace the current Initial ACL. (Input)
- 4) **acl\_count** is the number of entries in the dir\_acl structure. (Input)
- 5) **no\_sysdaemon\_sw** if "0"b, then a \*.SysDaemon.\* sma entry will be put on the Initial ACL after the existing Initial ACL has been deleted and before the user-supplied dir\_acl entries are added; if "1"b, then only the user-supplied dir\_acl will replace the existing Initial ACL. (Input)
- 6) **ring** is the ring number of the Initial ACL. (Input)
- 7) **code** is a standard status code. (Output)

Page 2

Note

If `acl_count` is zero then the existing Initial ACL will be deleted and only the action indicated by `no_sysdaemon_sw` will be performed (if any). In the case when `acl_count` is greater than zero, processing of the `dir_acl` entries is performed top to bottom, allowing later entries to overwrite previous ones if the `access_name` parts are identical.

Subroutine Call  
3/1/73**Name:** hcs\_\$replace\_inacl

This subroutine replaces an entire Initial Access Control List (Initial ACL) for new segments within a specified directory with a user-provided Initial ACL, and can optionally add an entry for \*.SysDaemon.\* with mode rw to the new Initial ACL. The segment\_acl structure described in hcs\_\$add\_inacl\_entries is used by this subroutine.

**Usage**

```
declare hcs_$replace_inacl entry (char(*), char(*), ptr,  
    fixed bin, bit(1), fixed bin, fixed bin(35));
```

```
call hcs_$replace_inacl (dirname, ename, acl_ptr, acl_count,  
    no_sysdaemon_sw, ring, code);
```

- 1) **dirname** is the superior directory portion of the path name of the directory in question. (Input)
- 2) **ename** is the entry name portion of the path name of the directory in question. (Input)
- 3) **acl\_ptr** points to the user supplied segment\_acl structure that is to replace the current Initial ACL. (Input)
- 4) **acl\_count** is the number of entries in the segment\_acl structure. (Input)
- 5) **no\_sysdaemon\_sw** if "0"b, then a \*.SysDaemon.\* rw entry will be put on the Initial ACL after the existing Initial ACL has been deleted and before the user-supplied segment\_acl entries are added; if "1"b, then only the user-supplied segment\_acl will replace the existing Initial ACL. (Input)
- 6) **ring** is the ring number of the Initial ACL. (Input)
- 7) **code** is a standard status code. (Output)

Note

If `acl_count` is zero then the existing Initial ACL will be deleted and only the action indicated by `no_sysdaemon_sw` will be performed (if any). In the case when `acl_count` is greater than zero, processing of the `segment_acl` entries is performed top to bottom, allowing later entries to overwrite previous ones if the `access_name` parts are identical.

Subroutine Call  
Development System  
05/10/71

Name: hcs\_\$reset\_working\_set

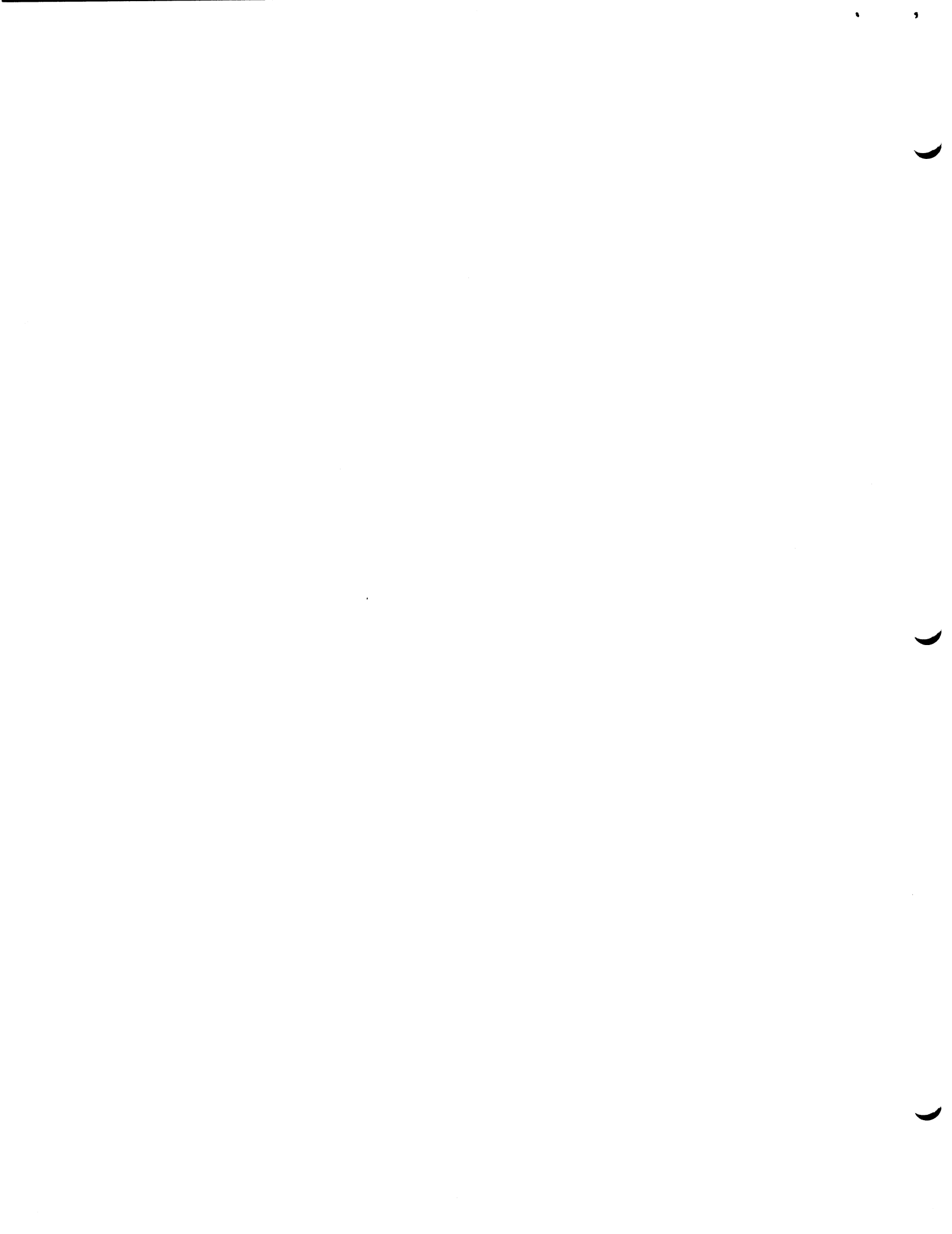
This entry is called to turn off the used bits of all pages in the page-trace list for the current process. This is equivalent to truncating the pre-page list and starting the gathering of pre-page statistics with the next page fault.

Usage

```
declare hcs_$reset_working_set entry;
```

```
call hcs_$reset_working_set;
```

There are no arguments.



Subroutine Call  
3/1/73

**Name:** hcs\_\$set\_dir\_ring\_brackets

This subroutine, given the path name of the superior directory and the name of the directory, will set that directory's ring brackets.

**Usage**

```
declare hcs_$set_dir_ring_brackets entry (char*), char(*),  
      (2) fixed bin(3), fixed bin(35));
```

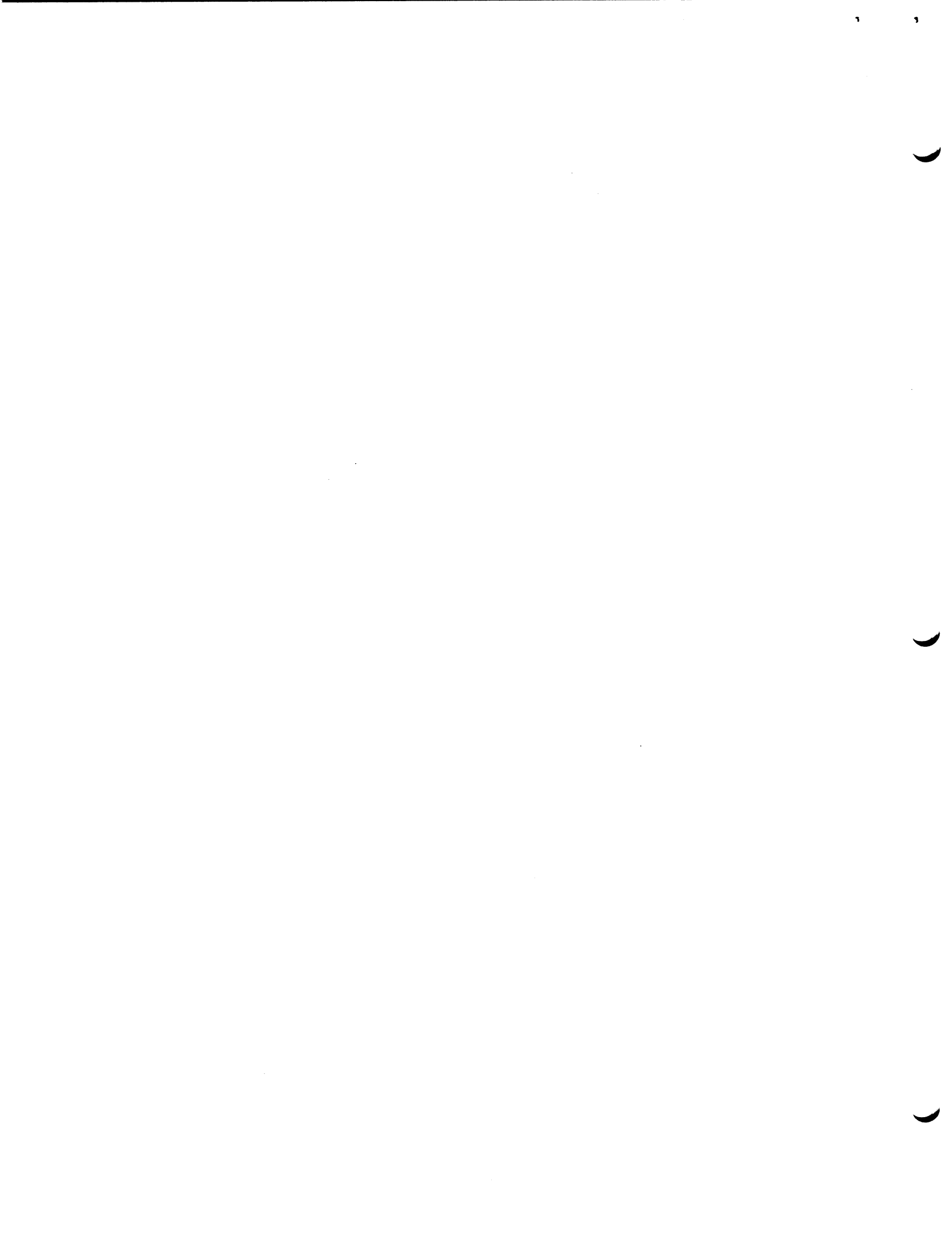
```
call hcs_$set_dir_ring_brackets (dirname, ename, drb, code);
```

- 1) **dirname** is the path name of the superior directory. (Input)
- 2) **ename** is the entry name of the directory in question. (Input)
- 3) **drb** is a 2-element array specifying the ring brackets of the directory. (Input)
- 4) **code** is a standard status code. (Output)

**Notes**

The user must have modify permission in the superior directory and the validation level must be less than or equal to both the present value of the first ring bracket and the new value of the first ring bracket that the user wishes set.

Ring brackets and validation levels are discussed in the MPM Subsystem Writers' Guide section, Intraprocess Access Control (Rings).





Subroutine Call  
3/30/73

Name: hcs\_\$set\_max\_length

This subroutine sets the max length of a segment, given a directory name and an entry name. The max length is the length beyond which the segment may not grow.

Usage

```
declare hcs_$set_max_length entry (char(*), char(*),  
    fixed bin(18), fixed bin(35));
```

```
call hcs_$set_max_length (dirname, ename, max_length, code);
```

- 1) `dirname` is the directory name of the segment whose max length is to be changed. (Input)
- 2) `ename` is the entry name of the segment whose max length is to be changed. (Input)
- 3) `max_length` is the new value in words for the max length of the segment. (Input)
- 4) `code` is a standard storage system status code. (See Notes below.) (Output)

Notes

A directory may not have its max length changed.

Modify permission with respect to the directory containing the segment is required.

Eventually, the max length of a segment will be accurate to units of 16 words, and if `max_length` is not a multiple of 16 words, it will be set to the next multiple of 16 words. However, currently the max length of a segment should be set in units of 1024 words, due to hardware restrictions.

If an attempt is made to set the max length of a segment greater than the system maximum, `sys_info$max_seg_size`, code will be set to `error_table_$argerr`.

If an attempt is made to set the max length of a segment greater than its current length, code will be set to `error_table_$invalid_max_length`.

The subroutine `hcs_$set_max_length_seg` may be used when the pointer to the segment is given, rather than a path name.



Subroutine Call  
3/30/73

Name: hcs\_\$set\_max\_length\_seg

This subroutine sets the max length of a segment, given the pointer to the segment. The max length is the length beyond which the segment may not grow.

Usage

```
declare hcs_$set_max_length_seg entry (ptr, fixed bin(18),  
fixed bin(35));
```

```
call hcs_$set_max_length_seg (segptr, max_length, code);
```

- 1) segptr is the pointer to the segment whose max length is to be changed. (Input)
- 2) max\_length is the new value in words for the max length of the segment. (Input)
- 3) code is a standard storage system status code. (see Notes below.) (Output)

Notes

A directory may not have its max length changed.

Modify permission with respect to the directory containing the segment is required.

Eventually, the max length of a segment will be accurate to units of 16 words, and if max\_length is not a multiple of 16 words, it will be set to the next multiple of 16 words. However, currently the max length of a segment should be set in units of 1024 words, due to hardware restrictions.

If an attempt is made to set the max length of a segment to greater than the system maximum, sys\_info\$max\_seg\_size, code will be set to error\_table\_\$argerr.

If an attempt is made to set the max length of a segment to less than its current length, code will be set to error\_table\_\$invalid\_max\_length.

The subroutine hcs\_\$set\_max\_length may be used when a path name of the segment is given, rather than the pointer.



Subroutine Call  
3/1/73

**Name:** hcs\_\$set\_ring\_brackets

This subroutine, given the directory name and entry name of a nondirectory segment, sets that segment's ring brackets.

**Usage**

```
declare hcs_$set_ring_brackets entry (char(*), char(*),  
    (3) fixed bin(3), fixed bin(35));
```

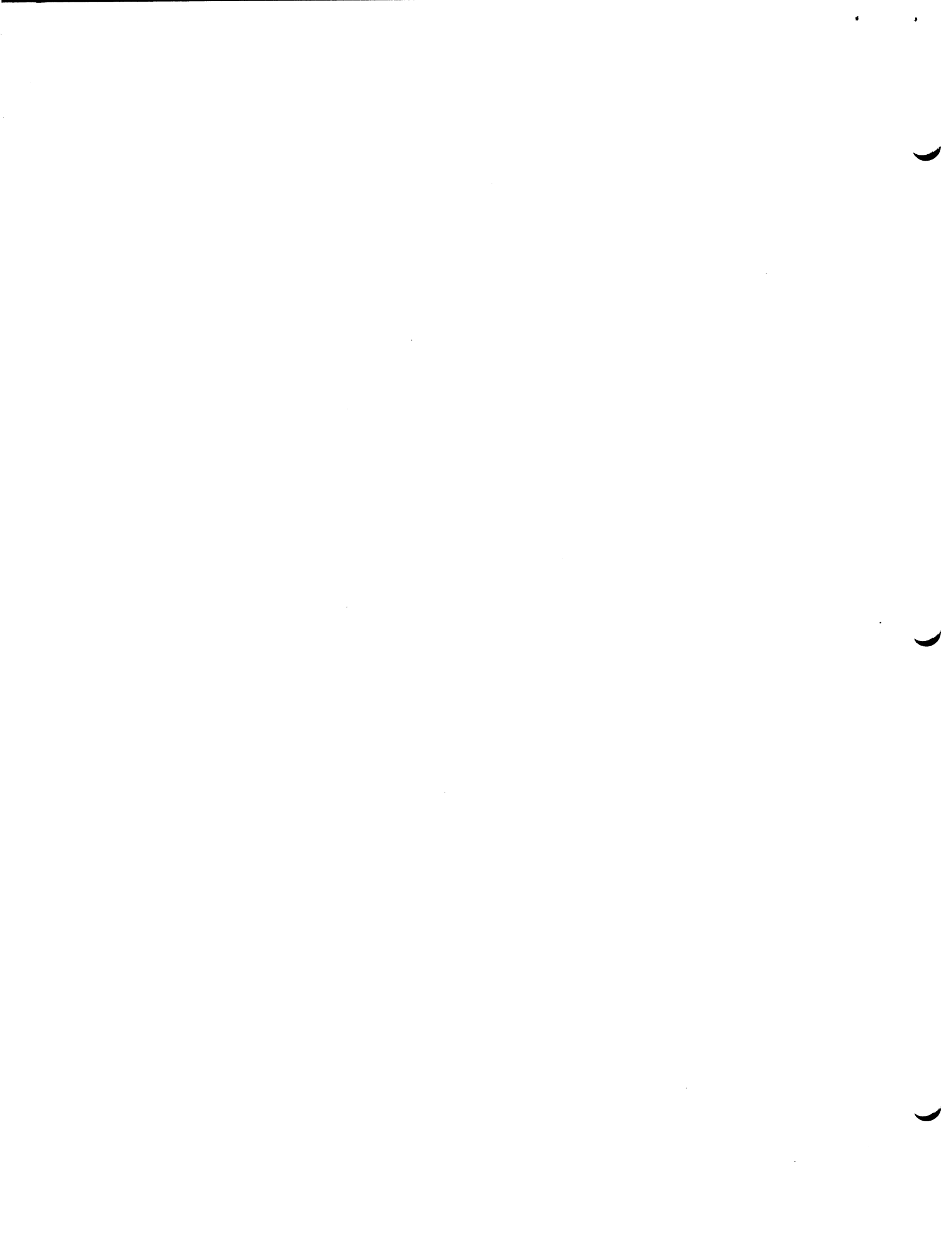
```
call hcs_$set_ring_brackets (dirname, ename, rb, code);
```

- 1) **dirname** is the directory portion of the path name of the segment in question. (Input)
- 2) **ename** is the entry name of the segment in question. (Input)
- 3) **rb** is a 3-element array specifying the ring brackets of the segment. (Input)
- 4) **code** is a standard status code. (Output)

**Notes**

The user must have modify permission to the directory and the validation level must be less than or equal to both the present value of the first ring bracket and the new value for the first ring bracket that the user wishes set.

Ring brackets and validation levels are discussed in the MPM Subsystem Writers' Guide section, Intraprocess Access Control (Rings).



Subroutine Call  
3/16/73

**Name:** hcs\_\$set\_safety\_sw

This subroutine allows the safety switch associated with a segment to be changed. The segment is designated by a directory name and an entry name. See the MPM Reference Guide section, Segment, Directory and Link Attributes, for a description of the safety switch.

**Usage**

```
declare hcs_$set_safety_sw entry (char(*), char(*),  
                                bit(1), fixed bin(35));
```

```
call hcs_$set_safety_sw (directory, entry, safety_sw, code);
```

- 1) directory is the name of the directory containing the segment whose safety switch is to be changed. (Input)
- 2) entry is the entry name of the segment whose safety switch is to be changed. (Input)
- 3) safety\_sw is the new value of the safety switch:  
= "0"b if the segment may be deleted.  
= "1"b if the segment may not be deleted. (Input)
- 4) code is a standard storage system status code. (Output)

**Note**

hcs\_\$set\_safety\_sw\_seg performs the same function when the pointer to the segment is provided rather than a pathname.





Subroutine Call  
3/15/73

Name: hcs\_\$set\_safety\_sw\_seg

This subroutine sets the safety switch of a segment, given the pointer to the segment. The safety switch of a segment is a protection against deletion.

Usage

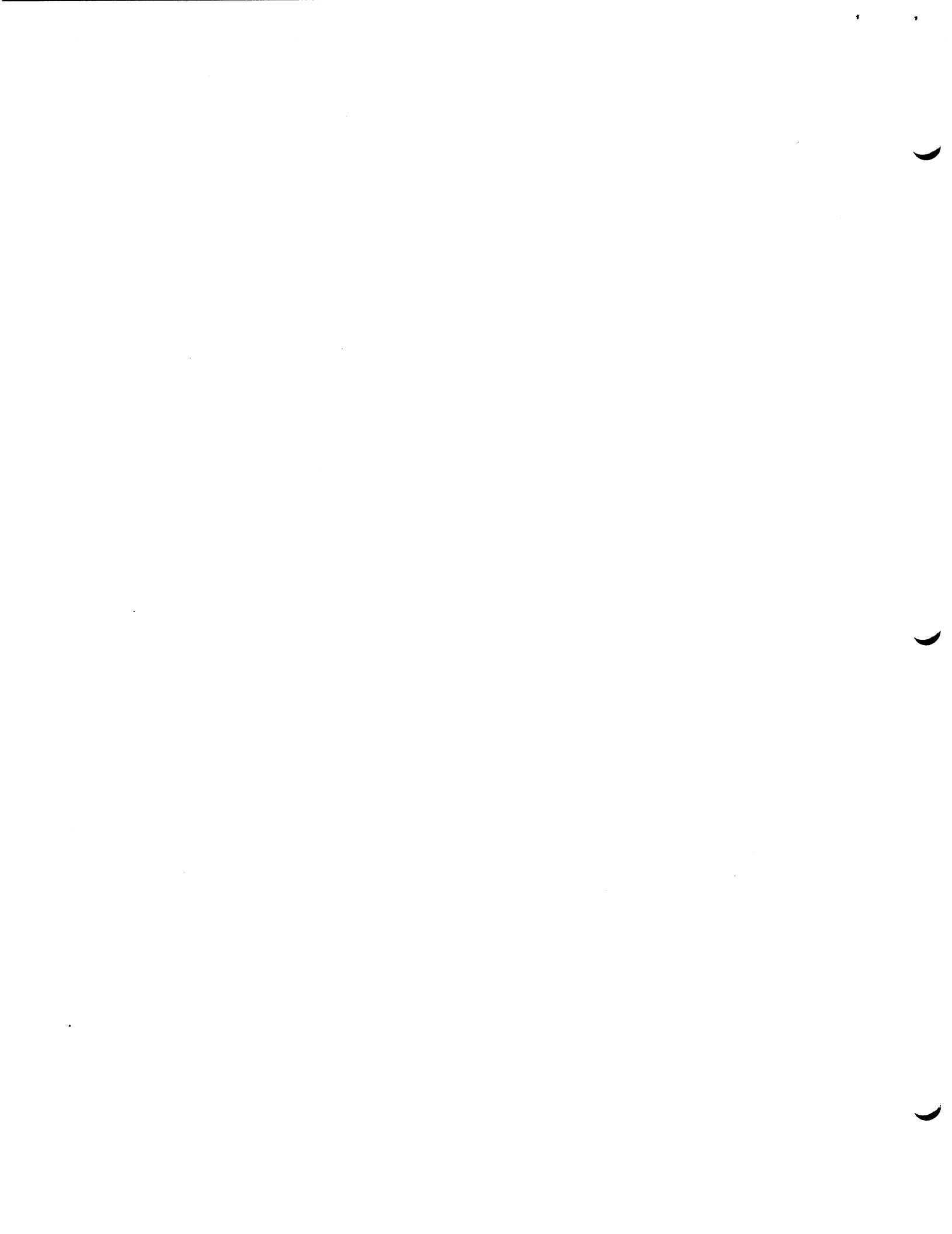
```
declare hcs_$set_safety_sw_seg (ptr, bit(1), fixed bin(35));  
call hcs_$set_safety_sw_seg (segptr, safety_sw, code);
```

- 1) segptr is the pointer to the segment whose safety switch is to be changed. (Input)
- 2) safety\_sw is the new value of the safety switch:  
= "0"b if the segment may be deleted.  
= "1"b if the segment may not be deleted. (Input)
- 3) code is a standard storage system status code. (Output)

Notes

The user must have modify permission with respect to the directory containing the segment whose safety switch is to be changed.

The subroutine hcs\_\$set\_safety\_sw performs the same function when provided with a path name of the segment rather than the pointer.



Subroutine Call  
2/7/73

Name: hcs\_\$wakeup

This entry sends an interprocess communication wakeup signal to a specified process over a specified event channel. If that process had previously called ipc\_\$block, it would be wakened. See the MPM write-up for ipc\_.

Usage

```
declare hcs_$wakeup entry (bit(36), fixed bin(71),
    fixed bin(71), fixed bin(35));
```

```
call hcs_$wakeup (process_id, channel_id, message, code);
```

- 1) process\_id is the process identifier of the target process. (Input)
- 2) channel\_id is the identifier of the event channel over which the wakeup is to be sent. (Input)
- 3) message is the event message to be interpreted by the target process. (Input)
- 4) code is a status code. It may be either error\_table\_\$invalid\_channel or one of four other values:
  - 0 no error;
  - 1 signalling was correctly done but the target process was in the stopped state;
  - 2 an input argument was incorrect so signalling was aborted;
  - 3 the target process was not found, (e.g., process\_id was incorrect or the target process has been destroyed), so signalling was aborted. (Output)



Subroutine Call  
Development System  
9/9/71

Name: ioa\_

This procedure is used to format character strings, fixed binary numbers, floating numbers, and pointers into complete character string form. The type of formatting to be performed is specified by the use of a control string. The single entry point described here has been designed to provide a more general interface to ioa\_ than has previously been available.

A description of the other entry points listed below and of control strings can be found in the MPM.

```
ioa_
ioa_$ioa_stream
ioa_$ioa_stream_nnl
ioa_$nnl
ioa_$rs
ioa_$rsnnl
ioa_$rsnp
ioa_$rsnpnnl
```

Entry: ioa\_\$general\_rs

This entry point is used to provide ioa\_ with a control string and format arguments taken from a previously created argument list to which a pointer has been obtained.

### Usage

```
declare ioa_$general_rs entry (ptr, fixed bin, fixed bin,
char(*), fixed bin, bit(1) aligned, bit(1) aligned);
```

```
call ioa_$general_rs (arglist_ptr, cs_argno, ff_argno,
retstring, len, padsw, nlsw);
```

- 1) arglist\_ptr is a pointer to the argument list from which the control string and format arguments are to be taken. (Input)
- 2) cs\_argno is the argument number of the control string in the argument list pointed to by arglist\_ptr. (Input)
- 3) ff\_argno is the argument number of the first format argument in the argument list pointed to by arglist\_ptr. (Input)

- 4) retstring            contains the formatted string. It should be large enough to allow for expansion. (Output)
- 5) len                specifies the number of significant characters in retstring. (Output)
- 6) padsw             if zero, the formatted string is not padded; if one, it is padded with blanks on the right. (Output)
- 7) nlsw              if zero, a "new line" is not appended; if one, a "new line" is appended to the formatted string. (Output)

Subroutine Call  
5/25/73Name: ipc\_

The Multics system supports a facility providing for communication between processes. For a thorough understanding (on a conceptual level) of interprocess communication, chapter 7 of E.I. Organick's book on Multics is recommended.\* The basic purpose of the interprocess communication facility is to provide control communication (by means of stop and go signals) between processes. A very primitive communication path is provided through which messages may be sent and waited for.

The subroutine ipc\_ is the user's interface to the Multics interprocess communication facility. Briefly, that facility works as follows. A process may establish event channels (which may be thought of as numbered slots in the facility's tables) in the current protection ring (for a discussion of rings see MPM Subsystem Writer's Guide section, Intraprocess Access Control (Rings)) and may go blocked waiting for an event on one or more channels. An event channel may be either an event-wait channel or an event-call channel. An event-wait channel is used to receive events that are merely marked as having occurred, and to wake up the process if it is blocked waiting for an event on that channel. An event-call channel is one on which the occurrence of an event causes a specified procedure to be called if (or when) the process is blocked waiting for an event on that channel. Naturally the specific event channel must be made known to the process which expected to notice the event. For an event to be noticed by an explicitly cooperating process, the event channel ID value is typically placed in a known location of a shared segment. For an event to be noticed by a system module, a subroutine call is typically made to the appropriate system module. A process may go blocked waiting for an event to occur, or may explicitly check to see if it has occurred. If an event occurs before the target process goes blocked, when it does go blocked it is immediately awakened.

The user may operate on an event channel only if his ring of execution is the same as his ring when the event channel was created.

The subroutine hcs\_\$wakeup (used to wake up a blocked process for a specified event) is described in an MPM Subsystem Writers' Guide subroutine write-up.

---

\*E.I. Organick, The Multics System: An Examination of its Structure. Cambridge, Mass., M.I.T. Press, 1972.

Page 2

Entry: ipc\_\$create\_ev\_chn

This entry creates an event-wait channel in the current ring.

Usage

```
declare ipc_$create_ev_chn entry (fixed bin(71),
    fixed bin(35));
```

```
call ipc_$create_ev_chn (channel_id, code);
```

- 1) channel\_id is the identifier of the event channel. (Output)
- 2) code is a standard status code; see Status Code Values below. (Output)

Entry: ipc\_\$delete\_ev\_chn

This entry destroys an event channel previously created by the process.

Usage

```
declare ipc_$delete_ev_chn entry (fixed bin(71),
    fixed bin(35));
```

```
call ipc_$delete_ev_chn (channel_id, code);
```

- 1) channel\_id is as above. (Input)
- 2) code is as above. (Output)

Entry: ipc\_\$decl\_ev\_call\_chn

This entry changes an event-wait channel into an event-call channel.

Usage

```
declare ipc_$decl_ev_call_chn entry (fixed bin(71), ptr,
    ptr, fixed bin, fixed bin(35));
```

```
call ipc_$decl_ev_call_chn (channel_id, procedure_ptr,
    data_ptr, priority, code);
```

- 1) channel\_id is as above. (Input)



- 2) `procedure_ptr` is a pointer to a procedure entry point to be invoked when an event occurs on the specified channel. (Input)
- 3) `data_ptr` is a pointer to data to be passed to and interpreted by that procedure entry point. (Input)
- 4) `priority` is a number indicating the priority of this event-call channel as compared to other event-call channels declared by this process for this ring. If, upon interrogating all the appropriate event-call channels, more than one is found to have received an event, the lowest-numbered priority will be honored first, and so on. (Input)
- 5) `code` is as above. (Output)

Entry: `ipc_$decl_ev_wait_chn`

This entry changes an event-call channel into an event-wait channel.

Usage

```
declare ipc_$decl_ev_wait_chn entry (fixed bin(71),
                                     fixed bin(35));
```

```
call ipc_$decl_ev_wait_chn (channel_id, code);
```

- 1) `channel_id` is as above. (Input)
- 2) `code` is as above. (Output)

Entry: `ipc_$drain_chn`

This entry resets an event channel so that any pending events (i.e., events which have been received for that channel) are removed.

Usage

```
declare ipc_$drain_chn entry (fixed bin(71), fixed bin(35));
```

```
call ipc_$drain_chn (channel_id, code);
```

- 1) `channel_id` is as above. (Input)

2) code is as above. (Output)

Entry: ipc\_\$cutoff

This entry inhibits the reading of events on a specified event channel. Any pending events are not affected. More may be received, but will not cause the process to wake up.

Usage

```
declare ipc_$cutoff entry (fixed bin(71), fixed bin(35));
call ipc_$cutoff (channel_id, code);
```

1) channel\_id is as above. (Input)

2) code is as above. (Output)

Entry: ipc\_\$reconnect

This entry enables the reading of events on a specified event channel for which reading had previously been inhibited (using ipc\_\$cutoff). All pending signals, whether received before or during the time reading was inhibited, are henceforth available for reading.

Usage

```
declare ipc_$reconnect entry (fixed bin(71), fixed bin(35));
call ipc_$reconnect (channel_id, code);
```

1) channel\_id is as above. (Input)

2) code is as above. (Output)

Entry: ipc\_\$set\_wait\_prior

This entry causes event-wait channels to be given priority over event-call channels when several channels are being interrogated; e.g., upon return from being blocked waiting on any of a list of channels. Only event channels in the current ring are affected.

Usage

```
declare ipc_$set_wait_prior entry (fixed bin(35));
call ipc_$set_wait_prior (code);
```

1) code is as above. (Output)

Entry: ipc\_\$set\_call\_prior

This entry causes event-call channels to be given priority over event-wait channels when several channels are being interrogated; e.g., upon return from being blocked waiting on any of a list of channels. Only event channels in the current ring are affected.

Usage

```
declare ipc_$set_call_prior entry (fixed bin(35));  
call ipc_$set_call_prior (code);
```

1) code is as above. (Output)

Entry: ipc\_\$mask\_ev\_calls

This entry causes ipc\_\$block (see below) to completely ignore event-call channels occurring in the user's ring at the time of this call.

Usage

```
declare ipc_$mask_ev_calls entry (fixed bin(35));  
call ipc_$mask_ev_calls (code);
```

1) code is as above. (Output)

Entry: ipc\_\$unmask\_ev\_calls

This entry reverses the effect of the entry ipc\_\$mask\_ev\_calls.

Usage

```
declare ipc_$unmask_ev_calls entry (fixed bin(35));  
call ipc_$unmask_ev_calls (code);
```

1) code is as above. (Output)

Entry: ipc\_\$block

This entry blocks the user's process until one or more of a specified list of events has occurred.

Usage

```
declare ipc_$block entry (ptr, ptr, fixed bin(35));
call ipc_$block (wait_list_ptr, info_ptr, code);
```

- 1) wait\_list\_ptr is a pointer to the base of a structure which specifies the channels on which events are being awaited. The structure is:

```
declare 1 wait_list based,
        2 nchan fixed bin,
        2 channel_id (nchan) fixed bin(71);
```

It is a count of the number of channels (nchan) and an array of the identifiers for those channels. (Input)

- 2) info\_ptr is a pointer to the base of a structure into which ipc\_\$block may put information about the event which caused it to return (i.e., which wakened the process). The structure has the declaration:

```
declare 1 event_info,
        2 channel_id fixed bin(71),
        2 message fixed bin(71),
        2 sender bit(36),
        2 origin,
        3 dev_signal bit(18) unaligned,
        3 ring bit(18) unaligned,
        2 channel_index fixed bin;
```

- 1) channel\_id is as above.
- 2) message is an event message as specified to hcs\_\$wakeup.
- 3) sender is the process ID of the sending process
- 4) dev\_signal if "1"b, this event occurred as the result of an I/O interrupt.
- 5) ring is the sender's validation level.
- 6) channel\_index is the index of channel\_id in the wait\_list structure

above. (Input)

3) code is as above. (Output)

Entry: ipc\_\$read\_ev\_chn

This entry reads the information about an event on a specified channel if the event has occurred.

Usage

```
declare ipc_$read_ev_chn entry (fixed bin(71), fixed bin,
                                ptr, fixed bin(35));
```

```
call ipc_$read_ev_chn (channel_id, ev_occurred, info_ptr,
                       code);
```

1) channel\_id is as above. (Input)

2) ev\_occurred if equal to 0, no event occurred on the specified channel; if equal to 1, an event occurred on the channel. (Output)

3) info\_ptr is as above. (Input)

4) code is as above. (Output)

Status Code Values

All of the entries described above return a value from 0 to 5 for the status code argument. The values mean the following:

- |   |   |
|---|---|
| 0 | no error.   |
| 1 | ring violation; e.g., the event channel resides in a ring which is not accessible from the caller's ring. |
| 2 | the table which contains the event channels for a given ring was not found.                               |
| 3 | the specified event channel was not found.  |
| 4 | a logical error in using ipc_ was encountered; e.g., waiting on an event event-call channel.              |
| 5 | a bad argument was passed to ipc_; e.g., a zero-value event channel identifier.                           |

Invoking an Event-Call Procedure

When a process is wakened on an event-call channel, control is immediately passed to the procedure specified by the entry `ipc_$decl_ev_call_chn`. The procedure is called with one argument, a pointer to the following structure:

```
declare 1 event_info based,  
        2 channel_id fixed bin(71),  
        2 message fixed bin(71),  
        2 sender bit(36),  
        2 origin,  
        3 dev_signal bit(18) unaligned,  
        3 ring bit(18) unaligned,  
        2 data_ptr ptr;
```

The first items of the structure are the same as in the information returned to `ipc_$block`. The last item, `data_ptr`, is the second argument to `ipc_$decl_ev_call_chn` and points to further data to be used by the called procedure.

Name: listen\_

The listen\_ procedure (referred to as the listener) is the base procedure for the basic command processing loop. In general, the listener reads command lines from "user\_input", calls the command processor to process each command line, and types a ready message after each command line is processed. It is called after a quit or unclaimed signal. If the first command line after a quit or unclaimed signal does not contain a start or hold command, the listener will automatically unwind the stack and reestablish the previous instance of itself after one command line is processed successfully.

Entry: listen\_

This call is usually issued early in the life of a newly created user process and establishes the base level of the listener and the standard command processing loop.

Usage

```
declare listen_ entry (char(*) varying);
```

```
call listen_ (initial_command_line);
```

- 1) initial\_command\_line is a command line to be executed before the first read call on "user\_input". If it is of zero length, it is ignored.  
(Input)

Entry: listen\_\$release\_stack

This entry is called after a quit or unclaimed signal has been processed. It sets a switch which causes the stack to be released if a hold request is not included in the next command line read. If a start request is typed, then listen\_\$release\_stack returns control to its caller.

Usage

```
declare listen_$release_stack entry (bit(1) aligned);
```

```
call listen_$release_stack (restore_attachments);
```

- 1) restore\_attachments is a flag which tells the caller of listen\_\$release\_stack whether or not it should restore the standard I/O attachments and the mode of user\_i/o to what they were at the time of the fault or quit that caused listen\_\$release\_stack to be invoked. "1"b means restore, "0"b means don't restore. (Output)



Subroutine Call  
4/30/73

Name: lss\_login\_responder\_

This is the login responder for the Limited Service System. It looks for the segment lss\_command\_list\_ in >system\_library\_standard, sets up handlers for conditions, starts the time governor if the ratio in the table is greater than zero, and limits which commands the user may use.

Usage

```
declare lss_login_responder_ entry;
```

```
call lss_login_responder_;
```

There are no arguments.

Entry: limited\_command\_system\_

This login responder is identical to the one above, except it looks for the segment lss\_command\_list\_ in the user's project directory before looking in >system\_library\_standard.

Usage

```
declare limited_command_system_ entry;
```

```
call limited_command_system_;
```

There are no arguments.

Note

The make\_commands command can be used to create the segment, lss\_command\_list\_.



Subroutine Call  
11/20/72Name: msf\_manager\_

The purpose of the msf\_manager\_ subroutine is to provide an easy to use and consistent method for handling files that may require more than one segment for storage. Examples of files that may be too large to be stored in one segment, hereafter referred to as multi-segment files (MSFs), are listings, data used as I/O streams, and APL workspaces. msf\_manager\_ should make MSFs almost as easy to use as single segment files (SSFs) in many applications.

MSFs are composed of one or more components, each the size of a segment, identified by unsigned integers. Any word in an SSF can be specified by a path name and a word number. Any word in an MSF can be specified by a path name, component number, and word number within the component. msf\_manager\_ provides the means for manipulating an MSF: creating components, accessing them, deleting them, truncating the MSF, and controlling access.

In this implementation, an MSF with only the component 0 is stored as an SSF. If components other than zero are present, they are stored as segments with name corresponding to the ASCII representation of their component numbers in a directory with the path name of the MSF.

In order to keep information between calls, msf\_manager\_ stores information about files in file control blocks (FCBs). The user is returned a pointer to a file control block by the open entry, and this pointer is then passed to the other msf\_manager\_ entries. The file is closed, and the FCB freed, by the close entry.

Entry: msf\_manager\_\$open

The open entry creates an FCB. It returns a pointer to it in the fcbp argument. The file need not exist to have an FCB created for it.

Usage

```
declare msf_manager_$open entry (char(*), char(*), ptr,  
    fixed bin(35));
```

```
call msf_manager_$open (dname, ename, fcbp, code);
```

- 1) dname is the path name of the directory containing the MSF. (Input)

Page 2

- 2) `ename` is the entry name of the MSF. (Input)
- 3) `fcbp` is a pointer to the FCB. (Output)
- 4) `code` is a storage system status code. It may have the same values as that returned by `hcs_$status_minf`, with the addition of `error_table_$dirseg`, which is returned when an attempt is made to open a directory.

Entry: `msf_manager_$get_ptr`

The `get_ptr` entry returns a pointer to the specified component in the file. If the component does not exist, it can be created. If the file is an SSF, and a component greater than 0 is requested, this entry will change the SSF to an MSF. This change will not affect a previously returned pointer to component 0.

### Usage

```
declare msf_manager_$get_ptr entry (ptr, fixed bin, bit(1),  
ptr, fixed bin(24), fixed bin(35));
```

```
call msf_manager_$get_ptr (fcbp, component, createsw, segp,  
bc, code);
```

- 1) `fcbp` is a pointer to the FCB. (Input)
- 2) `component` is the number of the component desired. (Input)
- 3) `createsw` is "1"b if a non-existing component should be created. (Input)
- 4) `segp` is a pointer to the specified component in the file, or null (if there is an error). (Output)
- 5) `bc` is the bit count of the component. (Output)
- 6) `code` is a storage system status code, which may have the same values as that returned by `hcs_$make_seg`. (Output)

Entry: msf\_manager\_\$adjust

The adjust entry sets the bit count of, truncates, and terminates the components of an MSF. It is given a maximum component number and a bit count within that component. The bit counts of all components with numbers less than the given component are set to `sys_info$max_seg_size*36`. All components with numbers greater than the given component are deleted. All components which have been initiated are terminated. This entry uses a three bit switch to control its actions.

Usage

```
declare msf_manager_$adjust entry (ptr, fixed bin,  
    fixed bin(24), bit(3), fixed bin(35));
```

```
call msf_manager_$adjust (fcbp, component, bc, switch,  
    code);
```

- 1) fcbp is a pointer to the FCB. (Input)
- 2) component is the component number, as above. (Input)
- 3) bc is the bit count to be placed on the specified component. (Input)
- 4) switch is the 3-bit control switch. If the first bit is "0"b, the setting of bit counts is suppressed. If the second bit is "0"b the truncation of the given component to length "bc" is suppressed. If the third bit is "0"b, the components will not be terminated. (Input)
- 5) code is a storage system status code. (Output)

Entry: msf\_manager\_\$close

This entry frees the FCB. It will terminate all components which the FCB indicates are initiated.

Usage

```
declare msf_manager_$close entry (ptr);
```

```
call msf_manager_$close (fcbp);
```

- 1) fcbp is the pointer to the FCB. (Input)

Entry: msf\_manager\_\$list\_acl

This entry returns the Access Control List (ACL) of the MSF.

Usage

```
declare msf_manager_$list_acl entry (ptr, ptr, fixed bin,  
ptr, fixed bin(35));
```

```
call msf_manager_$list_acl (fcbp, aclp, acl_count, areap,  
code);
```

- 1) fcbp is the pointer to the FCB. (Input)
- 2) aclp is the pointer to the ACL. See the MPM write-up for hcs\_\$acl\_list. (Output)
- 3) acl\_count is the number of entries in the ACL. (Output)
- 4) areap is a pointer to an area in which to put the ACL. (Output)
- 5) code is a storage system status code, which may have the same values as that returned by hcs\_\$acl\_list. (Output)

Entry: msf\_manager\_\$replace\_acl

This entry replaces the ACL of an MSF.

Usage

```
declare msf_manager_$replace_acl entry (ptr, ptr, fixed bin,  
fixed bin(35));
```

```
call msf_manager_$replace_acl (fcbp, aclp, acl_count, code);
```

- 1) fcbp is the pointer to the FCB. (Input)
- 2) aclp is the pointer to the new ACL. See the MPM write-up for hsc\_\$acl\_replace. (Input)
- 3) acl\_count is the number of entries in the ACL. (Input)
- 4) code is a storage system status code. (Output)

Name: nd\_handler\_

This procedure is provided to attempt to resolve the error\_table\_\$namedup error which may be encountered by such commands as copy (when performing the command would result in two entries in a directory having the same name). Given a directory and an entry name, it will first attempt to remove the name. No question is asked first since it is easy to add the name back. If it is successful, both the name which was removed and an alternate name on the segment are printed to inform the user of the name removed. If the entry must be deleted in order to removed the name (i.e., there is only one name on the entry), it will first ask permission. If the user says "yes", it will attempt to delete the entry, setting the access control list (ACL) if necessary. This routine will not delete a directory since in the context of a command such as copy, a directory would normally not be involved.

Usage

```
declare nd_handler_entry (char(*), char(*), char(*),
    fixed bin(35));
```

```
call nd_handler_ (caller, pname, ename, code);
```

- 1) caller is the name of the calling procedure and will precede all messages from nd\_handler\_. (Input)
- 2) pname is the path name of the directory containing the segment which caused the name duplication error. (Input)
- 3) ename is the entry name of the segment causing the name duplication error. (Input)
- 4) code is a error code:
  - =0 if the name is removed;
  - =1 if the name is still there. (Output)

Notes

Assuming that nd\_handler\_ was called by the copy command to remove the name foo, the following messages might appear on the terminal under the circumstances specified.

Page 2

- 1) The name is not the only one on the entry and can be removed:

copy: Name duplication. Old name foo removed from  
pname>zilch

where zilch was an alternate name on foo.

- 2) The entry must be deleted to removed the name:

copy: Name duplication. Do you want to delete the old  
segment foo?

copy: Name duplication. Do you want to unlink the old link  
foo?

In these cases, nd\_handler\_ expects an answer of "yes" or  
"no". Any other response is not acceptable (and the user is  
asked to respond with "yes" or "no").

copy: Name duplication. Directory foo not deleted.

Note that a directory is not deleted.

- 3) The entry cannot be removed even by setting the ACL:

copy: Name duplication. Unable to remove old entry foo.

Entry: nd\_handler\_\$del

This entry is the same as above except that the attempt to  
remove the name is skipped. It can be called whenever deletion  
is known to be necessary, as in certain commands after an attempt  
has already been made to remove the name.

#### Usage

```
declare nd_handler_$del entry (char(*), char(*), char(*),  
fixed bin(35);
```

```
call nd_handler_$del (caller, pname, ename, code);
```

Arguments are as above.



Subroutine Call  
Development System  
1/27/72

Name: set\_lock\_

This procedure is a tool provided to enable processes to execute critical sections of a program with the assurance that no other processes will be executing the same or other associated critical sections of code simultaneously. This is a means by which processes can be prevented from interfering with one another when referencing shared data.

The mutual exclusion of processes is obtained by the use of a caller-supplied lock word. This word should be declared as bit(36) aligned and should be initially set to "0"b (i.e., a word containing zero) indicating the unlocked state. When the program is about to enter a critical section of code, it calls the entry set\_lock\_\$lock. This entry places the unique lock identifier for the process in the lock word if no other process currently has its lock identifier in the lock word. If the lock word does already contain the lock identifier of some other process, then the entry set\_lock\_\$lock waits for that process to unlock the lock word. Since only one process at a time can have its lock identifier in the lock word, that process is assured (subject to the conditions stated below) that it is the only process currently executing the critical section of code. If many critical sections share the same lock word, then only one process may be executing in any of them at a given time. Once the critical section has been completed, the program calls set\_lock\_\$unlock to reset the lock to "0"b.

As stated earlier, this procedure is only a tool for solving the problem of mutual process exclusion and its use is successful only if all those processes executing critical sections of code obey the necessary conventions. These conventions include:

- 1) The set\_lock\_ procedure is the only procedure that modifies the lock word with the exception of the procedure that initializes the lock word to "0"b before any call to set\_lock\_ is made.
- 2) All processes issue calls to the entry set\_lock\_\$lock which result in the lock identifier appearing in the lock word before entering a critical section of code.
- 3) All processes issue a call to the entry set\_lock\_\$unlock which results in the lock word being set to zero after completing execution of a critical section of code.

Entry: set\_lock\_\$lock

This entry will attempt to place the lock identifier of the calling process in the given lock word. If the lock word contains "0"b, then the lock word will be set to the lock identifier of the calling process. If the lock word contains a valid lock identifier of another existing process, then set\_lock\_ will wait for this other process to unlock the lock word. If the other process does not unlock the lock word in a given period of time, set\_lock\_ will return with an indication of its lack of success. If the lock word contains a lock identifier not corresponding to an existing process, the lock word will be overwritten with the calling process' lock identifier and an indication that an overwriting has taken place will be returned; the call is still successful, however. Note though, that having to relock an invalid lock implies either a coding error in the use of locks or that a process having a lock set was unexpectedly terminated. In either case, the data being modified may be in an inconsistent state. If the lock word already contains the lock identifier of the calling process, then set\_lock\_ will not modify the lock word, but will return an indication of the occurrence of this situation. Note that this latter case may or may not indicate a programming error, depending on the programmer's conventions.

Usage

```
declare set_lock_$lock entry (bit(36) aligned, fixed bin,
                               fixed bin);
```

```
call set_lock_$lock (lock_word, wait_time, status);
```

- 1) lock\_word            is the lock word to be locked. (Input)
  - 2) wait\_time           indicates the length of real time, in seconds, which set\_lock\_ should wait for a validly locked lock word to be unlocked before returning unsuccessfully. A value of -1 indicates no time limit. (Input)
  - 3) status               0 indicates that the lock word was successfully locked because the lock word was previously unlocked;
- error\_table\_\$invalid\_lock\_reset indicates that the lock word was successfully locked, but the lock word previously contained an invalid lock identifier that was overwritten;

error\_table\_\$locked\_by\_this\_process indicates that the lock word already contained the lock identifier of the calling process and was not modified;

error\_table\_\$lock\_wait\_time\_exceeded indicates that the lock word contained a valid lock identifier of another process and could not be locked in the given time limit. (Output)

Entry: set\_lock\_\$unlock

This entry attempts to reset a given lock word to "0"b and will be successful if the lock word contained the lock identifier of the calling process.

Usage

declare set\_lock\_\$unlock entry (bit(36) aligned, fixed bin);

call set\_lock\_\$unlock (lock\_word, code);

- 1) lock\_word is the lock word to be reset. (Input)
- 2) code 0 indicates successful unlocking;

error\_table\_\$lock\_not\_locked indicates that the lock was not locked;

error\_table\_\$locked\_by\_other\_process indicates that the lock was not locked by this process and therefore was not unlocked. (Output)



Subroutine Call  
Development System  
2/25/72

Name: standard\_default\_handler\_

This procedure is the default condition handler for the Multics standard user environment. It handles all conditions for which no other handler was established in the given invocation of the user environment. The procedure simply dispatches the signalled conditions to other procedures which handle the specific conditions.

### Usage

This entry is meant to be established as a default condition handler and is therefore only directly invoked by the condition mechanism. It may be established as a default handler by calling `default_handler_$set` as follows:

```
declare standard_default_handler_ entry;  
declare default_handler_$set entry (entry);  
  
call default_handler_$set (standard_default_handler_);
```

See the MPM Reference Guide section, The Multics Condition Mechanism, for a description of default handlers.

Entry: standard\_default\_handler\_\$ignore\_pi

This entry is the same as the `standard_default_handler_` entry except that `program_interrupt` conditions are ignored, i.e., the handler returns and tells the condition mechanism to find another handler for `program_interrupt`. This entry is established as the default handler by invocations of the user environment other than the first invocation in order that the user may return to programs active in previous invocations of the user environment.

### Usage

This entry may be established as a default handler by calling `default_handler_$set` as follows:

```
declare standard_default_handler_$ignore_pi entry;  
declare default_handler_$set entry (entry);  
  
call default_handler_$set  
    (standard_default_handler_$ignore_pi);
```



Subroutine Call  
4/30/73

Name: start\_governor\_

This procedure uses timer\_manager\_ (described in an MPM Reference Guide subroutine write-up) to help limit the user to no more than interval\_length/ratio CPU seconds per interval\_length seconds of real time (where both interval\_length and ratio are supplied by the caller). When called, it sets up a timer if the ratio is positive and then returns. It then receives calls from timer\_manager\_ periodically to check CPU usage, and blocks the process for a short amount of time, if necessary, to stay within the ratio.

Usage

```
declare start_governor_ entry (fixed bin, fixed bin);  
call start_governor_ (ratio, interval_length);
```

- 1) ratio See the above description for the meaning of ratio. (Input)
- 2) interval\_length See the above description for the meaning of interval\_length. (Input)

Entry: stop\_governor\_

This entry stops the limiting of CPU usage.

Usage

```
declare stop_governor_ entry;  
call stop_governor_;
```

There are no arguments.





Name: system\_info\_

This procedure allows the user to obtain information concerning system parameters.

Entry: system\_info\_\$installation\_id

This entry returns the 32 character installation ID typed in the header of who and at dial up time.

Usage

```
declare system_info_$installation_id entry (char(*));  
call system_info_$installation_id (id);
```

1) id is the installation ID. (Output)

Entry: system\_info\_\$sysid

This entry returns the 8 character system ID typed in the header of who and at dial up time.

Usage

```
declare system_info_$sysid entry (char(*));  
call system_info_$sysid (sys);
```

1) sys is the system ID which identifies the current version of the system. (Output)

Entry: system\_info\_\$titles

This entry returns several character strings which more formally identify the installation.

Usage

```
declare system_info_$titles entry (char(*), char(*),  
char(*), char(*));
```

```
call system_info_$titles (c, d, cc, dd);
```

1) c is the company or institution name (a maximum of 64 characters). (Output)

Page 2

- 2) d is the department or division name (a maximum of 64 characters). (Output)
- 3) cc is the company name, double spaced (a maximum of 120 characters). (Output)
- 4) dd is the department name, double spaced (a maximum of 120 characters). (Output)

Entry: system\_info\_\$users

This entry returns the current and maximum number of load units and users.

Usage

```
declare system_info_$users entry (fixed bin, fixed bin,  
fixed bin, fixed bin);
```

```
call system_info_$users (mn, nn, mu, nu);
```

- 1) mn is the maximum number of users. (Output)
- 2) nn is the current number of users. (Output)
- 3) mu is the maximum number of load units (times 10). (Output)
- 4) nu is the current number of load units (times 10). (Output)

Entry: system\_info\_\$timeup

This entry returns the time at which the system was last started up.

Usage

```
declare system_info_$timeup entry (fixed bin(71));
```

```
call system_info_$timeup (tu);
```

- 1) tu is the time the system came up. (Output)

Entry: system\_info\_\$next\_shutdown

This entry returns the time of the next scheduled shutdown, and the reason for the shutdown, and the time the system will return, if this data is available.

Usage

```
declare system_info_$next_shutdown entry (fixed bin(71),  
char(*), fixed bin(71));
```

```
call system_info_$next_shutdown (td, rsn, tn);
```

- 1) td is the time of the next scheduled shutdown. If none is scheduled, this is zero. (Output)
- 2) rsn is the reason for the next shutdown (a maximum of 32 characters). If it is not known, it is blank. (Output)
- 3) tn is the time the system will return, if known; otherwise it is zero. (Output)

Entry: system\_info\_\$prices

This entry returns the per shift prices for interactive use.

Usage

```
declare system_info_$prices entry (0:7) float bin,  
(0:7) float bin, (0:7) float bin, (0:7) float bin,  
float bin, float bin);
```

```
call system_info_$prices (cpu, log, prc, cor, dsk, reg);
```

- 1) cpu is the CPU hour rate per shift. (Output)
- 2) log is the connect hour rate per shift. (Output)
- 3) prc is the process hour rate per shift. (Output)
- 4) cor is the page-second rate per shift. (Output)
- 5) dsk is the page-second rate for secondary storage. (Output)
- 6) reg is the registration fee per user per month. (Output)

Entry: system\_info\_\$device\_prices

This entry returns the per shift prices for system device usage.

Page 4

### Usage

```
declare system_info_$device_prices entry (fixed bin, ptr);  
call system_info_$device_prices (ndev, devp);
```

- 1) ndev is the number of devices with prices. (Output)
- 2) devp points to an array where device prices will be stored. (Input)

### Note

In the above entry, the user must provide the following array for device prices in his storage:

```
declare 1 dvt (16) based (devp) aligned,  
        2 device_id char(8),  
        2 device_price (0:7) float bin;
```

- 1) dvt is the user structure. Only the first ndev of the 16 will be filled in.
- 2) device\_id is the name of the device.
- 3) device\_price is the per hour price by shifts for the device.

Entry: system\_info\_\$shift\_table

This entry returns a table which tells when each shift begins and ends.

### Usage

```
declare system_info_$shift_table entry ((336) fixed bin);  
call system_info_$shift_table (st);
```

- 1) st is a table with one entry for each half hour, beginning with 0000 Monday. The table gives the shift number for that half hour period. Shifts may be from 0 to 7. (Output)

Entry: system\_info\_\$abs\_prices

This entry returns the prices for CPU and real time for each absentee queue.

Usage

```
declare system_info_$abs_prices entry ((4) float bin,  
                                         (4) float bin);
```

```
call system_info_$abs_prices (cpurate, realrate);
```

- 1) cpurate is the price per CPU hour for absentee queues 1-4.  
(Output)
- 2) realrate is the price per real-time hour for absentee  
queues 1-4. (Output)

Entry: system\_info\_\$io\_prices

This entry returns the prices for record transmission (printing or punching) for each I/O daemon queue.

Usage

```
declare system_info_$io_prices entry ((4) float bin);
```

```
call system_info_$io_prices (rp);
```

- 1) rp is the price per 1000 records (a record is 700 bits)  
for each I/O daemon queue. (Output)

Note

All entry points which take more than one argument will count their arguments and not attempt to return more values than there are arguments. Certain arguments, such as the price arrays, must be dimensioned as shown.



Subroutine Call  
5/18/73Name: transform\_command\_

This is a subroutine called by the command processor when running under the Limited Service System. It is used to restrict a user to a specified set of commands. It transforms the commands typed by the user into other commands as specified by a table which may be created by the make\_commands command (described in the MPM Subsystem Writers' Guide).

Usage

```
declare transform_command_ entry (ptr, fixed bin, ptr,  
fixed bin(35));
```

```
call transform_command_ (name_ptr, name_len, table_ptr,  
code);
```

- 1) name\_ptr is a pointer to the name of the command to be transformed. (Input) The transformed name of the command is also returned through this pointer. (Output)
- 2) name\_len is the length (in characters) of the command to be transformed. (Input) The length of the transformed command is also returned in this variable. (Output)
- 3) table\_ptr is a pointer to the table in the format produced by the make\_commands command. (Input)
- 4) code is zero if there are no errors; or is error\_table\_\$noentry, if the command is not in the table. (Output)

Notes

transform\_command\_ prints out an error message if the command given to it cannot be found in the table. The values of name\_ptr and name\_len remain unchanged.





Name: tssi\_

The procedure tssi\_ (translator storage system interface) simplifies the use of the storage system by language translators. The "get" entries prepare a segment for use as output from the translator: creating it if necessary, truncating it, and setting the Access Control List (ACL) to "rwa" for the current user. The "finish" entries set the bitcounts of segments, terminate them, and put the proper ACL on them. The "cleanup" entries are used by cleanup procedures in the translator. There are entries for both single segments and multi-segment files: the single segment entries have "segment" in the entry name, and the multi-segment file entries have "file" in the entry name.

Entry: tssi\_\$get\_segment

This entry returns a pointer to a specified segment. The ACL on the segment will be "rwa" for the current user. If an ACL had to be replaced to do this, aclinfop is returned pointing to information to be used in resetting the ACL.

Usage

```
declare tssi_$get_segment entry (char(*), char(*), ptr, ptr,  
    fixed bin(35));
```

```
call tssi_$get_segment (dname, sname, segp, aclinfop, code);
```

- 1) dname is the directory in which the segment resides. (Input)
- 2) sname is the name of the segment. (Input)
- 3) segp is the pointer to the segment, or is null if an error was encountered. (Output)
- 4) aclinfop is the pointer to ACL information (if any) needed by the finish entries. (Output)
- 5) code is a storage system status code. (Output)

Entry: tssi\_\$get\_file

This entry is the multi-segment file (MSF) version of the get\_segment entry. It will return a pointer to the specified file. Additional components, if necessary, may be accessed using msf\_manager\_\$get\_ptr (see the MPM write-up for msf\_manager\_), with the original segment to be considered as component 0.

Usage

```
declare tssi_$get_file entry (char(*), char(*), ptr, ptr,
ptr, fixed bin(35));
```

```
call tssi_$get_file (dname, sname, segp, aclinfo, fcbp,
code);
```

- 1) dname as above.
- 2) sname as above.
- 3) segp is the pointer to component 0 of the file. (Output)
- 4) aclinfo as above.
- 5) fcbp is the pointer to the file control block (FCB) needed by msf\_manager\_. (Output)
- 6) code as above.

Entry: tssi\_\$finish\_segment

The finish segment entry sets the bitcount on the segment after the translator is finished with it. It also terminates the segment. The ACL is reset to the way it was before the get\_segment entry was called. If none existed then, the mode is set to "mode" for the current user.

Usage

```
declare tssi_$finish_segment entry (ptr, fixed bin(24),
bit(36) aligned, ptr, fixed bin(35));
```

```
call tssi_$finish_segment (segp, bc, mode, aclinfo, code);
```

- 1) segp is the pointer to the segment. (Input)
- 2) bc is the bit count of the segment. (Input)
- 3) mode is the access mode to be put on the segment, e.g., "1100"b for "re", or "1011"b for "rwa". (Input)
- 4) aclinfo is the pointer to the saved ACL information returned by the get\_segment entry. (Input)
- 5) code as above.

Entry: tssi\_\$finish\_file

This entry is the same as the finish\_segment entry, except that it works on MSF's, and closes the file, freeing the FCB.

Usage

```
declare tssi_$finish_file entry (ptr, fixed bin, fixed
    bin(24), bit(36) aligned, ptr, fixed bin(35));
```

```
call tssi_$finish_file (fcbp, component, bc, mode, aclinfop,
    code);
```

- 1) fcbp is the pointer to the FCB returned by the get\_file entry. (Input)
- 2) component is the highest numbered component in the file. (Input)
- 3) bc is the bitcount of the highest numbered component. (Input)
- 4) mode as above.
- 5) aclinfop as above.
- 6) code as above.

Entry: tssi\_\$clean\_up\_segment

Programs which use tssi\_ must establish a cleanup procedure which calls this entry. (For a discussion of cleanup procedures see the MPM Reference Guide section Nonlocal Transfers and Cleanup Procedures.) If more than one call is made to tssi\_\$get\_segment, the cleanup procedure must make the appropriate call to tssi\_\$clean\_up\_segment for each aclinfop.

The purpose of this call is to free the storage that the get\_segment entry allocated to save the old ACLs of the segments being translated. It is to be used in case the translation is aborted (e.g., by a quit).

Usage

```
declare tssi_$clean_up_segment entry (ptr);  
call tssi_$clean_up_segment (aclinfop);
```

1) aclinfop as above.

Entry: tssi\_\$clean\_up\_file

This entry is the cleanup entry for MSF's. In addition to freeing ACL's, it closes the file, freeing the FCB.

Usage

```
declare tssi_$clean_up_file entry (ptr, ptr);  
call tssi_$clean_up_file (fcbp, aclinfop);
```

1) fcbp as above.

2) aclinfop as above.

Subroutine Call  
Development System  
05/18/71

Name: unwinder\_

The procedure unwinder\_ is used to perform a "non-local goto" on the Multics stack. It is not intended to be called by direct programming (i.e., an explicit "call" statement in a program) but, rather, by the generated code of a translator. For example, it is automatically invoked by a PL/I "goto" statement involving a non-local label variable. The ordinary user or subsystem writer should have no use for it.

When invoked, unwinder\_ traces the Multics stack backwards until it finds the stack frame associated with its label variable argument or until the stack is exhausted. In each stack frame it passes, it invokes the handler (if any) for the condition "cleanup". When it finds the desired stack frame, it passes control to the procedure associated with that frame at the location indicated by the label variable argument. If the desired stack frame cannot be found or if other obscure error conditions arise (e.g., the stack is not threaded correctly), unwinder\_ signals the condition "unwinder\_error".

#### Usage

```
declare unwinder_ entry (label);
```

```
call unwinder_ (tag);
```

1) tag is a non-local label variable. (Input)

#### Note

The current implementation of unwinder\_ does not cross protection rings.



Subroutine Call  
4/5/73Name: user\_info\_

This procedure allows the user to obtain information concerning his login session. The following entries are documented in the MPM Reference Guide:

```

user_info_
user_info_$whoami
user_info_$login_data
user_info_$usage_data
user_info_$homedir
user_info_$responder
user_info_$tty_data
user_info_$logout_data
user_info_$absin
user_info_$absout
user_info_$limits

```

Entry: user\_info\_\$absentee\_queue

This entry returns the user's current absentee queue.

Usage

```

declare user_info_$absentee_queue entry (fixed bin);
call user_info_$absentee_queue (q);

```

- 1) q is 1, 2, or 3 if the user is running on absentee queue 1, 2, or 3, respectively. If the user is not an absentee user, the value is -1. (Output)

Entry: user\_info\_\$load\_ctl\_info

This entry returns various load control parameters.

Usage

```

declare user_info_$load_ctl_info entry (char(*), fixed bin,
fixed bin(71), fixed bin);
call user_info_$load_ctl_info (group, status, protected,
weight);

```

- 1) group is the name of the user's load control group. (Output)
- 2) status = 0 if the user is a primary user;  
= 1 if the user is a secondary user. (Output)

Page 2

3) protected for primary users, this is the time when they become preemptable by others in their group. (Output)

4) weight is ten times the user's weight. (Output)

Entry: user\_info\_\$attributes

This entry returns the user's permission attributes, as defined by user control.

Usage

```
declare user_info_$attributes entry (char(300) varying);
```

```
call user_info_$attributes (attstring);
```

1) attstring is a character string which lists the user's attributes. The attributes are separated by commas and end with a semicolon. The legal attributes are:

```
administrator
anonymous
brief
dialok
guaranteed_login
multip
no_eo
no_primary
no_secondary
nobump
nolist
nostartup
preempting
vhomedir
vinitproc (Output)
```

Entry: user\_info\_\$outer\_module

This entry returns the name of the user's terminal outer module at process creation.

Usage

```
declare user_info_$outer_module entry (char(*));
```

```
call user_info_$outer_module (mod);
```

1) mod is the outer module name. (Output)



5/31/73

INDEX

This Index covers Parts II and III of the Multics Programmers' Manual, namely the Reference Guide and the Subsystem Writers' Guide.

The Index is organized around the numerically ordered Reference Guide and Subsystem Writers' Guide sections and the alphabetically ordered commands and subroutine write-ups, rather than by page number. Thus, for example, the entry for command level might read:

```
command level
  1.4
  cu_
  get_to_cl_ (SWG)
  listen_ (SWG)
```

The first item under command level refers to the Reference Guide section 1.4, the second to the write-up for the cu\_ subroutine, and the last two to the write-ups (in the SWG) for the get\_to\_cl\_ and listen\_ subroutines. They are referenced in the order that they appear in this manual. Note that command names can normally be distinguished from subroutines by the trailing underscore in the segment name of subroutines.

Some entries are of the form:

```
I/O (bulk)
  see bulk I/O
```

For simplicity of usage, these entries always refer to other places in the Index, never to normal Reference Guide or Subsystem Writers' Guide documents.

Some entries are followed by information within parentheses. This information serves to explain the entry by giving a more complete name or the name of the command under which the actual entry can be found. For example:

```
e (enter)
listnames (list)
```

In addition to this Index, other indexes to information are:

- 1) MPM Table of Contents
  - lists names of commands and subroutines with write-up issue dates
  - lists commands and subroutines documented under other write-ups; e.g., console\_output: see file\_output
- 2) Reference Guide Section 1.1: The Multics Command Repertoire
  - lists commands by function
- 3) Reference Guide Section 2.1: The Multics Subroutine Repertoire
  - lists subroutines by function
- 4) Reference Guide Section 8.3: Obsolete Procedures

- ! convention
  - see unique strings
- \* convention
  - see star convention
- 7-punch cards
  - see seven-punch cards
- <
  - expand\_path\_
    - see directories
- = convention
  - see equal convention
- >
  - expand\_path\_
    - see directories
    - see root directory
- abbreviations
  - 1.6
  - abbrev
    - see alternate names
    - see command processing
- ABEND
  - see error handling
- absentee queue
  - user (SWG)
  - user\_info\_ (SWG)
- absentee usage
  - 1.7
  - alm\_abs
  - cancel\_abs\_request
  - enter\_abs\_request
  - exec\_com
  - fortran\_abs
  - how\_many\_users
  - list\_abs\_requests
  - pll\_abs
  - runoff\_abs
  - who
- absin
  - see absentee usage
- absolute path names
  - expand\_path\_
    - see path names
    - see storage system
- access control
  - see protection
- access control list
  - 3.3
  - 3.4
  - deleteacl
  - deletecacl (deleteacl)
  - listacl
  - listcacl (listacl)
  - setacl
  - setcacl (setacl)
  - hcs\_\$add\_acl\_entries
  - hcs\_\$add\_dir\_acl\_entries
  - hcs\_\$delete\_acl\_entries
  - hcs\_\$delete\_dir\_acl\_entries
  - hcs\_\$list\_acl
  - hcs\_\$list\_dir\_acl
  - hcs\_\$replace\_acl
  - hcs\_\$replace\_dir\_acl
  - see protection
- account ID
  - user (SWG)
- accounting
  - resource\_usage
  - user (Active Function)
  - cpu\_time\_and\_paging\_
  - user\_info\_
    - see metering
  - hcs\_\$get\_process\_usage (SWG)
  - hcs\_\$quota\_get (SWG)
- ACL
  - see access control list
- active functions
  - 1.4
  - 1.7
  - active\_fnc\_err\_
- address reuse
  - hcs\_\$initiate
  - (continued)

Page 4

- address reuse
  - (continued)
  - hcs\_\$initiate\_count
  - hcs\_\$terminate\_file
  - hcs\_\$terminate\_name
  - hcs\_\$terminate\_noname
  - hcs\_\$terminate\_seg
- address space
  - 3.2
  - bind
  - get\_pathname (Active Function)
  - new\_proc
  - terminate
  - where
  - hcs\_\$delentry\_seg
  - hcs\_\$fs\_get\_ref\_name
  - hcs\_\$fs\_get\_seg\_ptr
  - hcs\_\$initiate
  - hcs\_\$initiate\_count
  - hcs\_\$make\_ptr
  - hcs\_\$make\_seg
  - hcs\_\$terminate\_file
  - hcs\_\$terminate\_name
  - hcs\_\$terminate\_noname
  - hcs\_\$terminate\_seg
  - see directory entry names
- aggregate data
  - 5.4
- alarms
  - timer\_manager\_
  - see clocks
- algol
  - 7.2
- aliases
  - see directory entry names
- alm
  - alm\_abs
- alternate names
  - see directory entry names
- anonymous users
  - 1.2
  - (continued)
- anonymous users
  - (continued)
  - enter
  - user (Active Function)
  - user\_info\_
- answering questions
  - answer
- archive segments
  - 5.5
- archiving
  - archive
  - archive\_sort
  - reorder\_archive
- ARDS display
  - see graphics
  - see terminals
- areas
  - area\_
  - alloc\_ (SWG)
  - area\_ (SWG)
  - area\_assign\_ (SWG)
  - freen\_ (SWG)
  - get\_system\_free\_area\_ (SWG)
- argument count
  - 5.4
  - cu\_
- argument descriptors
  - 5.4
  - decode\_descriptor\_
- argument list pointer
  - 5.4
  - cu\_
- argument lists
  - debug
  - trace\_stack
  - cu\_
  - decode\_descriptor\_
  - 12.2 (SWG)

- arithmetic operations
  - divide (Active Function)
  - minus (Active Function)
  - mod (Active Function)
  - plus (Active Function)
  - times (Active Function)
- array data
  - 5.4
- ASCII
  - 5.1
  - 5.2
- asking questions
  - answer
  - query (Active Function)
  - response (Active Function)
  - command\_query\_
  - dl\_handler\_ (SWG)
  - nd\_handler\_ (SWG)
- assembly languages
  - 8.5
  - alm
- attach table
  - 4.2
  - print\_attach\_table
  - ios\_
  - see I/O attachments
  - get\_at\_entry\_ (SWG)
- attachments
  - see I/O attachments
- attention
  - see process interruption
- author
  - 3.3
  - status
  - hcs\_\$star\_
  - hcs\_\$status\_
  - hcs\_\$get\_author (SWG)
- automatic logout
  - see logging out
- automatic variables
  - see stack segments
- background jobs
  - see absentee usage
- base conversion
  - see conversion
- BASIC
  - 7.2
  - basic
  - basic\_run
  - basic\_system
  - print\_dartmouth\_library
  - set\_dartmouth\_library
  - v5basic
- batch processing
  - see absentee usage
- binding
  - archive
  - bind
  - print\_bind\_map
  - make\_object\_map\_
  - see linking
- bit count author
  - hcs\_\$get\_bc\_author (SWG)
- bit counts
  - 3.3
  - adjust\_bit\_count
  - set\_bit\_count
  - status
  - adjust\_bit\_count\_
  - decode\_object\_
  - hcs\_\$initiate\_count
  - hcs\_\$set\_bc
  - hcs\_\$set\_bc\_seg
  - hcs\_\$star\_
  - hcs\_\$status\_
  - hcs\_\$get\_bc\_author (SWG)
- bit-string data
  - 5.4

Page 6

- blocks
  - see interprocess communication
  - see storage management
  - hcs\_\$wakeup (SWG)
  - ipc\_ (SWG)
- bound segments
  - 11.8 (SWG)
- brackets
  - see command language
  - see protection
- branches
  - see directories
  - see segments
- break
  - see process interruption
- breakpoints
  - debug
  - 11.6 (SWG)
- brief modes
  - change\_error\_mode
  - ready\_off
- broadcasting
  - broadcast\_
- bulk I/O
  - 4.1
  - 4.4
  - 5.3
  - console\_output
  - dprint
  - dpunch
  - file\_output
  - nstd\_
  - dprint\_ (SWG)
- CACL
  - see access control list
- call operator
  - 12.2 (SWG)
- calling sequences
  - 12.2 (SWG)
- cancelling
  - cancel\_abs\_request
  - see deleting
- canonicalization
  - 1.3
  - tw\_
- card formats
  - 4.4
- cards
  - see I/O
  - see punched cards
- catalogs
  - see directories
  - see directory entry names
- changing names
  - see directory entry names
- changing working directory
  - see working directory
- character codes
  - 1.3
  - 5.1
  - 5.2
- character formats
  - 5.1
- character string operations
  - index (Active Function)
  - length (Active Function)
  - substr (Active Function)
- character string output
  - ioa\_
  - ios\_
  - write\_list\_
  - ioa\_ (SWG)
- character string segments
  - 5.5

- character-string data
  - 5.4
- charges
  - see prices
- checking changes
  - check\_info\_segs
- checksum
  - 8.4
- cleanup tools
  - 6.2
  - 6.3
  - adjust\_bit\_count
  - compare\_ascii
  - display\_component\_name
  - endfile
  - fs\_chname
  - new\_proc
  - release
  - set\_bit\_count
  - terminate
  - truncate
  - adjust\_bit\_count\_
  - compare\_ascii\_
  - establish\_cleanup\_proc\_
  - hcs\_\$set\_bc
  - hcs\_\$set\_bc\_seg
  - hcs\_\$terminate\_file
  - hcs\_\$terminate\_name
  - hcs\_\$terminate\_noname
  - hcs\_\$terminate\_seg
  - hcs\_\$truncate\_file
  - hcs\_\$truncate\_seg
  - revert\_cleanup\_proc\_
  - term\_
- clocks
  - clock\_
  - convert\_date\_to\_binary\_
  - date\_time\_
  - decode\_clock\_value\_
  - timer\_manager\_
- closing files
  - endfile
  - see bit counts
  - see termination
- code conversion
  - see conversion
- coding standards
  - 2.5
- collating sequence
  - 5.1
  - 5.2
  - sort\_file
- combined linkage area
  - 12.1 (SWG)
- combined linkage segment
  - 3.1
- combining segments
  - archive
  - bind
- command environment
  - Section 1
  - 1.4
- command language
  - 1.4
  - 1.7
  - abbrev
  - get\_com\_line
  - set\_com\_line
  - see command processing
  - 14.3 (SWG)
- command level
  - 1.4
  - cu\_
  - get\_to\_cl\_ (SWG)
  - listen\_ (SWG)
- command names
  - 1.5
  - abbrev
  - see directory entry names
  - see searching
- command processing
  - 1.3
  - abbrev
  - (continued)

Page 8

command processing  
    (continued)  
    enter\_abs\_request  
    exec\_com  
    get\_com\_line  
    set\_com\_line  
    walk\_subtree  
    active\_fnc\_err\_  
    cu\_  
    hcs\_\$star\_  
    see active functions  
    see searching

command utility procedures  
    cu\_

commands  
    1.1  
    1.4  
    1.6  
    Section 9  
    see command processing

common access control list  
    see access control list

comparing character strings  
    equal (Active Function)  
    greater (Active Function)  
    less (Active Function)  
    compare\_ascii\_

comparing segments  
    compare\_ascii

compilers  
    see languages

complex data  
    5.4

condition names  
    1.5

conditions  
    6.1  
    6.2  
    6.3  
    (continued)

conditions  
    (continued)  
    6.5  
    change\_error\_mode  
    program\_interrupt  
    reprint\_error  
    active\_fnc\_err\_  
    com\_err\_  
    condition\_  
    default\_handler\_  
    reversion\_  
    signal\_  
    see cleanup tools  
    see process interruption  
    see unwinding  
    condition\_interpreter\_ (SWG)  
    standard\_default\_handler\_ (SWC)

console line length  
    see terminal line length

console output  
    see I/O  
    see interactive I/O

consoles  
    see terminals

control arguments  
    14.3 (SWG)

control characters  
    1.3  
    5.1  
    ioa\_  
    see character codes  
    ioa\_ (SWG)

conventions  
    11.7 (SWG)

conversion  
    com\_err\_  
    convert\_binary\_integer\_  
    convert\_date\_to\_binary\_  
    cv\_bin\_  
    cv\_dec\_  
    cv\_float\_  
    cv\_oct\_  
    (continued)



- conversion
  - (continued)
  - date\_time\_
  - decode\_clock\_value\_
  - read\_list\_
  - write\_list\_
  - see formatted I/O
  - see I/O
- coordination
  - set\_lock\_ (SWG)
- copy switch
  - ~~3.3~~
  - hcs\_\$initiate
  - hcs\_\$initiate\_count
- copying
  - copy
  - copy\_acl\_
  - copy\_names\_
  - copy\_seg\_
- cost saving features
  - alm\_abs
  - fortran\_abs
  - p11\_abs
  - see absentee usage
  - see archiving
  - see limited service systems
- CPU usage
  - ready
  - see metering
  - see time
- crawling out
  - see error handling
- creating directories
  - createdir
  - hcs\_\$append\_branchx
- creating links
  - link
  - hcs\_\$append\_link
- creating processes
  - enter\_abs\_request
  - (continued)
- creating processes
  - (continued)
  - login
  - logout
  - new\_proc
  - see logging in
- creating segments
  - basic\_system
  - copy
  - create
  - edm
  - qedx
  - hcs\_\$append\_branch
  - hcs\_\$append\_branchx
  - hcs\_\$make\_seg
- creator
  - see author
- current length
  - 3.3
  - see length of segments
- daemon
  - dprint
  - dpunch
  - see bulk I/O
  - dprint\_ (SWG)
- daemon\_dir\_dir
  - 3.1
- Dartmouth facilities
  - 7.2
  - basic
  - basic\_run
  - basic\_system
  - print\_dartmouth\_library
  - set\_dartmouth\_library
  - v5basic
- data control word
  - 4.2
- data conversion
  - see conversion

Page 10

## data representation

4.2  
5.3  
5.4  
8.4

## date conversion

see conversion

## dates

3.3  
date (Active Function)  
date (Active Function)  
date\_time (Active Function)  
date\_time (Active Function)  
day (Active Function)  
day (Active Function)  
day\_name (Active Function)  
day\_name (Active Function)  
long\_date (Active Function)  
month (Active Function)  
month\_name (Active Function)  
year (Active Function)  
clock\_  
convert\_date\_to\_binary\_  
date\_time\_  
decode\_clock\_value\_

## DCW

see data control word

## debugging tools

change\_error\_mode  
compare\_ascii  
debug  
display\_component\_name  
dump\_segment  
hold  
reprint\_error  
trace\_stack  
compare\_ascii\_  
stu\_

## decimal integers

convert\_binary\_integer\_  
see conversion

## default error handling

6.5  
(continued)

default error handling  
(continued)

change\_error\_mode  
reprint\_error  
active\_fnc\_err\_  
see process interruption  
condition\_interpreter\_ (SWG)  
condition\_interpreter\_ (SWG)

## default status messages

com\_err\_

## default working directory

change\_default\_wdir  
change\_wdir  
print\_default\_wdir  
get\_default\_wdir\_

## deferred execution

see absentee usage

## definition sections

11.3 (SWG)

## deleting

delete  
delete\_dir  
deleteforce  
terminate  
unlink  
delete\_  
hcs\_\$del\_dir\_tree  
hcs\_\$delentry\_file  
hcs\_\$delentry\_seg  
term\_  
see address reuse  
see cancelling  
see canonicalization  
see termination  
dl\_handler\_ (SWG)  
nd\_handler\_ (SWG)

## delimiters

4.2

## descriptors

5.4  
decode\_descriptor\_

- desk calculators
  - calc
  - decam
- device interface modules
  - see I/O system interface
- devices
  - system\_info\_ (SWG)
- dialing up
  - 1.2
- DIM
  - see I/O system interface
- directories
  - 3.1
  - list
  - listnames (list)
  - listtotals (list)
  - walk\_subtree
  - see creating directories
  - see default working directory
  - see deleting
  - see directory entry names
  - see home directory
  - see libraries
  - see process directories
  - see protection
  - see root directory
  - see storage quotas
  - see storage system
  - see working directory
- directory access modes
  - delete\_iacl\_dir
  - list\_iacl\_dir
  - set\_iacl\_dir
  - hcs\_\$add\_dir\_acl\_entries
  - hcs\_\$delete\_dir\_acl\_entries
  - hcs\_\$list\_dir\_acl
  - hcs\_\$replace\_dir\_acl
  - hcs\_\$add\_dir\_inacl\_entries (SWG)
  - hcs\_\$delete\_dir\_inacl\_entries (SWG)
  - hcs\_\$list\_dir\_inacl (SWG)
  - hcs\_\$list\_inacl (SWG)
  - hcs\_\$replace\_dir\_inacl (SWG)
  - hcs\_\$replace\_inacl (SWG)
- directory attributes
  - 3.3
  - delete\_iacl\_dir
  - delete\_iacl\_seg
  - list
  - listnames (list)
  - listtotals (list)
  - list\_iacl\_dir
  - list\_iacl\_seg
  - set\_iacl\_dir
  - set\_iacl\_seg
  - status
  - hcs\_\$add\_acl\_entries
  - hcs\_\$add\_dir\_acl\_entries
  - hcs\_\$delete\_acl\_entries
  - hcs\_\$delete\_dir\_acl\_entries
  - hcs\_\$list\_acl
  - hcs\_\$list\_dir\_acl
  - hcs\_\$replace\_acl
  - hcs\_\$replace\_dir\_acl
  - hcs\_\$star\_
  - hcs\_\$status\_
  - see protection
  - hcs\_\$add\_dir\_inacl\_entries (SWG)
  - hcs\_\$add\_inacl\_entries (SWG)
  - hcs\_\$delete\_dir\_inacl\_entries (SWG)
  - hcs\_\$delete\_inacl\_entries (SWG)
  - hcs\_\$get\_dir\_ring\_brackets (SWG)
  - hcs\_\$list\_dir\_inacl (SWG)
  - hcs\_\$list\_inacl (SWG)
  - hcs\_\$replace\_dir\_inacl (SWG)
  - hcs\_\$replace\_inacl (SWG)
  - hcs\_\$set\_dir\_ring\_brackets (SWG)
- directory creation
  - see creating directories
- directory deletion
  - see deleting
- directory entries
  - see directories
  - see links
  - see segments
- directory entry names
  - addname
  - deletename
  - entry (Active Function)
  - (continued)

Page 12

- directory entry names
  - (continued)
  - fs\_chname
  - list
  - listnames (list)
  - listtotals (list)
  - names
  - rename
  - status
  - strip\_entry (Active Function)
  - suffix (Active Function)
  - where
  - equal\_
  - hcs\_\$chname\_file
  - hcs\_\$chname\_seg
  - hcs\_\$fs\_get\_path\_name
  - hcs\_\$star\_
  - hcs\_\$status\_
  - see path names
  - see unique names
- directory hierarchy
  - Section 3
  - copy
  - link
  - move
  - status
  - unlink
  - walk\_subtree
  - copy\_acl\_
  - copy\_names\_
  - see storage system
- directory names
  - see default working directory
  - see directory entry names
  - see home directory
  - see process directories
  - see working directory
- directory renaming
  - see directory entry names
- directory restructuring
  - move
  - hcs\_\$fs\_move\_file
  - hcs\_\$fs\_move\_seg
- discarding output
  - discard\_output\_
- disconnected processes
  - see absentee usage
- disconnections
  - see logging out
- display terminals
  - 4.5
  - see graphics
  - see terminals
- diverting output
  - console\_output
  - file\_output
  - iocall
  - discard\_output\_
  - see I/O streams
- dope
  - see descriptors
- dumping segments
  - dump\_segment
- dynamic linking
  - 3.2
  - term\_
  - see address reuse
  - see linkage sections
  - see linking
  - see searching
  - see termination
- e (enter)
  - see logging in
- EBCDIC
  - 5.2
- editing
  - basic\_system
  - edm
  - qedx
- efficiency
  - see metering

- element size  
4.2
- emergency logout  
see logging out
- end of file  
see bit counts
- enter  
see logging in
- enterp  
see logging in
- entries  
see directories  
see links  
see segments
- entry names  
see directory entry names  
see entry point names
- entry operator  
12.2 (SWG)
- entry point data  
5.4
- entry point names  
print\_link\_info  
hcs\_\$make\_ptr  
see linking
- entry points  
5.4  
see interprocedure communication  
see linking
- entry sequence gates  
11.2 (SWG)
- entry sequences  
11.7 (SWG)
- EOF  
see end of file
- ep (enterp)  
see logging in
- EPL (obsolete)  
see PL/I language
- eplbsa (obsolete)  
see alm
- equal convention  
equal\_
- equals convention  
1.5
- erase characters  
1.3
- erasing  
1.3  
see canonicalization  
see deleting
- error codes  
see status codes
- error handling  
Section 6  
6.1  
6.2  
change\_error\_mode  
reprint\_error  
active\_fnc\_err\_  
com\_err\_  
command\_query\_  
condition\_  
default\_handler\_  
establish\_cleanup\_proc\_  
reversion\_  
revert\_cleanup\_proc\_  
signal\_  
see debugging tools  
see help  
convert\_status\_code\_ (SWG)  
condition\_interpreter\_ (SWG)  
standard\_default\_handler\_ (SWG)
- error messages  
see status messages

Page 14

- error recovery
  - 6.3
  - hold
  - program\_interrupt
  - release
  - establish\_cleanup\_proc\_  
see cleanup tools
  - see debugging tools
  - see process interruption
- error tables
  - see status tables
- error\_output
  - see I/O streams
- error\_table\_
  - see status codes
- escape conventions
  - 1.3
  - 5.2
- event channels
  - hcs\_\$wakeup (SWG)
  - ipc\_ (SWG)
- exec\_com
  - see active functions
- existence checking
  - exists (Active Function)
- expanded command line
  - see command processing
- expression evaluators
  - calc
  - see desk calculators
- expression words
  - 11.3 (SWG)
- external data
  - 5.4
- external symbols
  - print\_link\_info
  - make\_object\_map\_  
(continued)
- external symbols
  - (continued)
  - see interprocedure communication
  - see linking
- faults
  - 6.1
  - 6.5
  - see conditions
  - 13.6 (SWG)
- file I/O
  - file\_
- file mark
  - see bit counts
  - see magnetic tapes
- file system
  - 4.2
  - see storage system
- files
  - 5.3
  - file\_  
see I/O
  - see segments
- first-reference traps
  - 11.4 (SWG)
- fixed point data
  - 5.4
- floating point data
  - 5.4
- formats
  - 5.5
- formatted I/O
  - 4.1
  - 4.3
  - ioa\_  
see conversion
  - ioa\_ (SWG)
- formatted input
  - read\_list\_

- formatted output
  - runoff
  - runoff\_abs
  - ioa\_
  - write\_list\_
  - ioa\_ (SWG)
- formatting character strings
  - format\_line (Active Function)
  - string (Active Function)
- FORTRAN
  - 7.2
  - endfile
  - fortran
  - fortran\_abs
- free storage
  - see storage management
  - see storage management
- functions
  - see active functions
  - see procedures
- gate segments
  - 13.4 (SWG)
- gates
  - see protection
  - 13.4 (SWG)
- generating calls
  - cu\_
  - hcs\_\$make\_ptr
  - see pointer generation
  - find\_command\_ (SWG)
- generating pointers
  - see pointer generation
- graphic characters
  - see character codes
- graphic terminals
  - see display terminals
  - see terminals
- graphics
  - 4.1
  - 4.5
  - plot\_
  - see display terminals
- handling of unusual occurrences
  - Section 6
  - 6.1
- hardware registers
  - debug
- help
  - help
  - peruse\_text
- hierarchy
  - see directories
- hierarchy searching
  - see searching
- hold
  - see error recovery
  - see process interruption
- home directory
  - home\_dir (Active Function)
  - set\_search\_rules
  - user (Active Function)
  - user\_info\_
  - see default working directory
- I/O
  - Section 4
  - ioctl
  - print
  - ioa\_
  - ios\_
  - tape\_
  - see conversion
  - see formatted I/O
  - ioa\_ (SWG)
- I/O (bulk)
  - see bulk I/O

Page 16

- I/O attachments
  - 4.2
  - print\_attach\_table
  - get\_at\_entry\_ (SWG)
- I/O calls
  - 4.3
  - ios\_
- I/O cleanup
  - endfile
  - see cleanup tools
- I/O commands
  - console\_output
  - dprint
  - dpunch
  - file\_output
  - locall
  - iomode
  - line\_length
- I/O daemon
  - see daemon
- I/O errors
  - see I/O status
- I/O facilities
  - 4.1
- I/O modes
  - 4.2
  - locall
  - iomode
  - ios\_
- I/O status
  - 4.2
  - ios\_
- I/O streams
  - 4.2
  - locall
  - iomode
  - ios\_
  - syn
  - see stream names
- I/O switch
  - 4.2
  - 4.6
  - ios\_
  - syn
- I/O system flowchart
  - 4.2
- I/O system interface
  - 4.2
  - 4.3
  - 4.6
  - locall
  - iomode
  - line\_length
  - print\_attach\_table
  - broadcast\_
  - file\_
  - ios\_
  - syn
  - tw\_
  - see IOSIM
  - get\_at\_entry\_ (SWG)
- IBM 1050
  - see terminals
- IBM 2741
  - see terminals
- include files
  - 2.2
  - 3.2
  - p11
- information
  - check\_info\_segs
  - help
  - make\_peruse\_text
  - peruse\_text
  - who
  - see metering
  - see status
  - system\_info\_ (SWG)
- initial access control list
  - delete\_iacl\_dir
  - delete\_iacl\_seg
  - (continued)



- initial access control list
  - (continued)
  - list\_iacl\_dir
  - list\_iacl\_seg
  - set\_iacl\_dir
  - set\_iacl\_seg
  - see protection
  - hcs\_\$add\_inacl\_entries (SWG)
  - hcs\_\$add\_dir\_inacl\_entries (SWG)
  - hcs\_\$delete\_dir\_inacl\_entries(SWG)
  - hcs\_\$delete\_inacl\_entries (SWG)
  - hcs\_\$list\_dir\_inacl (SWG)
  - hcs\_\$list\_inacl (SWG)
  - hcs\_\$replace\_dir\_inacl (SWG)
  - hcs\_\$replace\_inacl (SWG)
- initial access control lists
  - 3.3
- initial ACL
  - see initial access control list
- initialized segments
  - set\_search\_rules
  - see Known Segment Table
- initiation
  - initiate
  - where
  - hcs\_\$initiate
  - hcs\_\$initiate\_count
  - hcs\_\$make\_ptr
  - hcs\_\$make\_seg
  - see dynamic linking
  - see linking
- input
  - ios\_
  - read\_list\_
  - see I/O
- input conversion
  - see formatted I/O
- installation parameters
  - system\_info\_ (SWG)
- integer representation
  - convert\_binary\_integer\_
- interaction tools
  - answer
  - program\_interrupt
  - command\_query\_
  - see debugging tools
  - see interactive I/O
- interactive I/O
  - ioa\_
  - read\_list\_
  - write\_list\_
  - ioa\_ (SWG)
- intermediate interface modules
  - see I/O system interface
- internal storage
  - 11.4 (SWG)
- interprocedure communication
  - see linking
  - hcs\_\$wakeup (SWG)
  - ipc\_ (SWG)
- interprocess communication
  - hcs\_\$wakeup (SWG)
  - ipc\_ (SWG)
- interrupts
  - 6.5
  - 8.5
  - program\_interrupt
  - see process interruption
- intersegment linking
  - print\_link\_info
  - make\_object\_map\_
  - see dynamic linking
  - see linking
- interuser communication
  - mail
  - hcs\_\$wakeup (SWG)
  - ipc\_ (SWG)
- IOSIM
  - nstd\_
  - tape\_
  - see T/O system interface
  - see synonyms

Page 18

IOSIM example  
4.6iteration  
index\_set (Active Function)Job Control Language  
see command processingjobs  
see absentee usage  
see processeskeypunches  
1.3kill characters  
1.3killing  
see cancellingKnown Segment Table (KST)  
3.1KST  
see Known Segment Tablel (login)  
see logging inlabel data  
5.4languages  
2.2  
7.2  
alm  
basic  
bind  
calc  
debug  
decam  
edm  
exec\_com  
fortran  
lisp  
pll  
qedx  
(continued)languages  
(continued)  
runoff  
runoff\_abs  
v5basiclength of arguments  
cu\_length of segment  
truncatelength of segments  
adjust\_bit\_count  
list  
listnames (list)  
listtotals (list)  
set\_bit\_count  
status  
adjust\_bit\_count\_  
decode\_object\_  
hcs\_\$initiate\_count  
hcs\_\$set\_bc  
hcs\_\$star\_  
hcs\_\$status\_  
hcs\_\$truncate\_file  
hcs\_\$truncate\_seg  
see bit countslibraries  
3.1  
3.2  
print\_dartmouth\_library  
print\_search\_rules  
set\_dartmouth\_library  
set\_search\_dirs  
set\_search\_rules  
hcs\_\$get\_search\_rules (SWG)  
hcs\_\$initiate\_search\_rules (SWG)limited service systems  
7.1  
7.2  
make\_commands (SWG)  
lss\_login\_responder\_ (SWG)  
transform\_command\_ (SWG)link attributes  
3.3  
(continued)

- link attributes
  - (continued)
  - list
  - listnames (list)
  - listtotals (list)
  - status
  - hcs\_\$star\_
  - hcs\_\$status\_
- link creation
  - see creating links
- link deletion
  - see deleting
- link names
  - see directory entry names
- link renaming
  - see directory entry names
- link resolution
  - hcs\_\$status\_
- linkage offset table
  - 12.1 (SWG)
- Linkage Offset Table (LOT)
  - see dynamic linking
  - see linking
- linkage sections
  - print\_link\_info
  - make\_object\_map\_
  - see linking
  - 11.4 (SWG)
  - 12.1 (SWG)
- linking
  - 3.2
  - bind
  - link
  - print\_search\_rules
  - set\_search\_dirs
  - set\_search\_rules
  - terminate
  - unlink
  - delete\_
  - hcs\_\$make\_ptr
  - (continued)
- linking
  - (continued)
  - see binding
  - see creating links
  - see dynamic linking
  - hcs\_\$get\_search\_rules (SWG)
  - hcs\_\$initiate\_search\_rules (SWG)
- links
  - see linking
  - 11.4 (SWG)
- LISP
  - 7.2
  - lisp
- listener
  - 1.3
  - cu\_
  - listen\_ (SWG)
- listing
  - list
  - listnames (list)
  - listtotals (list)
  - print
  - see I/O
  - see storage system
- load control group
  - user (SWG)
- load control parameters
  - user\_info\_ (SWG)
- loading
  - see binding
  - see linking
- locking
  - set\_lock\_ (SWG)
- logging in
  - 1.2
  - enter
  - login
- logging out
  - 1.2
  - logout

Page 20

- logical operations
  - and (Active Function)
  - not (Active Function)
  - or (Active Function)
- login
  - see logging in
- login directory
  - see default working directory
  - see logging in
- login responder
  - user (Active Function)
  - user\_info\_
- login time
  - user (Active Function)
  - user\_info\_
- login word
  - user (Active Function)
  - user\_info\_
- logon
  - see logging in
- logout
  - logout
  - see logging out
- LOT
  - see Linkage Offset Table
- machine conditions
  - debug
  - trace\_stack
- machine languages
  - 8.5
  - alm
  - debug
- macros
  - 1.7
  - abbrev
  - exec\_com
  - qedx
  - see active functions
  - see command processing
- magnetic tapes
  - 5.3
  - 8.4
  - nstd\_tape\_
- mail
  - see interuser communication
- mail box checking
  - mail
- main program
  - see procedures
  - see programming environment
- making known
  - see initiation
- making unknown
  - see termination
- maps
  - print\_bind\_map
  - make\_object\_map\_
  - 11.6 (SWG)
- maximum length
  - 3.3
- maximum line length
  - line\_length
- maximum segment length
  - set\_max\_length (SWG)
  - hcs\_\$get\_max\_length (SWG)
  - hcs\_\$set\_max\_length (SWG)
  - hcs\_\$set\_max\_length\_seg (SWG)
- mcc
  - see punched cards
- mcc cards
  - 4.4
- message of the day
  - print\_motd

messages  
 see I/O  
 see status messages  
 condition\_interpreter\_ (SWG)

metering  
 page\_trace  
 print\_linkage\_usage  
 resource\_usage  
 cpu\_time\_and\_paging\_  
 hcs\_\$status\_  
 timer\_manager\_  
 total\_cpu\_time\_  
 hcs\_\$get\_process\_usage (SWG)  
 hcs\_\$reset\_working\_set (SWG)

MIX  
 7.2

modes  
 3.4  
 4.2  
 see protection  
 see status

modifying segments  
 debug

monitoring  
 see metering

moving names  
 move\_names\_  
 see directory entry names

moving quotas  
 see storage quotas

moving segments  
 move  
 hcs\_\$fs\_move\_file  
 hcs\_\$fs\_move\_seg

multi-segment files  
 3.5  
 see I/O  
 msf\_manager\_ (SWG)

Multics card code  
 4.4  
 5.2  
 see punched cards

multiple device I/O  
 see broadcasting

multiple names  
 see directory entry names

name copying  
 copy\_names\_  
 see directory entry names

name duplications  
 nd\_handler\_ (SWG)

name space  
 see address space

names  
 1.5  
 see address space  
 see directory entry names  
 see path names

naming  
 see directory entry names

naming conventions  
 1.5  
 8.1  
 see directory entry names

nonlocal gotos  
 6.3

number conversion  
 see conversion

object maps  
 11.6 (SWG)

object segments  
 5.5  
 bind  
 print\_bind\_map  
 decode\_object\_  
 (continued)

Page 22

- object segments
  - (continued)
  - make\_object\_map\_
  - see linkage sections
  - 11.1 (SWG)
  - 11.2 (SWG)
  - 11.3 (SWG)
  - 11.4 (SWG)
  - 11.5 (SWG)
  - 11.6 (SWG)
  - 11.7 (SWG)
  - 11.8 (SWG)
- obsolete procedures
  - 8.3
- octal dumping of segments
  - debug
  - dump\_segment
- octal integers
  - alm
  - debug
  - decam
  - convert\_binary\_integer\_
  - cv\_oct\_
  - see conversion
- offline
  - see bulk I/O
- offset data
  - 5.4
- offset names
  - 1.5
- opening files
  - see initiation
- output
  - 4.4
  - dprint
  - dpunch
  - file\_output
  - print
  - discard\_output\_
  - ios\_
  - write\_list\_
  - (continued)
- output
  - (continued)
  - see I/O
  - dprint\_ (SWG)
- output conversion
  - see formatted I/O
- output line length
  - see terminal line length
- P
  - see interprocess communication
- packing
  - see archiving
  - see binding
- page faults
  - page\_trace
  - hcs\_\$reset\_working\_set (SWG)
- pages used
  - see metering
  - see records used
- paging
  - see storage system
- parameters
  - see argument lists
- parentheses
  - see command language
- parity
  - 8.5
- parsing
  - parse\_file\_
- passwords
  - see logging in
- path names
  - 1.5
  - 3.1
  - directory (Active Function)
  - get\_pathname (Active Function)
  - (continued)

- path names  
 (continued)  
 home\_dir (Active Function)  
 initiate  
 list  
 listnames (list)  
 listtotals (list)  
 list\_ref\_names  
 path (Active Function)  
 pd (Active Function)  
 print\_default\_wdir  
 print\_wdir  
 strip (Active Function)  
 wd (Active Function)  
 where  
 equal\_  
 expand\_path\_  
 get\_pdir\_  
 get\_wdir\_  
 hcs\_\$fs\_get\_path\_name  
 hcs\_\$initiate  
 hcs\_\$initiate\_count  
 hcs\_\$make\_seg  
 hcs\_\$star\_  
 hcs\_\$status\_  
 hcs\_\$truncate\_file  
 see linking
- permit list  
 see protection
- PL/I language  
 pl1  
 pl1\_abs
- pointer conversion  
 hcs\_\$fs\_get\_path\_name  
 hcs\_\$fs\_get\_ref\_name
- pointer data  
 5.4
- pointer generation  
 cu\_  
 hcs\_\$fs\_get\_seg\_ptr  
 hcs\_\$initiate  
 hcs\_\$initiate\_count  
 hcs\_\$make\_ptr  
 hcs\_\$make\_seg  
 find\_command\_ (SWG)
- preemption  
 user (SWG)  
 user\_info\_ (SWG)
- prepaging  
 hcs\_\$reset\_working\_set (SWG)
- prices  
 system\_info\_ (SWG)
- printer  
 see bulk I/O
- printing  
 4.1  
 4.4  
 dprint  
 dump\_segment  
 print  
 dprint\_ (SWG)
- procdef  
 see command processing
- procedures  
 2.1
- process creation  
 see creating processes
- process data segment  
 3.1
- process directories  
 3.1  
 pd (Active Function)  
 set\_search\_rules  
 get\_pdir\_  
 hcs\_\$make\_seg
- process groups  
 get\_group\_id\_
- process identifiers  
 get\_process\_id\_
- process information  
 user (Active Function)  
 user\_info\_  
 (continued)

Page 24

- process information
  - (continued)
  - see metering
  - user (SWG)
  - user\_info\_ (SWG)
- Process Initialization Table (PIT)
  - 3.1
- process interruption
  - 6.2
  - hold
  - program\_interrupt
  - release
  - start
  - default\_handler\_
  - timer\_manager\_
  - see conditions
  - standard\_default\_handler\_ (SWG)
- process overseer
  - user (SWG)
- process termination
  - logout
  - new\_proc
  - see logging out
- process termination fault
  - 13.6 (SWG)
- process\_dir\_dir
  - 3.1
- processes
  - new\_proc
  - see absentee usage
  - see logging in
  - see logging out
- processes created
  - user (SWG)
- program interruption
  - see process interruption
- program\_interrupt
  - see process interruption
- programming environment
  - Section 2
- programming languages
  - see languages
- programming standards
  - 2.5
- programming style
  - 2.5
- project names
  - 1.1
  - user (Active Function)
  - who
  - user\_info\_
- protection
  - 3.4
  - delete\_iacl\_dir
  - delete\_iacl\_seg
  - deleteacl
  - deletecacl (deleteacl)
  - list\_iacl\_dir
  - list\_iacl\_seg
  - listacl
  - listcacl (listacl)
  - set\_iacl\_dir
  - set\_iacl\_seg
  - setacl
  - setcacl (setacl)
  - copy\_acl\_
  - hcs\_\$add\_acl\_entries
  - hcs\_\$add\_dir\_acl\_entries
  - hcs\_\$delete\_acl\_entries
  - hcs\_\$delete\_dir\_acl\_entries
  - hcs\_\$fs\_get\_mode
  - hcs\_\$list\_acl
  - hcs\_\$list\_dir\_acl
  - hcs\_\$replace\_acl
  - hcs\_\$replace\_dir\_acl
  - see access control list
  - 13.4 (SWG)
  - set\_ring\_brackets (SWG)
  - get\_ring\_ (SWG)
  - hcs\_\$add\_dir\_inacl\_entries (SWG)
  - hcs\_\$add\_inacl\_entries (SWG)
  - hcs\_\$delete\_dir\_inacl\_entries (SWG)
  - (continued)



- protection  
     (continued)  
     hcs\_\$delete\_inacl\_entries (SWG)  
     hcs\_\$get\_dir\_ring\_brackets (SWG)  
     hcs\_\$get\_ring\_brackets (SWG)  
     hcs\_\$list\_dir\_inacl (SWG)  
     hcs\_\$list\_inacl (SWG)  
     hcs\_\$replace\_dir\_inacl (SWG)  
     hcs\_\$replace\_inacl (SWG)  
     hcs\_\$set\_ring\_brackets (SWG)  
     hcs\_\$set\_dir\_ring\_brackets (SWG)
- pseudo-device  
     4.2
- punched cards  
     4.1  
     4.4  
     5.2  
     dpunch  
     see bulk I/O
- push operator  
     12.2 (SWG)
- quits  
     see process interruption
- quitting  
     see process interruption
- quotas  
     resource\_usage  
     see storage quotas
- quoted strings  
     see command language
- radix conversion  
     decam  
     see conversion
- random number generators  
     random\_
- raw  
     see punched cards
- read-ahead  
     4.2
- ios\_
- reading cards  
     4.1  
     see bulk I/O  
     see punched cards
- ready messages  
     1.2  
     ready  
     ready\_off  
     ready\_on  
     cu\_
- ready mode  
     cu\_ (SWG)
- ready procedures  
     cu\_ (SWG)
- real data  
     5.4
- record quotas  
     see storage quotas
- redirecting output  
     console\_output  
     file\_output  
     see I/O streams  
     see output
- reference names  
     1.5  
     get\_pathname (Active Function)  
     initiate  
     list\_ref\_names  
     where  
     expand\_path\_  
     hcs\_\$fs\_get\_ref\_name  
     hcs\_\$fs\_get\_seg\_ptr  
     hcs\_\$initiate  
     hcs\_\$initiate\_count  
     hcs\_\$make\_ptr  
     hcs\_\$make\_seg  
     hcs\_\$terminate\_file  
     hcs\_\$terminate\_name  
     (continued)

Page 26

- reference names
  - (continued)
  - hcs\_\$terminate\_noname
  - hcs\_\$terminate\_seg
  - term\_
- referencing\_dir
  - set\_search\_rules
- rel\_link
  - see binding
- rel\_symbol
  - see binding
- rel\_text
  - see binding
- relative path names
  - expand\_path
  - see path names
- relative segments
  - see termination
- release
  - see error recovery
  - see process interruption
- relocation codes
  - 11.5 (SWG)
  - 11.7 (SWG)
- remote devices
  - see terminals
- removing segments
  - see deleting
  - see termination
- renaming
  - see directory entry names
- reserved characters
  - 5.2
  - see command language
- reserved names
  - 6.5
  - 8.1
- reserved segment numbers
  - hcs\_\$initiate
  - hcs\_\$terminate\_file
  - hcs\_\$terminate\_seg
- resource limits
  - resource\_usage
  - see accounting
  - see metering
  - see storage quotas
- resource usage
  - resource\_usage
- restarting
  - start
- return operator
  - 12.2 (SWG)
- ring brackets
  - see protection
  - 13.4 (SWG)
  - set\_ring\_brackets (SWG)
  - hcs\_\$get\_dir\_ring\_brackets (SWG)
  - hcs\_\$get\_ring\_brackets (SWG)
  - hcs\_\$set\_dir\_ring\_brackets (SWG)
  - hcs\_\$set\_ring\_brackets (SWG)
- rings
  - see protection
  - 13.4 (SWG)
  - set\_ring\_brackets (SWG)
  - get\_ring\_ (SWG)
  - hcs\_\$get\_dir\_ring\_brackets (SWG)
  - hcs\_\$get\_ring\_brackets (SWG)
  - hcs\_\$set\_dir\_ring\_brackets (SWG)
  - hcs\_\$set\_ring\_brackets (SWG)
- root directory
  - 3.1
- runtime
  - see programming environment
- runtime storage management
  - see storage management

## safety switch

3.3

safety\_sw\_off

safety\_sw\_on

hcs\_\$get\_safety\_sw (SWG)

hcs\_\$set\_safety\_sw (SWG)

hcs\_\$set\_safety\_sw\_seg (SWG)

## schedules

system\_info\_ (SWG)

## scratch segments

see temporary segments

## SDB

see Stream Data Block

## search rules

3.2

change\_default\_wdir

change\_wdir

print\_default\_wdir

print\_wdir

set\_search\_dirs

set\_search\_rules

where

change\_wdir\_

get\_wdir\_

hcs\_\$make\_ptr

see default working directory

see working directory

hcs\_\$get\_search\_rules (SWG)

hcs\_\$initiate\_search\_rules (SWG)

## searching

hcs\_\$fs\_get\_path\_name

hcs\_\$make\_ptr

see dynamic linking

see search rules

hcs\_\$get\_search\_rules (SWG)

hcs\_\$initiate\_search\_rules (SWG)

## secondary storage device

3.3

## segment access modes

delete\_iacl\_seg

list\_iacl\_seg

set\_iacl\_seg

(continued)

## segment access modes

(continued)

hcs\_\$add\_acl\_entries

hcs\_\$delete\_acl\_entries

hcs\_\$list\_acl

hcs\_\$replace\_acl

hcs\_\$add\_inacl\_entries (SWG)

hcs\_\$delete\_inacl\_entries (SWG)

## segment addressing

see pointer generation

## segment attributes

3.3

deleteacl

list

listnames (list)

listtotals (list)

listacl

safety\_sw\_off

safety\_sw\_on

setacl

status

hcs\_\$set\_bc

hcs\_\$set\_bc\_seg

hcs\_\$star\_

hcs\_\$status\_

see length of segments

see protection

set\_max\_length (SWG)

set\_ring\_brackets (SWG)

hcs\_\$get\_author (SWG)

hcs\_\$get\_bc\_author (SWG)

hcs\_\$get\_max\_length (SWG)

hcs\_\$get\_ring\_brackets (SWG)

hcs\_\$get\_safety\_sw (SWG)

hcs\_\$set\_max\_length (SWG)

hcs\_\$set\_max\_length\_seg (SWG)

hcs\_\$set\_ring\_brackets (SWG)

hcs\_\$set\_safety\_sw (SWG)

hcs\_\$set\_safety\_sw\_seg (SWG)

## segment copying

see copying

## segment creation

see creating segments

Page 28

- segment deletion
  - see deleting
- segment formats
  - 5.5
- segment formatting
  - indent
  - make\_peruse\_text
- segment initiation
  - see initiation
- segment length
  - see length of segments
- segment name operations
  - pd (Active Function)
- segment names
  - 1.5
  - 8.1
  - see directory entry names
- segment numbers
  - list\_ref\_names
- segment packing
  - see archiving
  - see binding
- segment referencing
  - see initiation
  - see linking
  - see pointer generation
- segment renaming
  - see directory entry names
- segment termination
  - see termination
- segment truncation
  - see truncation
- segments
  - 5.3
  - see creating segments
  - see deleting
  - (continued)
- segments
  - (continued)
  - see directory entry names
  - see initiation
  - see length of segments
  - see protection
  - see storage system
  - see temporary segments
  - see termination
- semaphores
  - see interprocess communication
  - set\_lock\_ (SWG)
- setting bit counts
  - see bit counts
- seven-punch cards
  - 4.4
  - dpunch
  - see punched cards
- shifts
  - system\_info\_ (SWG)
- short return operator
  - 12.2 (SWG)
- shriek names
  - see unique strings
- shutdown time
  - system\_info\_ (SWG)
- signals
  - see conditions
- simulated faults
  - 13.6 (SWG)
- simulation
  - random\_
- sleeping
  - timer\_manager\_
- snapping links
  - see dynamic linking

- sorting
  - archive\_sort
  - reorder\_archive
  - sort\_file
- source maps
  - 11.5 (SWG)
- space saving
  - see archiving
  - see binding
- special active function
  - user (Active Function)
- special characters
  - 1.3
  - see character codes
- special sessions
  - see logging in
- special subsystems
  - Section 7
- specifiers
  - see descriptors
- spooling
  - see bulk I/O
- stack frame pointer
  - cu\_
- stack frames
  - debug
  - trace\_stack
  - 12.1 (SWG)
- stack header
  - 12.1 (SWG)
- stack management
  - listen\_ (SWG)
- stack referencing
  - debug
  - trace\_stack
  - cu\_
- stack segment
  - 3.1
  - 12.1 (SWG)
- stacks
  - see stack frames
  - see stack segments
- Standard Data Formats and Codes
  - Section 5
- standard tape formats
  - see magnetic tapes
- standards
  - 2.5
  - 11.7 (SWG)
- star convention
  - 1.5
  - fs\_chname
  - equal\_
  - hcs\_\$star\_
- start
  - see error recovery
  - see process interruption
- start up
  - 1.2
  - exec\_com
  - see logging in
- start\_up.ec
  - see start up
- static linking
  - see binding
  - see linkage sections
  - see linking
- static storage
  - new\_proc
  - see storage management
- status
  - check\_info\_segs
  - help
  - how\_many\_users
  - (continued)

Page 30

- status
  - (continued)
  - list
  - listnames (list)
  - listtotals (list)
  - list\_abs\_requests
  - peruse\_text
  - status
  - who
  - hcs\_\$star\_
  - hcs\_\$status\_
  - see I/O status
  - hcs\_\$get\_author (SWG)
  - hcs\_\$get\_bc\_author (SWG)
  - hcs\_\$get\_dir\_ring\_brackets (SWG)
  - hcs\_\$get\_max\_length (SWG)
  - hcs\_\$get\_ring\_brackets (SWG)
  - hcs\_\$get\_safety\_sw (SWG)
- status codes
  - 4.2
  - 6.1
  - 6.4
  - com\_err\_
  - unpack\_system\_code\_
  - see I/O system interface
  - error\_table\_compiler (SWG)
  - convert\_status\_code\_ (SWG)
- status formats
  - 4.2
- status message
  - find\_command\_ (SWG)
- status messages
  - 6.4
  - reprint\_error
  - active\_fnc\_err\_
  - com\_err\_
  - command\_query\_
  - convert\_status\_code\_ (SWG)
  - condition\_interpreter\_ (SWG)
- status tables
  - 6.4
  - error\_table\_compiler (SWG)
  - convert\_status\_code\_ (SWG)
- storage allocation
  - see storage management
- storage hierarchy
  - see directories
  - see storage system
- storage management
  - area\_
  - see address reuse
  - see archiving
  - see deleting
  - see directories
  - see I/O
  - see length of segments
  - see segments
  - see storage quotas
  - alloc\_ (SWG)
  - area\_ (SWG)
  - area\_assign\_ (SWG)
  - freen\_ (SWG)
  - get\_system\_free\_area\_ (SWG)
  - tssi\_ (SWG)
- storage quotas
  - getquota
  - movequota
  - hcs\_\$quota\_get (SWG)
- storage system
  - Section 3
  - 4.2
  - see directory hierarchy
- storage system I/O
  - 4.3
  - console\_output
  - file\_output
- storage\_quotas
  - hcs\_\$quota\_move (SWG)
- Stream Data Block (SDB)
  - 4.6
  - see I/O system interface
- stream names
  - 1.5
  - 8.1

- streams
  - see I/O streams
- structure data
  - 5.4
- subroutines
  - 2.1
  - Section 10
  - see procedures
- subsystems
  - 1.2
  - Section 7
  - 7.2
  - see languages
- suffixes
  - 8.1
  - strip (Active Function)
  - strip\_entry (Active Function)
  - suffix (Active Function)
- symbol blocks
  - 11.5 (SWG)
- symbol sections
  - 11.5 (SWG)
- symbol tables
  - stu\_
- symbolic debugging
  - debug
  - stu\_
  - see debugging tools
- synchronization
  - 4.2
  - ios\_
  - see interprocess communication
- synonyms
  - syn
  - see directory entry names
  - see I/O system interface
- syntax analysis
  - parse\_file\_
- system libraries
  - 3.1
  - see libraries
  - see search rules
- system load
  - how\_many\_users
  - who
  - system\_info\_ (SWG)
- system parameters
  - system\_info\_ (SWG)
- system status
  - help
  - how\_many\_users
  - list\_abs\_requests
  - page\_trace
  - peruse\_text
  - print\_motd
  - who
  - hcs\_\$reset\_working\_set (SWG)
- system\_control\_dir
  - 3.1
- system\_library\_auth\_maint
  - 3.1
- system\_library\_standard
  - 3.1
- tapes
  - see magnetic tapes
- teletype model 33,35,37,38
  - see terminals
- temporary files
  - see temporary segments
- temporary segments
  - hcs\_\$make\_seg
  - unique\_chars\_
  - see process directories
  - see storage management
  - see unique names

Page 32

temporary storage  
  see process directories  
  see storage management  
  see temporary segments

terminal-line length  
  line\_length

terminals  
  1.2  
  1.3  
  4.1  
  console\_output  
  line\_length  
  set\_com\_line  
  user (Active Function)  
  read\_list\_  
  tw\_  
  user\_info\_  
  write\_list\_  
  see I/O

terminating processes  
  see process termination

termination  
  logout  
  new\_proc  
  terminate  
  hcs\_\$terminate\_file  
  hcs\_\$terminate\_name  
  hcs\_\$terminate\_noname  
  hcs\_\$terminate\_seg  
  term\_  
  see cancelling  
  see process termination

text editing  
  see editing

text formatting  
  runoff  
  runoff\_abs

text scanning  
  compare\_ascii  
  compare\_ascii\_  
  parse\_file\_

text sections  
  11.2 (SWG)

text sorting  
  see sorting

time  
  date\_time (Active Function)  
  date\_time (Active Function)  
  hour (Active Function)  
  minute (Active Function)  
  time (Active Function)  
  clock\_  
  convert\_date\_to\_binary\_  
  date\_time\_  
  decode\_clock\_value\_  
  timer\_manager\_  
  see metering

transfer vector  
  4.6

transfer vectors  
  11.2 (SWG)

translators  
  see languages

trap pairs  
  11.3 (SWG)

traps  
  see faults

traps on first reference  
  11.4 (SWG)

truncation  
  truncate  
  hcs\_\$truncate\_file  
  hcs\_\$truncate\_seg

type conversion  
  see conversion

typing conventions  
  1.3  
  abbrev  
  see canonicalization



- udd
  - see user\_dir\_dir
- unique Identifiers
  - 3.3
- unique names
  - hcs\_\$make\_seg
- unique strings
  - unique (Active Function)
  - unique\_bits\_
  - unique\_chars\_
- unlinking
  - unlink
  - delete\_
  - see deleting
  - see termination
- unsnapping
  - terminate\_refname (terminate)
  - terminate\_segno (terminate)
  - terminate\_single\_refname  
(terminate)
  - term\_
  - see termination
- unsnapping links
  - see termination
- unwinding
  - 6.3
  - listen\_ (SWG)
  - unwinder\_ (SWG)
- usage data
  - user (Active Function)
  - user\_info\_
  - see metering
- usage limits
  - start\_governor\_ (SWG)
- usage measures
  - see metering
- useless output
  - program\_interrupt
  - discard\_output\_
- user attributes
  - user (SWG)
  - user\_info\_ (SWG)
- user names
  - 1.1
  - 3.4
  - user (Active Function)
  - who
  - user\_info\_
- user parameters
  - user (Active Function)
  - user (SWG)
  - user\_info\_ (SWG)
- user weight
  - user (Active Function)
  - user\_info\_
- user\_dir\_dir
  - 3.1
- user\_i/o
  - see I/O streams
  - see terminals
- user\_input
  - see I/O streams
- user\_output
  - see I/O streams
- users
  - how\_many\_users
  - who
- V
  - see interprocess communication
- validation level
  - cu\_
  - see protection
  - 13.4 (SWG)
- variable length argument list
  - cu\_

Page 34

varying string data  
5.4

VII-punch cards  
see seven-punch cards

virtual memory  
see directory hierarchy  
see storage system

waiting  
timer\_manager\_  
hcs\_\$wakeup (SWG)  
ipc\_ (SWG)

wakeups  
timer\_manager\_  
hcs\_\$wakeup (SWG)  
ipc\_ (SWG)

wdir  
see working directory

working directory  
change\_wdir  
print\_search\_rules  
print\_wdir  
set\_search\_rules  
walk\_subtree  
wd (Active Function)  
change\_wdir\_  
expand\_path\_  
get\_wdir\_  
see default working directory

working set  
page\_trace  
hcs\_\$reset\_working\_set (SWG)

workspace  
4.2  
ios\_

write-behind  
4.2  
ios\_

writing to multiple I/O streams  
see broadcasting

(END)