SERIES 60 (LEVEL 68)

MULTICS PROGRAMMER'S MANUAL
COMMANDS AND ACTIVE FUNCTIONS

SUBJECT

Additions and Changes to Standard Multics Commands and Active Functions

SPECIAL INSTRUCTIONS

This is the third revision to AG92, dated January 1979.

This revision replaces Addendum A, B, and C of AG92, Revision 2.

The following commands are obsolete and have been moved to Appendix A in this manual for user convenience.

fs_chname	print_trans_search_rules
mail	set_trans_search_rules

The following commands appear in this manual for the first time:

archive_table	no_save_on_disconnect
convert_characters	save_on_disconnect
hangup	

SOFTWARE SUPPORTED

Multics Software Release 8.0

ORDER NUMBER

AG92-03

December 1979

Honeywell

SERIES 60 (LEVEL 68)

MULTICS PROGRAMMER'S MANUAL COMMANDS AND ACTIVE FUNCTIONS ADDENDUM A

SUBJECT

Additions and Changes to Standard Multics Commands and Active Functions

SPECIAL INSTRUCTIONS

This is the first addendum to AG92, Revision 3, dated December 1979.

Insert the attached pages into the manual according to the collating instructions on the back of this cover.

Note:

Insert this cover after the manual cover to indicate the updating of the document with Addendum A.

SOFTWARE SUPPORTED

Multics Software Release 8.0

ORDER NUMBER

AG92-03A

February 1980

26940
1380
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

To update the manual, remove old pages and insert new pages as follows:

Remove	Insert
	3-240.1, 3-240.2 3-240.3, 3-240.4 3-240.5, 3-240.6 3-240.7, blank
3-559 through 3-568	3-559 through 3-568
3-619, 3-620	3-619, blank 3-619.1 through 3-619.32 3-619.33, 3-620
4-3, 4-4	4-3, blank 4-3.1, 4-4
	4-18.1, blank
4-19, 4-20	4-19, blank 4-19.1, 4-20

File No.: 1L13

2/80

AG92-03A

The MPM Commands is organized into four sections. Section 1 contains general information on the use of the manual as well as a description, and definition, of format. Section 2 contains a list of the Multics system commands and active function repertoire, arranged functionally. Section 3 contains descriptions of selected standard Multics system commands and active functions, including the syntax of each, arranged alphabetically. Section 4 describes requests used to gain access to the system.

The MPM Subroutines is organized into three sections. Section 1 contains a list of the subroutine repertoire, arranged functionally. Section 2 contains descriptions of the standard Multics subroutines, including the declare statement, the calling sequence, and usage of each. Section 3 contains the descriptions of the I/O modules.

The MPM Subsystem Writers' Guide is a reference of interest to compiler writers and writers of sophisticated subsystems. It documents user-accessible modules that allow the user to bypass standard Multics facilities. The interfaces thus documented are a level deeper into the system than those required by the majority of users.

Examples of specialized subsystems for which construction would require reference to the MPM Subsystem Writers' Guide are:

- A subsystem that precisely imitates the command environment of some system other than Multics.
- A subsystem intended to enforce restrictions on the services available to a set of users (e.g., an APL-only subsystem for use in an academic class).
- A subsystem that protects some kind of information in a way not easily expressible with ordinary access control lists (e.g., a proprietary linear programming system, or an administrative data base system that permits access only to program-defined, aggregated information such as averages and correlations).

The MPM Peripheral I/O manual contains descriptions of commands and subroutines used to perform peripheral I/O. Included in this manual are commands and subroutines that manipulate tapes and disks as I/O devices.

The MPM Communications I/O manual contains information about the Multics Communication System. Included are sections on the commands, subroutines, and I/O modules used to manipulate communications I/O. Special purpose communications I/O, such as binary synchronous communication, is also included.

Several cross-reference facilities help locate information:

- Each manual has a table of contents that identifies the material (either the name of the section and subsection or an alphabetically ordered list of command and subroutuine names) by page number.
- Each manual contains an index that lists items by name and page number.
- Individual program descriptions reference other programs in the same and other manuals.

CONTENTS

PREFACE

Primary reference for user and subsystem programming on the Multics system is contained in six manuals. The manuals are collectively referred to as the Multics Programmers' Manual (MPM). Throughout this manual, references are frequently made to the MPM. For convenience, these references will be as follows:

Document	Referred To In Text As
Reference Guide (Order No. AG91)	MPM Reference Guide
Commands and Active Functions (Order No. AG92)	MPM Commands
Subroutines (Order No. AG93)	MPM Subroutines
Subsystem Writers' Guide (Order No. AK92)	MPM Subsystem Writers' Guide
Peripheral Input/Output (Order No. AX49)	MPM Peripheral I/O
Communications Input/Output (Order No. CC92)	MPM Communications I/O

The MPM Reference Guide contains general information about the Multics command and programming environments. It also defines items used throughout the rest of the MPM and, in addition, describes such subjects as the command language, the storage system, and the input/output system.

 File No.: 1L13

SECTION 1

MANUAL USE AND TERM DEFINITION

This section deals with the proper use of this manual, a description of the format used, and a general definition of terms. New users are particularly encouraged to read this section.

DESCRIPTION OF MANUAL FORMAT

Section 2 contains a breakdown by function of the programs described in this manual. Section 3 contains an alphabetized listing of selected standard Multics system commands and active functions. Section 4 contains descriptions of those commands used to gain access to the Multics system.

Each description in this manual closely parallels the info segment format available online by invoking the help command. Each description provides, minimally, the long (and short) name, syntax line, and function of the program. Standard headings, in the order in which they appear when present, are as follows:

SYNTAX AS A COMMAND:
SYNTAX AS AN ACTIVE FUNCTION:
FUNCTION:
ARGUMENTS:
CONTROL ARGUMENTS:
ACCESS REQUIRED:
NOTES:
EXAMPLES:

All headings appear in capital letters followed by a colon (:).

Some descriptions contain unique headings such as "LIST OF KEYS", or "LIST OF REQUESTS".

Syntax lines give the order of required and optional arguments accepted by a command or active function. Optional portions of syntax are enclosed in braces ({}). The syntax for

active functions is always shown enclosed in brackets ([ ]), which are required for active function use. To indicate that a command accepts more than one of a specific argument, the argument name is pluralized (e.g., paths, {paths}, {-control_args}).

NOTE: Keep in mind the difference between a plural argument name that is enclosed in braces (i.e., optional) and one that is not (i.e., required). If the plural argument is enclosed in braces, clearly no argument of that type need be specified. However, if there are no braces, at least one argument of that type must be specified. Thus "paths" in a syntax line could also be written as:
path1 {path2 ... pathn}
The convention of using "paths" rather than the above is merely a method of saving space.

Different arguments that must be specified in pairs are numbered (e.g., xxx1 yyy1 {... xxxn yyyn}). To indicate that the same generic argument must be specified in pairs, the arguments are indicated with letters and numbers (e.g., pathA1 pathB1 {... pathAn pathBn}).

Some of the standard arguments accepted by active functions are:

str — any character string.

num — any character string that represents a number, either decimal or binary. Examples are integers (5, 1024, or 101b), real numbers (1.37 or -10.01b), and floating-point numbers (1.3e+4 or 1010.001e+5b).

dt — any date-time character string in a form acceptable to the convert_date_to_binary_ subroutine described in MPM Subroutines. Examples are "4/25/79 noon est Sun", "November 7", "7:30 pm 10 June 1980", and "midnight".

star_name is any pathname or User_id conforming to the star convention, described under "Star Names" in the MPM Reference Guide.

Arguments, when present, are listed with a brief description and the default value if any. To indicate one of a group of the same arguments, an "i" is added to the argument name (e.g., pathi, User_idi).

The list of control arguments gives the possible values for -control_args in the syntax line. Both the long and short names

are given when both exist. Those that take values (the following argument, as in "-path >udd>m>Foo") generally indicate these values as follows:

STR — any character string; individual command descriptions indicate any restrictions (e.g., must be chosen from specified list; must not exceed 136 characters).

N — number; individual command descriptions indicate whether it is octal or decimal and any other restrictions (e.g., cannot be greater than 4).

DT — date-time character string in a form acceptable to the convert_date_to_binary_ subroutine described in the MPM Subroutines.

ID — numerical request identifier as described in the MPM Reference Guide.

path — pathname of a storage system entry; unless otherwise indicated, it can be either a relative or an absolute pathname.

The lines below are samples of control arguments that take values:

```
-access_name STR, -an STR
-ring N, -rg N
-date DT, -dt DT
-home_dir path, -hd path
```

The "NOTES" section is used to provide additional information and cross-reference with other manuals.

Examples, while not extensive, attempt to provide additional help and insight on the proper use and formatting of commands and active functions. Lines input by the user are preceded by an exclamation mark (!). Examples of command use show the response a user can expect to see on the terminal. Examples of active function use show the return value substituted by the command processor for the active string.

GENERAL DEFINITION OF A COMMAND

Commands are invoked at the beginning of command lines typed to the system, and immediately following unquoted, unbracketed semicolon (;). Some of the uses commands have are displaying information on the user's terminal, archiving data, and compiling programs. Each command has a specific purpose. The default action performed by a command is generally the most common use of

the command. Many commands have optional arguments that refine the actions that are performed. When invoked correctly, most commands either print information or modify storage system entries, but not both.

GENERAL DEFINITION OF AN ACTIVE FUNCTION

Active functions are most frequently used to shorten the amount of typing required to invoke commands. An active function is invoked inside an active string, a string surrounded by brackets ([]), which is replaced by a character string return value before the command line containing it is executed. Active functions are often used in conjunction with the exec_com, abbrev, and do commands to implement command-language macros.

When several commands are specified on a line, the first is executed before active functions in the second are expanded, and so on. Therefore, the execution of a command can affect the values of active functions appearing later in the line.

Examples of Command vs. Active Function Use

Many programs can be invoked as either command or active function. The format of the active function return string is often slightly different from the command's printed output. To illustrate this difference, examples using the status command and active function are shown below. In these examples, and all interactive examples throughout this manual, lines typed by the user are preceded with an exclamation mark (!).

```
! status report1 -nm
  names:    report_first_quarter.runoff
            report1.runoff
            report1
```

versus the corresponding status active function and response:

```
! string [status report1 -nm]
  report_first_quarter.runoff report1.runoff report1
```

Errors

Commands report errors by signalling command_error and printing a message. Messages that do not begin with "Warning:" usually terminate execution of the command, though later commands on the same line are subsequently executed.

Active functions report errors by signalling active_function_error. Default action is to print a message and return to command level. The user should respond by typing:

release

to abort the command line, and then issue a corrected line.

The command_error and active_function_error conditions are further described in the MPM Reference Guide.

GENERAL INFORMATION

Users are encouraged to take advantage of the information available in the manual index. The index alphabetically lists programs by name and subject (i.e., segment, date/time, resource limits, etc.). Cross references among program descriptions help to locate programs applicable to a given task.

SECTION 2

REFERENCE TO COMMANDS AND ACTIVE FUNCTIONS

The Multics commands and active functions documented in this manual are grouped below according to function. All commands and active functions are listed under at least one functional heading. Some commands and active functions are listed under more than one heading. Descriptions appear in Section 3 in alphabetical order.

FUNCTIONAL HEADINGS OF COMMANDS

Access to the System

dial
enter
enterp
login
logout

Storage System, Creating and Editing Segments

adjust_bit_count
compare_ascii
qedx
runoff

canonicalize
edm
indent
program_interrupt
runoff_abs
set_bit_count
sort_seg

Storage System, Segment Manipulation

adjust_bit_count
archive
compare
compare_ascii
copy
copy_file
create
damaged_sw_off
damaged_sw_on
delete
link
merge_ascii
move
set_bit_count
sort_seg
tape_archive
truncate
unlink
vfile_adjust
volume_dump_switch_off
volume_dump_switch_on

Storage System, Directory Manipulation

add_name
cancel_retrieval_request
copy_dir
create_dir
delete_dir
delete_name
enter_retrieval_request
link
list_retrieval_requests
list
move_dir
rename
safety_sw_off
safety_sw_on
status
tape_archive
unlink
vfile_status
volume_dump_switch_off
volume_dump_switch_on

Storage System, Access Control

check_iacl
copy_acl
copy_iacl_dir
copy_iacl_seg
delete_acl
delete_iacl_dir
delete_iacl_seg
list_accessible
list_acl
list_not_accessible
list_iacl_dir
list_iacl_seg
set_acl
set_iacl_dir
set_iacl_seg

Storage System, Address Space Control

add_search_paths
add_search_rules
attach_lv
change_default_wdir
change_wdir
delete_search_paths
print_proc_auth
print_search_paths
print_search_rules
print_wdir
set_search_paths
set_search_rules

delete_search_rules
detach_lv
get_system_search_rules
initiate
list_ref_names
new_proc
print_default_wdir
terminate
terminate_refname
terminate_segno
terminate_single_refname
where
where_search_paths

Formatted Output Facilities

cancel_daemon_request
dprint
dpunch
dump_segment
list_daemon_requests
move_daemon_request
overlay
print
runoff
runoff_abs

Language Translators, Compilers, Assemblers, and Interpreters

apl
basic
bind
cancel_cobol_program
cobol
cobol_abs
create_data_segment
display_cobol_run_unit
expand_cobol_source
fast
format_cobol_source
fortran
fortran_abs
indent
new_fortran
old_fortran
pl1
pl1_abs
profile
qedx
run_cobol
runoff
runoff_abs
set_fortran_common
stop_cobol_run

Object Segment Manipulation

archive
bind
date_compiled

Debugging and Performance Monitoring Facilities

attach_audit
change_error_mode
cumulative_page_trace
debug
display_audit_file
display_pl1io_error
dump_segment
general_ready
page_trace
probe
profile
progress
ready
ready_off
ready_on
repeat_query
reprint_error
resolve_linkage_error
trace
trace_stack

Input/Output System Control

assign_resource
cancel_resource
cancel_daemon_request
close_file
console_output
copy_cards
copy_file
discard_output
display_pl1io_error
dprint
dpunch
file_output
io_call
line_length
list_daemon_requests
list_resource_types
list_resources
print
print_attach_table
print_request_types
reserve_resource
tape_archive
unassign_resource
vfile_adjust
vfile_status

Command Level Environment

abbrev
add_search_paths
add_search_rules
answer
attach_audit
change_default_wdir
change_error_mode
change_wdir
console_output
delete_search_paths
delete_search_rules
display_audit_file
do
exec_com
fast
file_output
if
general_ready
get_system_search_rules
line_length
memo
new_proc
on
print_default_wdir
print_search_paths
print_search_rules
print_wdir
program_interrupt
ready
ready_off
ready_on
release
repeat_query
reprint_error
resolve_linkage_error
run
set_search_paths
set_search_rules
start
stop_run
where_search_paths

Communication Among Users

accept_messages
defer_messages
delete_message
immediate_messages
print_auth_names
print_mail
print_messages
read_mail
send_mail
send_message
send_message_acknowledge
send_message_express
send_message_silent
who

Communication with the System

cancel_retrieval_request
check_info_segs
damaged_sw_off
damaged_sw_on
help
how_many_users
enter_retrieval_request
list_help
list_retrieval_requests
move_abs_request
print_motd
volume_dump_switch_off
volume_dump_switch_on
who

Accounting

get_quota
move_quota
resource_usage

Control of Absentee Computations

cancel_abs_request
cobol_abs
enter_abs_request
fortran_abs
how_many_users
list_abs_requests
move_abs_request
pl1_abs
runoff_abs
who

Miscellaneous Tools

calc
calendar
canonicalize
decode
encode
manage_volume_pool
memo
progress
walk_subtree

FUNCTIONAL HEADINGS OF ACTIVE FUNCTIONS

Arithmetic
Character String
Condition Handling
Conversion
Date and Time
Input/Output
Logical
Miscellaneous
Pathname Manipulation
Question Asking
Storage System Attributes
Storage System Names
User/Process Information

Arithmetic

ceil
divide
floor
max
min
minus
mod
plus
quotient
times
trunc

Character String

after
before
bool
collate
collate9
copy_characters
decat
format_line
high
high9
index
index_set
length
low
lower_case
ltrim
picture
reverse
reverse_after
reverse_before
reverse_decat
reverse_index
reverse_search
reverse_verfiy
rtrim
search
string
substr
translate
underline
unique
upper_case
verify

Condition Handling

on

Conversion

binary
decimal
hexadecimal
octal

Date and Time

date
date_time
day
day_name
hour
long_date
minute
month
month_name
time
year

Input/Output

io_call

Logical

and
equal
exists
greater
less
nequal
ngreater
nless
not
or

Miscellaneous

contents
default

Pathname Manipulation

directory
entry
equal_name
path
strip
strip_entry
suffix

Question Asking

query
response

Storage System Attributes

lv_attached
status

Storage System Names

branches
default_wdir
directories
entries
files
get_pathname
home_dir
links
master_directories
msfs
nondirectories
nonfiles
nonmaster_directories
nonmsfs
nonnull_links
nonsegments
nonzero_files
nonzero_msfs
nonzero_segments
null_links
process_dir
segments
working_dir
zero_segments

User/Process Information

have_mail
last_message
last_message_sender
last_message_time
severity
system
user

SECTION 3

COMMANDS

This section contains descriptions of selected Multics commands and active functions, presented in alphabetical order.

SYNTAX AS A COMMAND:

abbrev

FUNCTION: provides the user with a mechanism for abbreviating parts of (or whole) command lines in the normal command environment.

NOTES: The abbrev command sets up a special command processor that is called for each command line input to the system until abbrev processing is explicitly reverted. The abbrev command processor checks each input line to see if it is an abbrev request line (recognized by a period (.) as the first nonblank character of the line) and, if so, acts on that request. (Requests are described below under "List of Control Requests.") If the input line is not an abbrev request line and abbreviations are included in the line, the abbreviations are expanded once and the expanded string is passed on to the normal Multics command processor. The abbrev command processor is, therefore, spliced in between the listener and the normal command processor. Note that abbreviations are expanded only once; i.e., abbreviations cannot be nested.

The abbrev command is driven by a user profile segment that contains the user's abbreviations and other information pertinent to execution on the user's behalf. The profile segment resides (by default) in the user's home directory. If the profile segment is not found, it is created and initialized with the name Person_id.profile where Person_id is the login name of the user. For example, if the user Washington logs in under the States project, the default profile segment is:

>user_dir_dir>States>Washington>Washington.profile

The profile segment being used by abbrev can be changed at any time with the .u control request (see below) to any profile segment in the storage system hierarchy to which the user has appropriate access. The entryname of a profile segment must have the suffix profile. A new profile segment can be created by specifying a nonexistent segment to the .u control request. The segment is then created and initialized as a profile segment, assuming the user has the necessary access. The user must be careful not to delete or terminate the segment that is

currently being used as his profile unless he first quits out of abbrev by issuing the .q control request (see below).

The user can suppress expansion of a particular string in a command line by enclosing it within quotes ("). To suppress expansion of an entire command line, see the .<space> control request.

A user might want to include the invocation of the abbrev command in a start_up.ec segment so that he is automatically able to abbreviate whenever he is logged in. See the MPM Reference Guide for a definition of start_up.ec.

NOTES ON CONTROL REQUESTS: An abbrev request line has a period (.) as the first nonblank character of the line. An abbrev request line, with the exception of the .s and .<space> requests, is neither checked for embedded abbreviations nor (even in part) passed on to the command processor. If the command line is not an abbrev request line, abbrev expands it and passes it on to the current command processor.

LIST OF CONTROL REQUESTS: The character immediately after the period of an abbrev request line is the name of the request. The following requests are recognized:

.a <abbr> <rest of line>
add the abbreviation <abbr> to the current profile segment. It is an abbreviation for <rest of line>. Note that the <rest of line> string can contain any characters. If the abbreviation already exists, the user is asked whether to redefine it. The user must respond with "yes" or "no". The abbreviation must be no longer than eight characters and must not contain break characters.

.ab <abbr> <rest of line>
add an abbreviation that is expanded only if found at the beginning of a line or directly following a semicolon (;) in the expanded line. In other words, this is an abbreviation for a command name.

.af <abbr> <rest of line>
add an abbreviation to the profile segment and force it to overwrite any previous abbreviation with the same name. The user is not asked whether to redefine the abbreviation.

.abf <abbr> <rest of line>
add an abbreviation that is expanded only at the beginning of a line and force it to replace any previous abbreviation with the same name. The user is not asked whether to redefine the abbreviation.

.d <abbr1> ... <abbrn>
delete the specified abbreviations from the current profile segment.

.f
enter a mode (the default mode) that forgets each command line after executing it. See the .r and .s requests.

.l <abbr1> ... <abbrn>
list the specified abbreviations and the strings they stand for. If no abbreviations are specified, all abbreviations in the current profile segment are listed.

.la <letter1> ... <lettern>
list all abbreviations starting with the specified letters. <letteri> is expected to be a single character. If no letters are specified, all abbreviations in the current profile segment are listed.

.q
quit using the abbrev command processor. This request resets the command processor to the one in use before invoking abbrev and, hence, prevents any subsequent action on the part of abbrev until it is explicitly invoked again.

.r
enter a mode that remembers the last line expanded by abbrev. See the .f and .s requests.

.s <rest of line>
show the user how <rest of line> would be expanded but do not execute it. The .s request with no arguments shows the user the last line expanded by abbrev and is valid only if abbrev is remembering lines. See the .f and .r requests.

.u <profile>
specify to abbrev the pathname of a profile segment to use. <profile segment> becomes the current working profile segment. The user needs "r" access to use the profile segment and "w" access to add and delete abbreviations.

.p
print the name of the profile segment being used.

.<space> <rest of line>
pass <rest of line> on to the current command processor without expanding it. Using this request, the user can issue a command line that contains abbreviations that are not to be expanded.

NOTES ON BREAK CHARACTERS: When abbrev expands a command line, it treats certain characters as special or break characters. An abbreviation cannot contain break characters. Any character string that is less than or equal to eight characters long and is bounded by break characters is a candidate for expansion. The string is looked up in the current profile segment and, if it is found, the expanded form is placed in (a copy of) the command line to be passed on to the normal command processor.

The characters that abbrev treats as break characters are:

newline	
formfeed	
vertical tab	
horizontal tab	
space	
quote	"
dollar sign	$
apostrophe	'
grave accent	`
period	.
semicolon	;
vertical bar	|
parentheses	( )
less than	<
greater than	>
brackets	[ ]
braces	{ }

EXAMPLES: Suppose that a user is typing the segment name suffix fortran repeatedly while editing FORTRAN source segments. The user might wish to abbreviate the suffix to "ft" as follows:

Invoke the abbrev command:

```
! abbrev
```

Define the abbreviation:

```
! .a ft fortran
```

Now that "ft" is defined invoke a text editor to create or edit the source segment:

```
! qedx
! r sample.ft
```

In order to write out one of the segments from qedx by a different name, the user must type the expanded name since the qedx command (and not the abbrev command processor) is intercepting all terminal input. For example, after editing sample.fortran the user might want to write out the changed version as example.fortran. This can be done by typing to qedx:

```
! w example.fortran
```

If instead the user types:

```
! w example.ft
```

a segment is created by exactly that name (example.ft). In this case, if the user tries to print the segment while at command level (by typing "print example.ft"), the abbrev processor expands the command line and the print command looks for a segment named example.fortran; since no such segment exists, the print command responds with an error message.

SYNTAX AS A COMMAND:

am {destination} {-control_args}

FUNCTION: initializes or reinitializes the user's process for accepting messages sent by the send_message command and notifications of the form "You have mail." sent by the send_mail command.

ARGUMENTS:

destination
is of the form Person_id.Project_id to specify a mailbox. The default is the user's default mailbox. If destination contains either < or >, it is assumed to be the pathname of a mailbox.

CONTROL ARGUMENTS:

-brief, -bf
prevents accept_messages from informing the user that it is creating a mailbox and prints messages in short format (see the -short control argument below).

-pathname path, -pn path
specifies a mailbox by pathname. The mbx suffix is assumed. This control argument and the destination argument are mutually exclusive.

-call {cmdline}
when the message is received, instead of printing it in the default format, accept_messages calls the command processor with a string of the form:

cmdline number sender time message {path}

where:

cmdline is any Multics command line; cmdline must be enclosed in quotes if it contains blanks or other command language characters.

number is the sequence number of the message, assigned when the -hold control argument is used; otherwise, number is 0.

sender is the User_id of the person who sent the message.

time is the date-time the message was sent.

message is the actual message sent.

path is the pathname of the mailbox to which the message was sent. If the message was sent to the default mailbox, path is omitted.

To reverse the effect of a previously specified -call control argument, the user can specify the -call control argument with no cmdline argument.

-flush DT
discards messages sent before the specified date-time, where DT is a string acceptable to the convert_date_to_binary_ subroutine (described in the MPM Subroutines). This control argument is intended to be used by operators and consultants.

-hold, -hd
holds messages until explicitly deleted by the delete_message command. Messages printed when the -hold is in effect are preceded by an identifying number.

-long, -lg
precedes every message printed by the sender's Person_id and Project_id. This is the default.

-nohold
reverts -hold

-prefix STR
places the string STR in front of all messages printed as they are received. The string can be up to 12 characters long and can contain the ioa_ control strings ^/ ^| and ^- if desired.

-print, -pr
prints all messages that were received since the last time the user was accepting messages.

-short, -sh
precedes consecutive messages from the same sender by "=:" instead of the Person_id and Project_id.

-time N, -tm N
prints undeleted messages every N minutes, preceded by a message of the form:

You have X messages

where X is the number of undeleted messages. If N equals 0, time mode is reset.

NOTES: The user should not give conflicting control arguments in the same invocation of the command (i.e., -long and -short or -long and -brief).

If the mailbox:

>udd>Project_id>Person_id>Person_id.mbx

does not exist, the accept_messages command creates it. An event channel is created to receive wakeups from send_message so that when a message is received, it is printed on the user's terminal. Messages sent when the user is not logged in or when the user is deferring messages (see the defer_messages command) are saved in the mailbox and can be read later by invoking the print_messages command. The send_mail command stores mail in the same mailbox. See "Extended Access" in the print_mail command description for an explanation of mailbox access.

Channel and process identifiers are stored in the user's mailbox. Since only one process can receive a wakeup when a message is placed in the mailbox, it is not advisable for several users to share the same mailbox.

SYNTAX AS A COMMAND:

an path names

FUNCTION: adds alternate name(s) to the existing name(s) of a segment, multisegment file, directory, or link.

ARGUMENTS:

path
is the pathname of a segment, multisegment file, directory, or link.

names
are additional names to be added.

ACCESS REQUIRED: modify on the parent directory.

NOTES: The equal and star conventions can be used. See "Constructing and Interpreting Names" in the MPM Reference Guide.

Two entries in a directory cannot have the same entryname; therefore, special action is taken by this command if the added name already exists in the directory that contains the path argument. If the added name is an alternate name of another entry, the name is removed from this entry, added to the entry specified by path, and the user is informed of this action. If the added name is the only name of another entry, the user is asked whether to delete this entry. If the answer is "yes", the entry is deleted and the name is added to the entry specified by path; if the answer is "no", no action is taken.

See also the descriptions of the delete_name and rename commands.

EXAMPLES:

The command line:

```
! an >my_dir>example.pl1 sample.pl1
```

adds the name sample.pl1 to the segment example.pl1 in the directory >my_dir.

The command line:

```
! an >udd>**.private ==.public
```

adds to every entry having a name with private as the last component a similar name with public, rather than private, as the last component.

SYNTAX AS A COMMAND:

```
asp search_list search_path1 {-control_args} ...
     search_pathN {-control_args}
```

FUNCTION: adds one or more search paths to the specified search list.

ARGUMENTS:

search_list
is the name of the search list to which the new search paths are added. Synonyms of search_list are described in the individual command descriptions.

search_pathi
specifies a new search path, where search_pathi is a relative or absolute pathname or a keyword. (For a list of acceptable keywords see "List of Keywords" below.) Each search_pathi can be followed by either the -after, -before, -first, or -last control argument to specify its position within the search list. If no search path position control argument is specified, -last is assumed.

CONTROL ARGUMENTS:
are used only with the search_path arguments and can be chosen from the following:

-after STR, -af STR
specifies that the new search path is positioned after the STR search path. The current search path is an absolute or relative pathname or a keyword. In representing STR it is necessary to use the same name that appears when the print_search_paths command is invoked. This control argument is incompatible with -before and -first.

-before STR, -be STR
specifies that the new search path is positioned before the STR search path. This control argument is incompatible with -after and -first.

-first, -ft
specifies that the new search path is positioned as the first search path in the search list. This control argument is incompatible with -after and -before.

-last, -lt
specifies that the new search path is positioned as the last search path in the search list.

LIST OF KEYWORDS:

Listed below are the keywords accepted as search paths in place of absolute or relative pathnames. There is no restriction as to the position of any of these keywords within the search list.

-home_dir, -hd
-process_dir, -pd
-referencing_dir, -rd
-working_dir, -wd

NOTES: In addition, a pathname can be specified with the Multics active function [user name] or [user project]. A search path enclosed in quotes is not expanded when placed in the search list. It is expanded when referenced in a user's process. This feature allows search paths to be defined that identify the process directory or home directory of any user.

If a link target does not exist, the search facility continues to search for a matching entryname.

The search facility is composed of the following commands:

add_search_paths, asp
delete_search_paths, dsp
print_search_path, psp
set_search_paths, ssp
where_search_paths, wsp

EXAMPLES:

The command line:

```
! asp translator >udd>Project_id>Person_id>include
```

adds the absolute pathname >udd>Project_id>Person_id>include as a search path. This new search path is positioned as the last search path in the translator search list.

add_search_paths (asp)

The command line:

```
! asp trans <include_files -first
```

adds the absolute pathname represented by the relative pathname <include_files as a search path to the trans search list where trans is a synonym for translator. This new search path is positioned as the first search path in the search list.

The command line:

```
! asp info info_files -after >doc>info
```

adds the absolute pathname represented by the relative pathname info_files as a search path to the info search list. This new search path is positioned in the info search list after the >doc>info search path.

The command line:

```
asp translator >udd>[user project]>incl -be >ldd>include
```

adds the unexpanded pathname >udd>[user project]>incl to the translator search list. This new search path is positioned before the >ldd>include search path.

SYNTAX AS A COMMAND:

```
asr path11 {-control_arg path21} ...
          path1n {-control_arg path2n}
```

FUNCTION: allows the user to change object segment search rules dynamically. The search rules to be added can be inserted at any point in the current search rules.

ARGUMENTS:

path1i
is usually a pathname (relative or absolute) representing a directory to be added to the current search rules. It can also be a keyword (see "List of Keywords" below).

path2i
is usually a pathname (relative or absolute) representing a current search rule. It can also be a keyword (see "List of Keywords" below).

CONTROL ARGUMENTS:
must precede the path2i argument.

-before, -be
place path1i before the current search rule identified by path2i.

-after, -af
place path1i after the current search rule identified by path2i.

LIST OF KEYWORDS: In addition to pathnames, both the path1 and path2 arguments accept the keywords:

initiated_segments
referencing_dir
working_dir

LIST OF PATH1 KEYWORDS: The path1 argument also accepts the keywords:

add_search_rules (asr)

home_dir
process_dir
site-defined keywords.

(See the description of the set_search_rules command for an explanation of the site-defined keywords.)

NOTES: If the add_search_rules command is invoked without the control_arg and path2i arguments, the pathname or keyword specified by path1i is appended to the end of the user's current search rules.

Any representation of a current search rule is acceptable for the path2i argument. It is not necessary to use the same name that appears when the print_search_rules command is invoked.

SYNTAX AS A COMMAND:

abc paths {-control_args}

FUNCTION: sets the bit count of a segment that for some reason does not have its bit count set properly (e.g., the program that was writing the segment got a fault before the bit count was set, or the process terminated without the bit count being set).

ARGUMENT:

paths
are the pathnames of segments and multisegment files. The star convention is allowed.

CONTROL ARGUMENTS:

-character, -ch
set the bit count to the last nonzero character. The default is the last nonzero word.

-chase
chases links when using the star convention. The default is to not chase links when using the star convention.

-long, -lg
print a message when the bit count of a segment is changed, giving the old and new values.

-no_chase
does not chase links when using the star convention. This is the default.

ACCESS REQUIRED: The user must have write access on the segment or multisegment file. Modify on the parent directory is not required.

NOTES: The adjust_bit_count command looks for the last nonzero 36-bit word or (if specified) the last nonzero character in the segment and sets the bit count to indicate that the word or character is the last meaningful data in the segment.

If the bit count of a segment can be computed but cannot be set (e.g., the user has improper access to the segment), the

adjust_bit_count (abc)

computed value is printed so that the user can use the set_bit_count command after resetting access or performing other necessary corrective measures. See the description of the set_bit_count command.

The adjust_bit_count command should not be used on segments in structured files. The vfile_adjust command should be used to adjust inconsistencies in structured files.

SYNTAX AS A COMMAND:

```
af strA strB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[af strA strB]
```

FUNCTION: returns the string following the first occurrence of strB in strA. If strB does not occur in strA, the null string is returned.

EXAMPLES:

```
! string [after abcdef123def456 def]
  123def456
! string [after abcdef gh]

! string [format_line XY^aZZ [after 1.4596e+17 7]]
  XYZZ
```

SYNTAX AS A COMMAND:

```
and tf_args
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[and tf_args]
```

FUNCTION: returns true if all the tf_args are equal to true, otherwise it returns false. If any one of the tf_args does not have the value true or false, an error message is printed.

SYNTAX AS A COMMAND:

answer STR {-control_args} command_line

FUNCTION: provides preset answers to questions asked by another command.

ARGUMENTS:

STR
is the desired answer to any question. If the answer is more than one word, it must be enclosed in quotes. If STR is -query, the question is passed on to the user. The -query control argument is the only one that can be used in place of STR.

command_line
is any Multics command line. It can contain any number of separate arguments (i.e., have spaces within it) and need not be enclosed in quotes.

CONTROL ARGUMENTS:

-brief, -bf
suppresses printing (on the user's terminal) of both the question and the answer.

-query
skips the next answer in a sequence, passing on the question to the user. The answer is read read from the user_io I/O switch.

-then STR
supplies the next answer in a sequence.

-times N
gives the previous answer (STR, -then STR, or -query) N times only (where N is an integer).

NOTES: Answer provides preset responses to questions by establishing an on unit for the condition command_question, and then executing the designated command. If the designated command calls the command_query_ subroutine (described in the MPM Subroutines) to ask a question, the on unit is invoked to supply the answer. The on unit is reverted when the answer command returns to command level. See "List of System

Conditions and Default Handlers" in the MPM Reference Guide for a discussion of the command_question condition.

If a question is asked that requires a yes or no answer, and the preset answer is neither "yes" nor "no", the on unit is not invoked.

The last answer specified is issued as many times as necessary, unless followed by the -times N control argument.

EXAMPLES: To delete the test_dir directory without being interrogated by the delete_dir command, type:

```
! answer yes -bf delete_dir test_dir
```

To automatically see the first three blocks of an info segment named fred.info and then be interrogated about seeing any more blocks, type:

```
! answer yes -times 2 help fred
```

The help command prints the first block, then prints another block every time the user answers yes. In this example, the first three blocks are printed before the user is interrogated.

Sequences of answers are especially useful in exec_coms and absentee jobs. To supply the sequence of answers "yes, no, no, yes", type:

```
! answer yes -then no -times 2 -then yes command_line
```

To supply the sequence of answers "no, ask the user twice, yes, no", type:

```
! answer no -query -times 2 -then yes -then no command_line
```

SYNTAX AS A COMMAND:

apl {workspace_id} {-control_args}

FUNCTION: invokes the APL interpreter, optionally loading a saved workspace.

ARGUMENTS:

workspace_id
is the pathname of a saved workspace to be loaded. The default is to load the user's continue workspace, if any, otherwise to provide a clear workspace.

CONTROL ARGUMENTS:

-terminal_type STR, -ttp STR
specifies the kind of terminal being used. Possible values of STR are:

1050
2741
1030
ARDS
ASCII
BITPAIRED
CORR2741
LA36
TEK4013
TEK4015
TELERAY11
TN300
TYPEPAIRED

This control argument specifies which one of several character translation tables is to be used by APL when reading or writing to the terminal. Since there are several different kinds of APL terminals, each incompatible with the rest, it is important that the correct table be used. Specifying a terminal type to APL changes the terminal type only as long as APL is active. The default depends on the user's existing terminal type (refer to the set_tty command, in MPM Communications I/O, CC92). These terminal types default to the same APL terminal type: 1050, 2741, CORR2741, ARDS, TN300, TEK4013, TEK4015, ASCII, LA36, TELERAY11. All other terminal types default to ASCII. The APL terminal types

BITPAIRED and TYPEPAIRED are generic terminal types that can be used with any APL/ASCII terminal of the appropriate type.

-brief_errors, -bfe
causes APL to print short error messages. This is the default.

-long_errors, -lge
causes APL to print long error messages. The short form of the message is printed, followed by a more detailed explanation of the error.

-user_number N
sets the APL user number (returned by some APL functions) to N. The default is 100.

-check, -ck
causes a compatibility error to occur if a monadic transpose of rank greater than 2, or a residue or encode with a negative left argument is encountered. (The definition of these cases is different in Version 2 APL from Version 1 APL).

-debug, -db
causes APL to call the listener (cu_$cl) upon system errors. This puts the user at a new command level. The default is to remain in APL. This control argument is intended for debugging apl itself.

-no_quit_handler, -nqh
causes APL to ignore the quit condition. The default is to trap all quits within APL.

-temp_dir path, -td path
changes the directory that is used to hold the temporary segments that contain the active workspace to path. The default is to use the process directory.

NOTES: This command invokes the Version 2 APL interpreter, which replaces the obsolete Version 1 APL interpreter.

For a complete description of the APL language, terminal conventions, and directions for converting Version 1 APL workspaces, refer to Multics APL, Order No. AK95.

SYNTAX AS A COMMAND:

ac key archive_path paths

FUNCTION: combines an arbitrary number of separate segments into one single segment. The constituent segments that compose the archive are called components of the archive segment.

ARGUMENTS:

key
is one of the functions listed below under "List of Keywords." The key functions are listed according to their operation.

archive_path
is the pathname of the archive segment to be created or used. The archive suffix is added if the user does not supply it. If the archive segment does not exist for replace and append operations, it is created as described above. The star convention can be used with extraction and table of contents operations.

paths
are the components to be operated on by table of contents and delete operations. For append, replace, update and extract operations, each path specifies the pathname of a segment corresponding to a component whose name is the entryname portion of the pathname. The star and equal conventions cannot be used. (Some operations may not require any path arguments; refer to the specific operation for details.)

LIST OF OPERATIONS: The archive command performs a variety of operations that the Multics user can employ to create new archive segments and to maintain existing ones. The operations are:

Table of contents
print a table of contents of an archive segment.

Append
append components to, or create, an archive segment.

Replace
replace components in, append to, or create an archive segment.

Update operation
update an archive segment by replacing components with more recently modified ones.

Delete
delete specified components of an archive segment.

Extract
extract components from an archive segment and place them in segments in the storage system.

Each of these general operations can be specialized to perform several functions and, in many cases, can be combined with the copy and deletion features described below. Such combinations give the user extensive control over the maintenance of his archive segments.

LIST OF KEYWORDS:
is one of the key functions listed below according to their operation.

Table of Contents Operation:

t
print the entire table of contents if no components are named by the path arguments; otherwise print information about the named components only. A title and column headings are printed at the top.

tl
print the table of contents in long form; operates like t, printing more information for each component.

tb
print the table of contents, briefly; operates like t, except that the title and column headings are suppressed.

tlb
print the table of contents in long form, briefly; operates like tl, except that the title and column headings are suppressed.

Append Operation:

a
append named components to the archive segment. (The segments

corresponding to the appended components are not affected.) If a named component is already in the archive, a diagnostic is issued and the component is not replaced. At least one component must be named by the path arguments. If the archive segment does not exist, it is created.

ad
append and delete; operates like a and then deletes all segments that have been appended to the archive. If the safety switch is on for any of the corresponding segments, the user is asked whether to delete the segment.

adf
append and force deletion; operates like a and then forces deletion of all segments that have been appended to the archive.

ca
copy and append; operates like a, appending components to a copy of the new archive segment created in the user's working directory.

cad
copy, append, and delete; operates like ad, appending components to a copy of the archive segment and deleting the appended segments.

cadf
copy, append, and force deletion; operates like adf, appending components to a copy of the archive segment and forcibly deleting the segments requested for appending.

Replace Operation:

r
replace components in, or add components to the archive segment. When no components are named in the command line, all components of the archive for which segments by the same name are found in the user's working directory are replaced. When a component is named, it is either replaced or added. If the archive segment does not exist, it is created.

rd
replace and delete; operates like r, replacing or adding components, then deletes all segments that have been replaced or added.

rdf
replace and force deletion; operates like r and forces deletion of all replaced or added segments.

cr
copy and replace; operates like r, placing an updated copy of the archive segment in the user's working directory instead of changing the original archive segment.

crd
copy, replace and delete; operates like rd, placing an updated copy of the archive segment in the user's working directory.

crdf
copy, replace, and force deletion; operates like rdf, placing an updated copy of the archive segment in the user's working directory.

Update Operation:

u
update; operates like r except that it replaces only those components for which the corresponding segment has a date-time modified later than that associated with the component in the archive. If the component is not found in the archive segment, it is not added.

ud
update and delete; operates like u and deletes all updated segments after the archive has been updated.

udf
update and force deletion; operates like u and forces deletion of all updated segments.

cu
copy and update; operates like u, placing an updated copy of the archive segment in the user's working directory.

cud
copy, update, and delete; operates like ud, placing an updated copy of the archive segment in the user's working directory.

cudf
copy, update, and delete force; operates like udf, placing an updated copy of the archive segment in the user's working directory.

Delete Operation:

d
delete from the archive those components named by the path arguments.

cd
copy and delete; operates like d, placing an updated copy of the archive segment in the working directory.

Extract Operations:

x
extract from the archive those components named by the path arguments, placing them in segments in the storage system. The directory where a segment is placed is the directory portion of the path argument. The access mode stored with the archive component is placed on the segment for the user performing extraction. If a segment already exists, this operation observes the duplicated name convention in a manner similar to the copy command. If no component names are given, all components are extracted and placed in segments in the working directory. The archive segment is not modified.

xf
extract and delete force; operates like x, forcing deletion of any duplicate names or segments found where the new segment is to be created.

NOTES: The process of placing segments in an archive is particularly useful as a means of eliminating wasted space that occurs when individual segments do not occupy complete pages of storage. Archiving is also convenient as a means of packaging sets of related segments; it is used this way when interfacing with the Multics binder (see the bind command description in this document).

The table of contents operation and the extract operation use the existing contents of an archive segment; the other operations change the contents of an archive segment. A new archive segment can be created with either the append or replace operation. In each of the operations that add to or replace components of the archive, the original segment is copied and the copy is written into the archive, leaving the original segment untouched unless deletion is specified as part of the operation. Use of the various operations is illustrated in the "Examples" at the end of this description.

The table of contents operation is used to list the contents of an archive segment. It can be made to print information in long or brief form with or without column headings.

The append operation is used to add components to the archive segment and to create new archive segments. When adding to an existing archive, if a component of the same name as the segment requested for appending is already present in the archive segment, a diagnostic message is printed on the user's terminal and the segment is not appended. When several segments are requested for appending, only those segments whose names do not match existing components are added to the archive segment.

The replace operation is similar to the append operation in that it can add components to the archive segment, and therefore, it is also used to create new archive segments. However, unlike the append operation, if a component of the same name as the segment requested for replacing is already present in the archive segment, that component is overwritten with the contents of the segment. When several segments are requested for replacing, those segments whose names do not match existing components are added to the archive segment, as in the append operation.

The update operation replaces existing components only if the date-time modified of a segment requested for updating is later than that of the corresponding component currently in the archive segment. When a segment whose name does not match an existing component of the archive segment is requested for updating, it is not added to the archive segment.

The delete operation is used only to delete components from archive segments. It cannot delete segments from the storage system and is not analogous to the deletion feature described below.

The extract operation is used to create copies of archive components elsewhere in the storage system. The extract operation performs a function opposite to the append operation.

In addition to the operations described above, there are two features, copying and deletion, that can be combined with

certain operations to modify what they do. Since copying and deletion are features and not operations, they cannot stand alone, but must always be combined with those operations that permit their use. The deletion feature is distinct from the delete operation, as noted below.

The copying feature can be combined with the append, replace, update, and delete operations. Since an archive segment can be located anywhere in the storage system, it is occasionally convenient to move the segment during the maintenance process or to modify the original segment while temporarily retaining an unmodified version. When the copying feature is used, the original archive segment is copied from its location in the storage system, updated, and placed in the user's working directory.

The deletion feature can be combined with the append, replace, and update operations to delete segments from the storage system after they have been added to or replaced in an archive segment. The deletion can be forced to bypass the system's safety function, i.e., the user is not asked whether to delete a protected segment before the deletion is performed. (This is analogous to the operation of the delete_force command.) Nothing is deleted until after the archive segment has been successfully updated.

Deletion of segments (deletion feature) is not to be confused with deletion of components from archive segments. The delete operation is a stand-alone function of the archive command that operates only on components of archive segments, deleting them from the archive. The deletion feature, on the other hand, performs deletions only when combined with an operation of the archive command, and then deletes only segments from the storage system after copies of those segments have been added to, or used to update, archive segments.

The archive command can operate in two ways: if no components are named on the command line, the requested operation is performed on all existing components of the archive segment; if components are named on the command line, the operation is performed only on the named components.

The star convention can be used in the archive segment pathname with extract and table of contents operations; it cannot be used with append, replace, update, and delete

operations. Component names cannot be specified using the star convention. See "Constructing and Interpreting Names" in the MPM Reference Guide for a discussion of the star convention.

No commands other than archive, archive_table, archive_sort, and reorder_archive should be used to manipulate the contents of an archive segment; using a text editor or other command might result in unspecified behavior during subsequent manipulations of that archive segment. See the descriptions of the archive_sort command and the reorder_archive command in the MPM Subsystem Writers' Guide.

Each component of an archive segment retains certain attributes of the segment from which it was copied. These consist of a single name, the effective mode of the user who placed the component in the archive, the date-time the segment was last modified, and the bit count of the segment. In addition, the date-time that the component was placed in the archive segment is maintained. When a component is extracted from an archive segment and placed in the storage system, the new segment is given the name of the component, the bit count of the component, and the mode associated with the component for the user performing the extraction.

The date-time-modified value of a component has a precision of one tenth of a minute. This means that a copy of a component modified less than a tenth of a minute after the archived copy is not updated. Users who update archives in exec_com segments should be aware of this limitation.

The archive command maintains the order of components within an archive segment. When new components are added, they are placed at the end. The archive_sort or reorder_archive commands (described in the MPM Subsystem Writers' Guide) can be used to change the order of components in an archive segment.

The archive command cannot be used recursively. The user is asked a question if the command detects an attempt to use the archive command prior to the completion of its last operation.

Because the replacement and deletion operations are not indivisible, it is possible for them to be stopped before

completion and after the original segment has been truncated. This can happen, for example, if one gets a record quota overflow. When this situation occurs, a message is printed informing the user of what has happened. In this case, the only good copy of the updated archive segment is contained in the process directory.

Archive segments can be placed as components inside other archive segments, preserving their identity as archives, and can later be extracted intact.

When the archive command detects an internal inconsistency, it prints a message and stops the requested operation. For table of contents and extraction operations, it will have already completed requests for those components appearing before the place where the format error is detected.

For segment deletions after replacement requests, if the specified component name is a link to a segment, the segment linked to is deleted. The link is not unlinked.

The archive command observes segment protection by interrogating the user when (unforced) deletion is requested of a segment to which the user does not have write permission. If the user can obtain write permission (i.e., has modify permission on the superior directory) and replies that the segment should be deleted, the segment is deleted.

The archive command refers to the archive segment by full pathname (rather than only the entryname portion) in all printed messages. See "Examples" below.

EXAMPLES: Assume that the user has several short segments and wishes to consolidate them to save space. The working directory, >udd>Project_id>dir_one, might initially look like the following:

```
! list

   Segments = 5, Lengths = 5.

   rw      1  epsilon
   rw      1  delta
   rw      1  gamma
```

```
rw      1  beta
rw      1  alpha
```

The user creates an archive segment (using the append key) containing four of the five segments.

```
!  archive a greek alpha beta gamma delta
   archive: Creating >udd>Project_id>dir_one>greek.archive
```

The working directory then has one more segment (the archive segment), and a table of contents of the new archive segment shows the four components.

```
!  list

   Segments = 6, Lengths = 6.

   rw      1  greek.archive
   rw      1  epsilon
   rw      1  delta
   rw      1  gamma
   rw      1  beta
   rw      1  alpha
```

```
!  archive tl greek

          >udd>Project_id>dir_one>greek.archive

name                    updated     mode      modified    length

alpha          09/12/74 1435.0 rw   09/12/74 1434.2  441
beta           09/12/74 1435.0 rw   09/12/74 1434.2  257
gamma          09/12/74 1435.0 rw   09/12/74 1434.2  694
delta          09/12/74 1435.0 rw   09/12/74 1434.2  109
```

After changing the segment delta, the user replaces it in the archive segment and appends (using the replace key) the segment epsilon to the archive segment. The user also deletes the component gamma.

```
! archive r greek delta epsilon
  archive: epsilon appended to >udd>Project>dir_one>
           greek.archive

! archive d greek gamma
```

A table of contents now shows a different set of components.

```
! archive t greek

          >udd>Project_id>dir_one>greek.archive

 updated               name

09/12/74 1435.0      alpha
09/12/74 1435.0      beta
09/12/74 1437.5      delta
09/12/74 1437.5      epsilon
```

The user later replaces the component alpha with an updated copy and deletes the storage system segment alpha, causing the updated column of a table of contents to change and a list of the working directory to show one less segment.

```
! archive rd greek alpha

! archive t greek

          >udd>Project_id>dir_one>greek.archive

   updated                          name

   09/12/74 1641.5  alpha
   09/12/74 1435.0  beta
   09/12/74 1437.5  delta
   09/12/74 1437.5  epsilon

! list

   Segments = 5, Lengths = 5.
```

```
rw      1  greek.archive
rw      1  epsilon
rw      1  delta
rw      1  gamma
rw      1  beta
```

In another directory, >udd>Project>dir_two, which contains a more recent version of the segment alpha, the user copies and updates the archive segment, causing the component alpha to be replaced and the updated archive segment to be placed in the working directory.

```
! archive cu <dir_one>greek
  archive: Copying >udd>Project_id>dir_one>greek.archive
  archive: alpha updated in >udd>Project_id>dir_two>
             greek.archive

! list

  Segments = 2, Lengths = 2.

  rw      1  greek.archive
  rw      1  alpha

! archive t greek

        >udd>Project_id>dir_two>greek.archive

   updated                        name

  09/12/74 1648.3  alpha
  09/12/74 1435.0  beta
  09/12/74 1437.5  delta
  09/12/74 1437.5  epsilon

! ac t <dir_one>greek

        >udd>Project_id>dir_one>greek.archive

   updated             name

  09/12/74 1641.5  alpha
  09/12/74 1435.0  beta
```

```
09/12/74 1437.5  delta
09/12/74 1437.5  epsilon
```

Notice that the entry in the updated column for the component alpha differs in the two tables of contents. Finally, the user extracts two components into the new working directory, presumably to work on them.

```
! archive x greek beta delta

! list

   Segments = 4, Lengths = 4.

   rw      1  delta
   rw      1  beta
   rw      1  greek.archive
   rw      1  alpha
```

SYNTAX AS A COMMAND:

act archive_path {starnames}

SYNTAX AS AN ACTIVE FUNCTION:

[act archive_path {starnames}]

FUNCTION: returns the names of specified archive components in a specified archive segment. Names are returned separated by single spaces.

ARGUMENTS:

archive_path
is the pathname of an archive segment, with or without the archive suffix. The star convention is NOT allowed.

LIST OF OPTIONAL ARGUMENTS:

starnames
are optional component names to be matched against names of archive components. The star convention is allowed.

NOTES: Invoked as a command, archive_table prints the component names, one name per line.

SYNTAX AS A COMMAND:

ar resource_type {-control_args}

FUNCTION: calls the resource control package (RCP) to assign a resource to the user's process.

ARGUMENT:

resource_type
specifies the type of resource to be assigned. Currently, only device types can be specified. The -device control argument is used to name a specific device to assign. Other control arguments are used to specify characteristics of the device to be assigned. The following device type keywords are supported:
tape_drive
disk_drive
console
printer
punch
reader
special

CONTROL ARGUMENTS:

-device STR, -dv STR
specifies the name of the device to be assigned. If this control argument is specified, other control arguments that specify device characteristics are ignored. (See "Examples" below.) If the -long control argument (see below) is used in conjunction with this control argument, a message containing the name of the assigned device is printed on the user's terminal; otherwise, no message is printed.

-model N
specifies the device model number characteristic. Only a device that has this model number is assigned. In order to find the model numbers that are acceptable, use the print_configuration_deck command described in System Tools, Order No. AZ03.

-track N, -tk N
specifies the track characteristic of a tape drive. The value can be either 9 or 7. If this control argument is not specified and if the -volume control argument is not specified, a track value of 9 is used when assigning a tape device.

-density N, -den N
specifies the density capability characteristic of a tape drive. There can be more than one instance of this argument. A tape drive is assigned that is capable of being set to all of the specified densities. The acceptable values for this argument are:

200
556
800
1600
6250

Note that the values permitted depend on the particular hardware on the system.

-train N, -tn N
specifies the print train characteristic of a printer.

-line_length N, -ll N
specifies the line length of a printer. Its value must be one that is found in the "line length" field of a printer PRPH configuration card. If this field is not specified on a printer PRPH configuration card, this device characteristic is ignored for this printer.

-volume STR, -vol STR
specifies the name of a volume. If possible, the device assigned is one on which this volume has already been placed. If this is not possible (e.g., the volume is on a device assigned to a process) any available, appropriate, and accessible device will be assigned.

-number N, -nb N
specifies the number of resources to assign. All of the resources assigned have the device characteristics specified by any other arguments passed to this command. If this control argument is not specified, one resource is assigned.

-comment STR, -com STR
is a comment string that is displayed to the operator when the resource is assigned. If more than one string is required, the entire string must be in quotes. Only printable ASCII characters are allowed. Any unprintable characters (also tabs or new lines) found in this string are converted to blanks.

-long, -lg
specifies that all of the device characteristics of the assigned device should be printed. If this argument is not supplied, only the name of the assigned device is printed.

-system, -sys
specifies that the user wants to be treated as a system process during this assignment. If this argument is not specified or if the user does not have the appropriate access, then the RCP assumes that this assignment is for a nonsystem process.

-wait {N}, -wt {N}
specifies that the user wants to wait if the assignment cannot be made at this time because the resources are assigned to some other process. The value N specifies the maximum number of minutes to wait. If N minutes elapse and a resource is not yet assigned, an error message is printed. If N is not specified, it is assumed that the user wants to wait indefinitely.

-speed N
specifies the speed of a tape drive. The acceptable values depend on the particular hardware on the system and can be the following:

75
125
200

NOTES: Currently, only device resources can be assigned. An assigned device still must be attached by a call to some I/O module. If a device is successfully assigned, the name of the device is printed. (If the user requests a specific device that is successfully assigned, the name of the device is not printed unless the user asks for it. See the -device and -long control arguments above.)

EXAMPLES: In the example below, the user issues the assign_resource command with the "tape_drive" keyword and the -model control argument. The system responds with the name of the assigned device.

```
! assign_resource tape_drive -model 500

   Device tape_04 assigned
```

In the next example, the user issues the assign_resource command with the "tape_drive" keyword and the -device and -long control arguments. The system responds with the name of the assigned device and the model number, track, density and speed characteristics.

```
! assign_resource tape_drive -device tape_05 -long

   Device tape_05 assigned
   Model     =  500
   Tracks    =  9
   Densities =  200 556 800 1600
   Speed     =  125
```

SYNTAX AS A COMMAND:

ata {old_switch {new_switch}} {-control_args}

FUNCTION: sets up a specified I/O switch, with a stream_input_output opening, to be audited by the audit_ I/O module.

ARGUMENTS:

old_switch
is the name of an I/O switch to be audited. The default is user_i/o. If only one switch is specified, it is the old_switch.

new_switch
is the name of an I/O switch to be used by the audit_ I/O module. If only one switch argument is given, it is the old_switch. The default value for new_switch is audit_i/o.<time>, where <time> has the value MM/DD/YY--hhmm.m.

CONTROL ARGUMENTS:

-truncate, -tc
truncates the audit file if it already exists. If this control argument is not given, the audit file is extended by default.

-pathname path, -pn path
specifies that path is the pathname of the audit file to use. If pathname is not given, the audit file is in the user's home directory and named date.audit.

NOTES: If used with no arguments, attach_audit sets up auditing for the user_i/o I/O switch with input and output audited and editing on.

Auditing of old_switch is done by moving the attachment of old_switch to new_switch and then attaching old_switch to new_switch via audit_. See the MPM Subroutines discussion of the audit_ I/O module and the MPM Commands discussion of detach_audit for more information.

LIST OF AUDITING REQUESTS:
A three-character sequence is used to make an auditing

request: the audit trigger character ("!" by default), followed by the specific request character, followed by a newline. An auditing request can either be alone on a line or have text preceding it on the same line.

!.
prints the combination of input and/or output being audited.

!?
prints a brief description of available auditing requests.

!e
enters the audit editor. The entry preceding this sequence becomes the current line to be edited.

!E
enters the audit editor, and processes any text preceding the sequence on the same line as editing requests. If no text precedes the sequence, the effect is the same as for !e.

!a
expands abbreviations in the input line. (See the abbrev command in this manual for more information.)

!r
redisplays the input line and strips off the newline. Further input can then be appended to the redisplayed line until another newline is typed, but no further erase or kill processing is performed on the redisplayed portion. The redisplayed line plus the appended input (if any) becomes the input line that is returned to the I/O module being audited.

!t
instructs the audit_ I/O module not to log the input line; this makes the input line transparent.

!d
specifies that the input line to which this is appended is deleted. This is used to kill a line that has been redisplayed with the !r request.

!n
specifies no operation; this is useful when the !n follows another auditing request sequence that the user does not want interpreted.

NOTES ON AUDIT FILE:

The audit file, by default, has the pathname:

>udd>Project_id>Person_id>date.audit

where date is the first eight characters (the date portion) returned by the date_time_ subroutine at the time of attaching, and is of the form "MM/DD/YY". This pathname can also be specified using active functions:

[home_dir]>[date].audit

The default audit file size is unlimited, and the audit file can become a multisegment file.

The audit editor operates on entries, rather than lines, and the entry type identifiers are:

EL edit line

IC input characters

IL input line

OC output characters

TC trace of control operations

TM trace of modes operations

NOTES ON AUDIT EDITOR: The audit editor is invoked by typing the e or E auditing request sequence described above. It edits and executes lines that have been logged by the audit_ I/O module. The syntax of editing requests is similar to that of qedx requests (see the qedx command in this manual). Any number of requests can be on the same line; spaces are ignored.

Addressing is done the same way as in the qedx editor, with two exceptions. The "." is a request for self-identification rather than an indicator for the current entry, and addresses are expressed in terms of entries in the audit file rather than lines in a buffer. The edit buffer contains only one entry at a time. If the default search tag is in use, as is the case unless specifically overridden, the absolute entry number refers to the number of entries, with the default search tag, from the beginning of the file. Similarly, a relative entry address refers to the number of entries, with the default search tag, before or after the current address.

LIST OF EDITING REQUESTS: The audit editor requests are presented below in two categories: familiar (qedx-like) requests, and special requests.

s/REGEXP/STR/
substitutes the string STR for occurrences of the regular expression REGEXP in the edit buffer.

ADR
locates the entry with address ADR. If ADR is not followed by a request, the audit file entry is printed. An ADR can contain an absolute entry reference at its beginning, relative addresses in any portion, and regular expressions in any portion. If a regular expression in the address is preceded by the less than character (<), a backward search is done to find a match for the regular expression. An absolute address is either a number, or the dollar sign ($) to indicate the last entry in the audit file.

{ADR1,ADR2}p
prints the current entry if no ADR is specified; prints the addressed audit file entry if a single address is specified; prints entries from address 1 through address 2 if two addresses are specified.

..STR
passes the string STR to the command processor and then returns to the audit editor.

q
quits the editor and returns the current line to the I/O module being audited, with the !e or !E sequence included.

.expand
expands abbreviations in the edit buffer (see the abbrev command in the MPM Commands for a discussion of expansion of abbreviations).

.off
disables auditing of input and output in the editor.

.on
enables auditing of input and output in the editor.

.l
addresses the last audit file entry returned by the audit editor.

.r[STR]
quits the editor and returns the string STR to the I/O module

being audited. If STR is not specified, the r request quits the editor and returns the edit buffer.

.n
returns a newline character.

.type
prints the audit file entry type of the current position.

.exec
passes the edit buffer to the command processor and returns to the audit editor.

.d/STR/
sets the default search tag to the string STR. If STR is only one character, only the first character of the tag is used to determine if an entry is seen (in counting entries and doing searches). If STR is two characters, the match is made on both characters of the tag.

.?
prints a brief description of available audit editor requests.

:
overrides the default search tag for those requests following on the same line (i.e., any tag is matched). A newline reestablishes the default search tag.

The REGEXP field of a substitute request is interpreted as a qedx-style regular expression. The STR field of a substitute request is also interpreted as in qedx, and the & convention is supported. If REGEXP is null in a substitute request, the last REGEXP specified in a previous substitute request is used.

No lines in the audit file are changed by the editor; only copies are modified.

If the audit editor is being audited, the audit editor can be invoked from within the editor. For every level of the editor, a distinct last returned line is remembered.

EXAMPLES: In the example given below, there has been such extensive use of the erase character that the user may want to see it displayed. In order to verify the input line given, it

can be replayed by using the !r request. The ! at the beginning of the line indicates lines typed by the user.

```
!  str#ty =#-print_mod########modes red!r
   stty -pmodes red
```

This line does not end with a newline character, so the next character typed would appear immediately following the "red" and on the same line. In this example, -pmodes was entered instead of -modes. Typing the following on that same line:

```
!                  ##########modes red!r
```

does not correct the error, but returns:

```
   stty -pmodes redmodes red
```

The erase character cannot be used to correct portions of a line that has already been replayed. The current situation can be corrected as follows:

```
!  stty -pmodes redmodes red!e
!  p
   stty -pmodes redmodes red
!  s/redmodes red/red/ s/pmodes/modes/p
   stty -modes red
!  .r
```

The above procedure enters the audit editor with the !e request. The p request prints the contents of the edit buffer. If no argument is given for p, the most recent input line is printed. Corrections are made to the line and the modified line is printed. The request .r exits the audit editor and returns the line to the I/O module being audited.

An alternative procedure is the following:

```
!  stty -pmodes redmodes red!n
!  stty -modes red
```

The request !n suppresses the entire input line and it is then reentered correctly.

In the first example given, there are two ways to set the red shift mode. It can be turned off and then on again, as follows:

```
!  stty -modes ^red
!  .l.r!E
```

The .l.r enters the audit editor. This puts the last entry returned by the audit editor in the edit buffer, then returns the contents of the buffer. To request the "stty -modes ^red" command, type:

```
!  </^stty/p.r!E
   stty -modes ^red
```

This does a backward search in the audit file for an input entry beginning with stty, puts this entry in the edit buffer, prints the contents of the edit buffer, and returns the contents of the edit buffer.

To see the last five input entries in the audit file at this point, type:

```
!  -4,p!E
   s/redmodes red/red/ s/pmodes/modes/p
   .r
   stty -modes ^red
   .l.r!E
   </^stty/p.r!E
```

To see the last five output entries prior to this invocation of the audit editor, type:

```
!  .d/0/
!  -4,p
   stty -pmodes red stty -pmodes redmodes red
   stty -pmodes redmodes red
   stty -modes red
   stty -modes ^red
```

Note that the entries that are the result of a replay (!r) do not end in a newline character, so they run together on the same line when being printed.

attach_lv (alv)

SYNTAX AS A COMMAND:

alv volume_name

FUNCTION: calls the resource control package (RCP) to attach a logical volume.

ARGUMENTS:

volume_name
specifies the name of the volume to be attached.

ACCESS REQUIRED: A user must have rw access to the logical volume to be attached, as defined by the access control segment (ACS) associated with the logical volume.

NOTES: Attaching a logical volume involves informing the storage system that a particular volume is attached for a particular process. A logical volume (unless it is a public logical volume) must be attached for each process that wishes to use it. To be attached, the logical volume must first be physically mounted. This mounting involves mounting all of the physical volumes that compose the logical volume.

If the specified volume is not already mounted, the system operators are requested to mount the volume, if appropriate resources are available. The attach_lv command does not return until the volume is mounted or the operator has denied the request.

The status command issued with the -device control argument prints the name of the logical volume on which a segment resides.

SYNTAX AS A COMMAND:

basic path {-control_arg}

FUNCTION: invokes the BASIC compiler to translate a segment containing BASIC source code. Either the compiled code is executed, or a standard object segment is created to be executed at a later time.

ARGUMENTS:

path
is the pathname of the segment to be translated. The basic suffix need not appear as part of the pathname. It must, however, be the last component of the name of the source segment.

CONTROL ARGUMENTS:

-compile
requests BASIC to compile the program and generate a bindable Multics standard object segment. The resulting object segment is placed in the user's working directory.

-time N
where N is a decimal number that requests a limit of N seconds on the execution of the BASIC program. If the limit is exceeded, the user is asked whether to continue.

NOTES: The -compile and -time control arguments are incompatible.

If the -compile control argument is not specified, the compiled code is then executed and not saved for future execution. If the -compile control argument is specified, a standard object segment is created for subsequent execution.

For a description of the BASIC language on the Multics system, consult the Multics BASIC manual, Order No. AM82.

For information on using the FAST subsystem to compile BASIC source code, refer to the Multics FAST Subsystem Users' Guide, Order No. AU25.

For a description of the features common to all Multics programming languages, see "Programming Languages" in the MPM Reference Guide. A description of object segments can be found in the MPM Reference Guide in "Creating an Object Segment."

SYNTAX AS A COMMAND:

```
be strA strB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[be strA strB]
```

FUNCTION: returns the string preceding the first occurrence of strB in strA. If strB does not occur in strA, the entire string strA is returned.

EXAMPLES:

```
! string [before abcdef123defabc def]
  abc
! string [before abcdef g]
  abcdef
! string [before abcdef123 abc]

! string [format_line XY^aZZ [before 1.4596e+17 7]]
  XY1.4596e+1ZZ
```

SYNTAX AS A COMMAND:

```
bin values
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[bin values]
```

FUNCTION: returns one or more values in binary.

ARGUMENTS:

value
is a value to be processed. The last character of value indicates its type. Acceptable types are binary (b), quartenary (q), octal (o), hexadecimal (h), and unspecified (u). Any valid PL/I real value is allowed. The absence of any specifier means decimal. The unspecified value is limited to 8 characters.

EXAMPLES:

```
! string [binary 657.4o]
110100111.1
```

SYNTAX AS A COMMAND:

bind paths {-control_args}

FUNCTION: produces a single bound object segment from one or more unbound object segments, stored in archive segments, which are called the components of the bound segment.

ARGUMENTS:

paths
are the pathnames of archive segments containing one or more component object segments to be bound. The archive suffix is assumed. Up to 16 input archive segments can be specified. They are logically concatenated in a left-to-right order to produce a single sequence of input component object segments.

CONTROL ARGUMENTS:

-update paths, -ud paths
indicates that the following list of archive segments (paths) specifies update rather than input object segments. The archive suffix is assumed in paths. Up to a combined total of 16 input and update segments can be specified. The contained update object segments are matched against the input object segments by object segment name. Matching update object segments replace the corresponding input object segments; unmatched ones are appended to the sequence of input object segments. If several update object segments have the same name, only the last one encountered is bound into the bound segment.

-list, -ls
produces a listing segment whose name is derived from the name of the bound object segment plus a suffix of list. The listing segment is generated for the purpose of dprinting; it contains the bound segment's bind control segment (see "Notes on Bindfile" below), its bind map, and that information from the bound object segment printed by the print_link_info command. This control argument cannot be invoked with -map. In the absence of the -list or -map control arguments, no listing segment is generated.

-map
produces a listing segment (with the suffixes list and map) that contains only the bind map information. This control argument is incompatible with -list. In the absence of the

-list or -map control arguments, no listing segment is generated.

-brief, -bf
suppresses printing of warning messages.

NOTES: Compilers and the assembler produce unbound object segments. Binding has three benefits: the reduction of storage fragmentation, the prelinking of external references between the components, and the reduction of size of address space necessary to execute the components.

Each of these benefits saves CPU time and storage usage if the set of components bound is used with regularity. This reduction in usage translates directly into lower charges for the users of the bound segment. System efficiency is also increased by binding together common sets of programs. A reference in one component to an entrypoint defined in another component is resolved during the binding. This prelinking avoids the cost of dynamic linking, and it also ensures that the reference is linked to the component regardless of the state of a process at the moment that dynamic linking takes place. References to an entrypoint are prelinked unless the contrary is specified by an instruction in the bindfile. The bindfile is a segment containing instructions that control various aspects of the binding operation (see "Notes on Bindfile" below). (See the description of the print_link_info command in the MPM Subsystem Writers' Guide.)

NOTES ON OUTPUT: The binder produces as its output two segments: an executable bound object segment and an optional, printable ASCII listing segment. The name of the bound segment is, by default, derived from the entryname of the first input archive segment encountered by stripping the archive suffix from it. The name of the listing segment is derived from the name of the bound segment by adding the list suffix to it. Use of the Objectname master statement in the bindfile (see "List of Master Keywords" below) allows the name of the bound segment to be stated explicitly. In addition, use of the Addname master statement in the binding instructions causes additional segment names to be added to the bound segment. The primary name of the bound segment must not be the same as the name of any component.

NOTES ON BINDFILE: The bindfile is a segment containing symbolic instructions that control the operation of the binder. Its

entryname must contain the bind suffix and it must be archived into any one of the input archive segments (at any location within that archive segment) where it is automatically located and recognized by the binder.

In case two bindfiles are specified, one in an input archive segment and the other in an update archive segment, the latter takes precedence and a warning message is printed to that effect.

The syntax of the bindfile statements consist of a keyword followed by zero or more parameters and then delimited by a statement delimiter. Master statements pertain to the entire bound object segment; normal statements pertain to a single component object within the bound segment. Master statements are identified by master keywords that begin with a capital letter; normal keywords begin with a lowercase letter. A keyword designates a certain action to be undertaken by the binder pertaining to parameters following the keyword.

LIST OF BINDFILE DELIMITERS:

: keyword delimiter
It is used to identify a keyword followed by one or more parameters. A keyword that is followed by no parameters is delimited by a statement delimiter.

; statement delimiter.

, parameter delimiter
(Note, the last parameter is delimited by a statement delimiter).

/* begin comment

*/ end comment

LIST OF MASTER KEYWORDS:

Objectname
the parameter is the segment name of the new bound object.

Order
the parameters are a list of objectnames in the desired binding order. In the absence of an order statement, binding is done in the order of the input sequence. The order

statement requires that there be a one-to-one correspondence between its list of parameters and the components of the input sequence.

Force_Order
same as Order, except that the list of parameters can be a subset of the input sequence, allowing the archive segments to contain additional segments that are not to be bound (e.g., source programs).

Global
the parameters can be either retain, delete, or no_link. The parameter selected pertains to all component object segments within the bound segment. A global or explicit statement concerning a single component object or a single external symbol of a component object overrides the Global statement for that component object or symbol.

Addname
the parameters are the symbolic names to be added to the bound segment. If Addname has no parameters, it causes the segment names and synonyms of those component objects for which at least a single entrypoint was retained to be added to the bound segment.

No_Table
does not require parameters. It causes the symbol tables from all the component symbol sections containing symbol tables to be omitted from the bound segment. If this keyword is not given, all symbol tables are kept.

Perprocess_Static
does not require parameters. It causes the perprocess_static flag of the bound segment to be turned on, which prevents the internal static storage from being reset during a run unit.

If no bindfile is specified, the binder assumes default parameters corresponding to the following:

Objectname: segment name of the first input archive file.

Global: retain; /*regenerate all definitions*/

LIST OF NORMAL KEYWORDS:

objectname
the single parameter is the name of a component object as it appears in the archive segment. The objectname statement

indicates that all following normal statements (up to but not including the next objectname statement) pertain to the component object whose name is the parameter of the objectname statement.

synonym
the parameters are symbolic segment names declared to be synonymous to the component object's objectname. When b is declared to be a synonym for a, references (in the bound components) of the form b or b$x (any x) are resolved during binding by searching for a definition of b or x in component a. A synonym instruction must be given if such references are to be prelinked. The synonym instruction also affects dynamic linking so that if b is a reference name for the bound segment, then references of the form b or b$x are resolved by searching component a. In this case, the synonym instruction may reduce the cost of dynamic linking, and it avoids possible ambiguities when two components contain definitions for the symbol b. Users should take care to state explicitly in a synonym statement all the normally used segment names of a component object. For example, the create and create_dir commands are implemented in one procedure, and both have abbreviations; thus a bindfile for the bound segment in which this procedure resides contains:

```
objectname:    create;

synonym:       create, cr, create_dir, cd;
```

Failure to state segment names results in inefficient linker performance.

retain
the parameters are the names of entrypoints defined within the component object segment that the user wishes to retain as entrypoints of the bound object segment.

delete
the parameters are the names of entrypoints defined within the component object segment that the user does not wish to be retained as entrypoints of the new bound segment.

The retain and delete statements are considered exclusive. An error message is displayed if the binder recognizes that two or more such statements were made regarding any single entrypoint.

no_link
the parameters are the names of entrypoints that are <u>not</u> to be

prelinked during binding. The no_link statement implies a retain statement for the specified names.

global
the parameter can be either retain, delete, or no_link. The parameter selected becomes effective for all entrypoints of the component object. An explicit retain, delete, or no_link statement concerning a given entrypoint of the component object overrides the global statement for that specific entrypoint. A global no_link causes all external references to the component object to be regenerated as links to entrypoints; this allows execution-time substitution of such a component by a free standing version of it, for example for debugging purposes.

table
does not require parameters. It causes the symbol table for the component to be retained and is needed to override the No_Table master keyword, described above.

NOTES ON ERROR MESSAGES: The binder produces three types of error messages. Messages beginning with the word "Warning" do not necessarily represent errors, but warn the user of possible inconsistencies in the input components or bindfile. Messages beginning with the word "binder_" normally represent errors in the input components. Errors detected during the parsing of the bindfile have the format:

Bindfile Error Line #N ....

where N is the line number of the erroneous statement. If an error is detected during parsing, the binder aborts because it cannot bind according to the user's specifications.

The message:

"binder_: Fatal error has occurred; binding unsuccessful."

indicates that it was impossible for the binder to produce an executable object segment because of errors detected during binding. The bound object segment is left in an unpredictable state.

EXAMPLES: The bindfile for the debug command, which is named bound_debug.bind, is as follows:

```
Objectname:  bound_debug;
Global:      delete;     /*delete all old definitions*/

Addname;                 /*add names debug, db, list_arg_
                           and gr_print to bound segment
                           bound_debug_*/

objectname:  debug;
synonym:     db;         /*indicate db is synonymous to debug*/
retain:      debug,
             db;         /*retain entrynames debug$debug and
                           debug$db*/

objectname:  list_arg_;
retain:      list_arg_; /*retain entryname list_arg_$list_arg_*/
objectname:  gr_print;
retain:      gr_print;  /*retain entryname gr_print$gr_print*/
```

The following illustrates other uses of the bindfile:

```
Objectname:  bound_test;
Global:      delete;     /*delete all old definitions*/
Order:       test,       /*list all components in the
                           order they are to be bound*/
             test_utility,
             test_init,
             reset;

Addname:     test,
             test_utility, /*add so that link can be
                             snapped to version in
                             bound segment*/
             reset;

No_Table;                /*omit all symbol tables*/

objectname:  reset;
retain:      reset;

objectname:  test;
retain:      test;

objectname:  test_utility;
synonym:     rest_of_test; /*another entrypoint*/
no_link:     test_utility; /*do not prelink to this
                             entrypoint; generate
                             external link*/
             table;        /*keep this component's
                             symbol table*/
```

SYNTAX AS A COMMAND:

```
bool B1 B2 B3
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[bool B1 B2 B3]
```

FUNCTION: performs bit string operations on character string representations of bit strings.

ARGUMENTS:

B1, B2, and B3
are bit strings entered as 0 and 1 characters. B3 must be 4 bits long. It causes the following logical operations to be performed on B1 and B2.

B3	Name	Result
0000	clear	all zeroes
0001	and	B1 & B2
0010		B1 &^B2
0011	move B1	B1
0100		^B1 & B2
0101	move B2	B2
0110	xor	(B1&^B2) | (^B1&B2)
0111	or	B1 | B2
1000	^or	^(B1|B2) = (^B1&^B2)
1001	^xor	^((B1&^B2) | (^B1&B2)) = (^B1|B2) & (B1|^B2)
1010	invert B2	^B2
1011		^(^B1&B2) = (B1|^B2)
1100	invert B1	^B1
1101		^(B1&^B2) = (^B1|B2)
1110	^and	^(B1&B2) = (^B1|^B2)
1111	^clear	all ones

NOTES: The shorter of the two strings is extended at the right with zeroes to equal the length of the longer string.

EXAMPLES:

```
! string [bool 1010 0101 0111]
  1111
! string [bool 1001001 1101001010 0110]
  0100000010
```

SYNTAX AS A COMMAND:

```
branches star_names {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[branches star_names {-control_arg}]
```

FUNCTION: returns the entrynames or absolute pathnames of segments, directories, and multisegment files that match one or more star names.

ARGUMENTS:

star_name
is a star name to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames. The default is to return entrynames.

NOTES: Only one name per branch is returned; i.e., if a branch has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by branches is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

A synonym for branches is nonlinks.

EXAMPLES:

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w    0  empty_seg
  re     1  test
  r w    1  test.list
  r w    1  test.pl1
  re     1  prog
  r w    1  prog.list
  r w    1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w  513  prog.output
  r w  257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                  >udd>Apple>Jones>temp_seg_2
  prog.temp1                  >udd>Apple>Jones>temp_seg_1
  junk                        >udd>Apple>Jones>empty_seg

! string [branches prog*.**]
  prog.pl1 prog.list prog.data prog_stuff prog prog.output
! string [branches prog.*]
  prog.pl1 prog.list prog.data prog.output
```

SYNTAX AS A COMMAND:

calc {expression}

SYNTAX AS AN ACTIVE FUNCTION:

[calc expression]

FUNCTION: provides the user with a calculator capable of evaluating arithmetic expressions with operator precedence, a set of often-used functions, and a memory that is symbolically addressable (i.e., by identifier).

ARGUMENTS:

expression
is an arithmetic expression (see below) to be evaluated. If this argument is specified, the calc command prints its value and returns to command level. The expression arguments must be quoted if it contains spaces or other command language characters. Variables are not allowed.

NOTES: Invocation of calc with a newline enters calculator mode. The user can then type in expressions, assignment statements, or list requests, separated from each other by one or more newline characters. All of these operations are described below.

The user must use the quit request with a newline character to return to command level.

NOTES ON EXPRESSIONS: Arithmetic expressions involving real values and the operands +, -, *, /, and ** (addition, subtraction, multiplication, division, and exponentiation) can be typed in. A prefix of either plus or minus is allowed. Parentheses can be used, and blanks between operators and values are ignored. Calc evaluates each expression according to rules of precedence and prints out the result. The quit request (followed by a newline character) returns the user to command level. The order of evaluation is as follows--

expressions within parentheses

function references

prefix +, prefix -

**

*, /

+, -

For example, if the user types--

```
2 + 3 * 4
```

calc responds--

```
=   14
```

Operations of the same level are processed from left to right except for the prefix plus and minus, which are processed from right to left. This means 2**3**4 is evaluated as (2**3)**4.

Numbers can be integers (123), fixed point (1.23) and floating point (1.23e+2, 1.23e2, 1.23E2, or 1230E-1). All are stored as float bin(27). An accuracy of about seven figures is maintained. Variables (see below) can be used in place of constants, e.g., pi * r ** 2.

Seven functions are provided: sin, cos, tan, atan, abs, ln, and log (ln is base e, log is base 10). They can be nested to any level, e.g., sin(ln(var).5*pi/180).

NOTES ON ASSIGNMENT STATEMENTS: The value of an expression can be assigned to a variable. The name of the variable must be from one to eight characters in length and must be made up of letters (uppercase and/or lowercase) and the underscore character (_). The form is--

<variable>=<expression>

For example, the following are legal assignment statements--

```
x = 35

Rho = sin(2*theta)
```

The calc command does not print any response to assignment statements. The variables "pi" and "e" have preassigned values of 3.14159265 and 2.7182818, respectively.

NOTES ON THE LIST REQUEST: If "list" is typed, calc prints out the names and values of all the variables that have been declared so far. The value of any individual variable can be displayed by typing the name of the variable followed by a newline character.

EXAMPLES: The lines typed by the user are preceded by an exclamation mark (!).

```
!  calc
!  2+2
   =   4
!  r = 1.5
!  pi*r**2
   =   7.068583
!  sin(0.01)
   =   9.999832E-3
!  143e11+(12e13
   too few )
!  143e11+(12e13)
   =   1.343E+14
!  list

   r      =   1.5
   e      =   2.718282
   pi     =   3.141592
!  q
```

SYNTAX AS A COMMAND:

calendar {DT} {paths} {-control_arg}

FUNCTION: prints a calendar page for one month.

ARGUMENTS:

DT
identifies which month is printed. This argument must be a date acceptable to the convert_date_to_binary_ subroutine (described in the MPM Subroutines). If the DT argument is not given, the current month is printed.

pathi
is the pathname of a segment that contains a list of events in the form of text to be inserted into the calendar. For information on segment format, see "Notes" below.

CONTROL ARGUMENTS:

-fw
labels boxes with fiscal week numbers. The calendar command assumes that each fiscal week begins on Monday and ends on Sunday and fiscal week 1 is the first full week of the calendar year. Fiscal week 1 of 1980 therefore begins on Monday, January 7, 1980.

NOTES: Each box for a calendar day is 16 characters wide and 7 high. Each box in the calendar contains the number of the day of the month; other information can also appear in the box, at the user's option. The month preceding the specified month and the month following it are also printed.

Each segment contains lines that set up a string to be inserted into the appropriate box of the calendar. The fields in these lines are separated by commas and have the form--

date,07/04,Independence Day

The first field is the operation code (either date, rel, easter or rename). The second and succeeding fields depend on which operation code is used. Lines that produce a date not in the current month are ignored. Lines beginning with an asterisk (*) are comment lines. Leading space is NOT allowed.

There are three fields for the date operation code with the first field containing the operation code, date. The second field is any date acceptable to the convert_date_to_binary_ subroutine. (This date is converted relative to the day before the beginning of the month, so that "Mon" is the first Monday in the month, etc.) The third field is arbitrary text. Up to 16 characters are inserted into the calendar in the appropriate place, if the date specified in the line (see example above) falls in the calendar month.

There are five fields for the rel operation code. The first contains the operation code, rel, itself; the second is the one or two digit month number, or 0, -1, or +1. A month of 0 is the current month, -1 is the month preceding the current month, and +1 is the month following the current month. The third is a date, relative to the day before the first of the previous month. The fourth field is a date relative to the third field and specifies the day selected. The fifth field is text. For more complete information on how these can be specified, see the convert_date_to_binary_$relative subroutine entrypoint in the MPM Subroutines. Thus, the line--

```
rel,11,Mon,Tue,Election Day
```

defines the first Tuesday after the first Monday in November, and places the text, "Election Day," in the proper calendar day box.

There are only two fields for the easter operation code. The second is the text (e.g., "Easter") that is inserted into the box for Easter.

```
easter,Easter
```

The fourth operation code is rename which has three fields; the first being the name of the operation code, the second is an existing day or month name, and the third field is the character string to replace it.

```
rename, Monday, Lundi
```

Users can insert up to six lines of text for any date. This is accomplished by supplying multiple date or rel entries for the desired date (see Washington's birthday under "Examples" below).

EXAMPLES: The following illustrates the kind of segment a user might create to put fixed holidays into a calendar.

```
* holidays
*
date,01/01,New Year's Day
date,02/02,Ground Hog Day
rel,2,Mon,2 weeks,Washington's
rel,2,Mon,2 weeks, birthday
easter,Easter
rel,4,Mon,2 weeks,Patriot's Day
rel,5,Sun,1 week,Mother's Day
rel,5,05/24,Mon,Memorial Day
date,07/04,Independence Day
rel,9,0,Mon,Labor Day
rel,10,Mon,1 week,Columbus Day
date,11/11,Veterans Day
rel,11,Mon,Tue,Election Day
rel,11,Thu,3 weeks,Thanksgiving
date,12/25,Christmas Day
```

Additionally, a user might create a segment to include personal information in a calendar.

```
* personal calendar info
*
date,12/10,Mike's Birthday
date,03/07,Dad's Birthday
  .
  .
  .
  .
```

Assume that the user wants a calendar for the coming December, including fiscal week numbers, holidays, and personal information. If the above segments are named "holidays" and "personal" (and are in the working directory), the user types the following to print the calendar on the terminal:

```
calendar 12/01 -fw holidays personal
```

SYNTAX AS A COMMAND:

car request_identifiers {-control_args}

FUNCTION: allows a user to delete a request for an absentee computation that is no longer needed.

ARGUMENTS:

request_identifiers
can be chosen from the following--

path
is the full or relative pathname for the absentee input segment of the request to be cancelled. The star convention can be used to match the entry names of segments.

-entry STR, -et STR
identifies the request to be cancelled by STR, the entryname portion of the absentee input segment pathname. The star convention is allowed.

-id ID
identifies the request to be cancelled by its request ID number. See the MPM Reference Guide for a description of request ID's.

CONTROL ARGUMENTS:

-foreground, -fg
specifies that the foreground absentee queue contains the request(s) to be cancelled.

-queue N, -q N
specifies that absentee queue N contains the request to be cancelled, where N is an integer specifying the number of the queue. The default queue is 3. For convenience in writing exec_coms and abbreviations, the word foreground or fg following the -queue control argument performs the same function as the -foreground control argument. If the -queue, -fg, and -all control arguments are omitted, only the default priority queue is searched.

-all, -a
indicates that all priority queues are to be searched starting with the highest priority queue and ending with the lowest priority queue.

-brief, -bf
suppresses messages telling that a particular request identifier was not found or that requests were cancelled when using star names or the -all control argument.

-sender STR
specifies that only requests from sender STR should be cancelled. One or more request identifiers must also be specified. In most cases, the sender is an RJE station identifier.

-user User_id
specifies the name of the submitter of the request to be cancelled, if it is not the same as the group identifier of the process. The User_id can be specified as Person_id.Project_id, Person_id, or .Project_id. This control argument is primarily for operators and administrators. Both r and d extended access to the queue are required.

ACCESS REQUIRED: The user must have read (r) and delete (d) extended access to the queue.

NOTES: The -queue, -foreground, and -all control arguments are mutually incompatible.

Normally, deletion can be made only by the user who originated the request.

When star names are not used and a single request identifier matches more than one request in the queue(s) searched, none of the requests are cancelled. However, a message is printed telling how many matching requests there are.

If the absentee process has already logged in, the user is given the choice of bumping the job and cancelling the request from the queue, or allowing the job to continue running and remain in the queue. This allows the user to cancel a running absentee process.

cancel_abs_request (car)

EXAMPLES:

The command line:

```
!  car >udd>Demo>Jones>dump>translate
```

deletes the absentee request that the user had made in queue 3 that was associated with the control segment >udd>Demo>Jones>dump>translate.absin.

The command line:

```
!  car >udd>Demo>Jones>doc>**.draft
```

deletes the absentee requests that the user made in queue 3 that were associated with all control segments ending with the ".draft.absin" component combination found in the >udd>Demo>Jones>doc directory.

cancel_cobol_program (ccp)

SYNTAX AS A COMMAND:

ccp names {-control_arg}

FUNCTION: causes one or more programs in the current COBOL run unit to be cancelled.

ARGUMENTS:

names
are the reference names of COBOL programs that are active in the current run unit. If the name specified in the PROG-ID statement of the program is different from its associated namei argument, namei must be in the form refname$PROG-ID.

CONTROL ARGUMENTS:

-retain_data, -retd
leaves the data segment associated with the program intact for debugging purposes. (See "Notes" below.)

NOTES: The results of the cancel_cobol_program command and the execution of the CANCEL statement from within a COBOL program are similar. The only difference is that if a namei argument is not actually a component of the current run unit, an error message is issued and no action is taken; for the CANCEL statement, no warning is given in such a case.

To preserve program data for debugging purposes, the -retain_data control argument should be used. The data associated with the cancelled program is in its last used state; it is not restored to its initial state until the next time the program is invoked in the run unit.

Cancelling ensures that the next time the program is invoked within the run unit, its data is in its initial state. Any files that have been opened by the program and are still open are closed and the COBOL data segment is truncated.

Refer to the run_cobol command for information concerning the run unit and the COBOL runtime environment. Also refer to the related commands display_cobol_run_unit (dcr) and stop_cobol_run (scr).

cancel_daemon_request (cdr)

SYNTAX AS A COMMAND:

cdr request_identifiers {-control_args}

FUNCTION: deletes an I/O daemon request that is no longer needed.

ARGUMENTS:
request_identifiers can be chosen from the following:

path
is the full or relative pathname of the input segment. The star convention is allowed to match the entrynames of segments.

-entry STR, -et STR
identifies the request to be cancelled by STR, the entryname portion of the input segment pathname. The star convention is allowed.

-id ID
identifies the request to be cancelled by its request ID number. See the MPM Reference Guide for a description of Request ID's.

CONTROL ARGUMENTS:

-request_type STR, -rqt STR
indicates that the request to be cancelled is to be found in the queue for the request type identified by the string STR. If this control argument is not given, the default request type is "printer". Request types can be listed by the print_request_types command.

-queue N, -q N
specifies that queue N of the request type contains the request to be cancelled, where N is a decimal integer specifying the number of the queue. If this control argument is omitted, only the default queue for the request type is searched. This control argument is incompatible with the -all control argument.

cancel_daemon_request (cdr)

-all, -a
searches all priority queues for the specified request type starting with the highest priority queue and ending with the lowest priority queue. This control argument is incompatible with the -queue control argument.

-brief, -bf
suppresses messages telling that a particular request identifier was not found or that requests were cancelled when using star names or the -all control argument.

-user User_id
specifies the name of the submitter of the request to be cancelled, if not the group identifier of the process. The User_id can be equal to Person_id.Project_id, Person_id, or .Project_id. Both r and d extended access to the queue are required. This control argument is primarily for operators and administrators.

ACCESS REQUIRED: The user must have o extended access to the queue to cancel their own requests. The user must have r and d extended access to cancel a request entered by another user.

NOTES: When star names are not used and a single request identifier matches more than one request in the queue(s) searched, none of the requests are cancelled. However, a message is printed telling how many matching requests there are.

Normally, deletion can be made only by the user who originated the request.

See the descriptions of the dprint and dpunch commands in this manual.

cancel_daemon_request (cdr)

EXAMPLES: The command line:

```
! cdr >udd>Alpha>Jones>dump>translate.list
```

deletes the request that the user made in queue 3 of the default request type printer, to print the segment >udd>Alpha>Jones>dump>translate.list.

The command line:

```
! cdr >udd>Alpha>Jones>dump>probe.pl1 -request_type punch
```

deletes the request that the user made in queue 3 of request type "punch" to punch the segment >udd>Alpha>Jones>dump>probe.pl1.

The command line:

```
! cdr joe sam *.*
```

cancels the requests to print segments joe, sam, and any requested segments with two-component entrynames in the current working directory in queue 3 of the default request type.

cancel_resource (cnr)

SYNTAX AS A COMMAND:

```
cnr -id reservation_id {-control_arg}
```

FUNCTION: cancels reservations made with the reserve command using the reservation identifier obtainable from the list_resources command.

ARGUMENTS:

reservation_id
must be present and is the reservation identifier of the reservation to be cancelled. It must be preceded by the -id control argument.

CONTROL ARGUMENTS:

-priv
allows a privileged cancellation to be done, such as the cancellation of a reservation belonging to another user. Use of -priv requires access to rcp_sys_.

NOTES: Reservation identifiers can be obtained by using the list_resources command.

cancel_retrieval_request (crr)

SYNTAX AS A COMMAND:

crr request_identifiers {-control_args}

FUNCTION: allows a user to delete a request for a volume retrieval that is no longer needed.

ARGUMENTS:
request_identifiers can be chosen from the following:

path
is the full or relative pathname of the segment or subtree of the retrieval request to be cancelled. The star convention is allowed to match the entrynames.

-entry STR, -et STR
identifies the request to be cancelled by STR, the entryname portion of the segment or subtree pathname. The star convention is allowed.

-id ID
identifies the request to be cancelled specified by its request ID number. See the MPM Reference Guide for a description of request ID's.

CONTROL ARGUMENTS:

-queue N, -q N
specifies that retrieval queue N contains the request to be cancelled, where N is a decimal integer specifying the number of the queue. If this control argument is omitted, only the default priority queue is searched. This control argument is incompatible with the -all control argument.

-all, -a
indicates that all retrieval queues are to be searched starting with the highest priority queue and ending with the lowest priority queue. This control argument is incompatible with the -queue control argument.

-brief, -bf
suppresses messages telling the user that a particular request identifier was not found or that requests were cancelled when using star names or the -all control argument.

cancel_retrieval_request (crr)

-user User_id
specifies the name of the submitter of the requests to be cancelled, if not equal to the group identifier of the process. The User_id can be Person_id.Project_id, Person_id, or .Project_id. Both r and d extended access to the queue are required. This control argument is primarily for operators and administrators.

ACCESS REQUIRED: The user must have read (r) and delete (d) extended access to the queue.

NOTES: Normally, deletion can be made only by the user who originated the request.

When star names are not used and a single request identifier matches more than one request in the queue(s) searched, none of the requests are cancelled. However, a message is printed telling how many matching requests there are.

EXAMPLES:

The command line:

```
! crr >udd>Demo>Jones>dump>translate
```

deletes the retrieval request for the specified segment or subtree that the user had made in queue 3.

canonicalize (canon)

SYNTAX AS A COMMAND:

canon path1 {path2} {-control_arg}

FUNCTION: ensures that the contents of a segment are in canonical form.

ARGUMENTS:

path1
is the pathname of the input segment.

path2
is the pathname of the output segment. If path2 is omitted, path1 is overwritten with the canonicalized contents of the input segment.

CONTROL ARGUMENTS:
can be -tabs in one of the following two forms:

-tabs -every X
inserts tabs at 1+n*X (where n= 1, 2, 3, ...).

-tabs n1,n2,...,n20
inserts tabs at the tab stops specified. Up to 20 tab stops can be given. No spaces are allowed in the list.

NOTES: The command ensures that all characters in a print position are sorted in the proper order and removes all ASCII carriage return (015) characters. When the -tabs control argument is specified, the canonicalize command replaces blank spaces with the appropriate tab stops. Conversely, if the -tabs argument is not specified, horizontal tab stops are replaced by the correct number of blank spaces.

canonicalize (canon)

EXAMPLES: To canonicalize the segment named my_prog and establish tab stops at three specified positions, the user might type:

```
! canon my_prog -tabs 7,21,35
```

To canonicalize the same segment, rename it to new_prog, and set up tab stops at 10-space intervals, the user might type:

```
! canon my_prog new_prog -tabs -every 10
```

To canonicalize the segment old_prog, which already contains tab stops that are now to be replaced with blank spaces, the user can accomplish both operations in one pass by typing:

```
! canon old_prog
```

SYNTAX AS A COMMAND:

```
ceil num
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[ceil num]
```

FUNCTION: returns the smallest decimal integer greater than or equal to its argument.

EXAMPLES:

```
! string [ceil 4.7]
5
! string [ceil -3.5]
-3
```

change_default_wdir (cdwd)

SYNTAX AS A COMMAND:

cdwd {path}

FUNCTION: records a specified directory as the user's default working directory for the duration of the current process or until the next change_default_wdir command is issued.

ARGUMENTS:

path
is the pathname of a directory. If path is not specified, the current working directory becomes the default working directory.

NOTES: The change_default_wdir command is used in conjunction with the change_wdir command. When the change_wdir command is issued with no argument, the default working directory becomes the current working directory.

The original default working directory is the user's home directory upon logging in.

See also the descriptions of the change_wdir (cwd) and print_default_wdir (pdwd) commands.

change_error_mode (cem)

SYNTAX AS A COMMAND:

cem {-control_args}

FUNCTION: controls the amount of information printed by the default handler for system conditions. It determines the length of messages for the life of a process or until it is invoked again in the process.

CONTROL ARGUMENTS:

-brief, -bf
prints only the condition name.

-long, -lg
prints more complete messages. In particular, if the condition was detected in a support procedure, the name of that procedure is printed in addition to the name of the most recent user procedure. If a segment that signalled a condition (or caused it to be signalled) is bound, both the offset relative to the base of the procedure and the offset relative to the base of the segment are printed.

NOTES: If this command is not issued or is issued with no control arguments, the user receives default length error messages. Default length messages are intermediate in length between the brief and long messages.

For a complete discussion of conditions and their handling see the MPM Reference Guide. Refer to the description of the reprint_error command for a similar, but more selective, capability.

change_wdir (cwd)

SYNTAX AS A COMMAND:

cwd {path}

FUNCTION: changes the user's working directory to the directory specified as an argument.

ARGUMENTS:

path
is the pathname of a directory. If path is not specified, the default working directory is assumed.

ACCESS REQUIRED: The user must have s permission on the directory containing path, but no access to path is required.

NOTES: A working directory is a directory in which the user's activity is centered. Its pathname is remembered by the system so that the user need not type the full absolute pathname of segments inferior to that directory.

If path specifies a nonexistent directory, an error message is printed on the user's terminal and the current working directory is not changed.

No access to path is required for this command to be employed. However, once the working directory has been changed, the user can proceed only according to the user's access to path. That is, to effectively use path as a working directory, the user must have sma access permission for path; however, restricted uses are possible in accordance with the access mode attributes on the directory. For example, the user must have at least status permission to list the directory.

See also the descriptions of the change_default_wdir (cdwd) and print_default_wdir (pdwd) commands.

SYNTAX AS A COMMAND:

```
check_iacl {path} {-control_args}
```

FUNCTION: lists segments whose access control lists (ACLs) disagree with the initial ACL for segments.

ARGUMENTS:

path
is the pathname of the directory whose segment ACLs are to be checked against the segment initial ACL. If path is omitted, the working directory is assumed.

CONTROL ARGUMENTS:

-all, -a
lists User_ids in a segment ACL but not contained in the initial ACL; also lists User_ids included in the initial ACL but omitted from a segment ACL. If this control argument is not specified, only User_ids in addition to those in the initial ACL are listed.

-exclude User_id, -ex User_id
excludes the specified User_id from the comparison. Up to ten -exclude control arguments can be specified. The star convention is allowed.

EXAMPLES:

```
! check_iacl

   oldMap.com.runoff
     ACL added: rew Jones.Demo.*
     ACL added: rew Jordan.Work.*

   add_search.com.runoff
     ACL added:  rew Jones.Demo.*
```

check_info_segs (cis)

SYNTAX AS A COMMAND:

cis {-control_args}

FUNCTION: prints a list of new or modified info segments. It saves the current time in the user profile, so that when it is invoked again, it lists segments created or modified since the last invocation.

CONTROL ARGUMENTS:

-date DT, -dt DT
If this argument is specified, check_info_segs uses the date specified by DT instead of the date in the user profile. The DT argument must be acceptable to the convert_date_to_binary_ subroutine (described in the MPM Subroutines). The time of last invocation in the user profile is not updated to the current time.

-long, -lg
If this argument is specified, check_info_segs lists the date-time-entry-modified as well as the name of any segment selected as having been created or modified during the interval in question.

-brief, -bf
If this argument is specified, check_info_segs does not print the names of selected segments and suppresses the comment "no change" if no segments are selected as having been created or modified during the interval in question. This control argument is intended for use with the -call control argument described below.

-no_update, -nud
If this argument is specified, check_info_segs does not place the current time into the user profile.

-call cmdline
If this argument is specified, check_info_segs calls the command processor with a string of the form "cmdline path" for each selected segment, after the name of the segment is typed; path is the absolute pathname of the segment. The cmdline must be enclosed in quotes if it contains blanks.

-pathname spath, -pn spath
If this control argument is specified, check_info_segs assumes that spath is a pathname with one or more asterisks (stars) in the entryname portion. All new or modified segments that

match spath are selected. Refer to "Constructing and Interpreting Names" in the MPM Reference Guide for a discussion of star names.

All specified directories are searched, in the order that the arguments are given. If the -pathname argument is not specified, the default is to search the directories in the "info_segments" search list for "**.info". See "Notes on Search List" section below.

NOTES: The first time check_info_segs is invoked by a particular user, it just initializes the time in the user profile to the current time, prints a comment, and does not list any segments. If a profile does not exist, check_info_segs creates one in the user's home directory. The profile segment has the name Person_id.profile, where Person_id is the Person_id given at login time.

The check_info_segs command checks the date-time-entry-modified for any segment pointed to by a link, not the time the link was modified.

The check_info_segs command cannot detect that a segment has been deleted since the last invocation of the command.

NOTES ON SEARCH LIST:

The check_info_segs command uses the "info_segments" search list which has the synonyms "info_segs" and "info". The default "info_segments" search list is:

```
>doc>iml_info
>doc>info
```

These directories contain info segments provided by the site and those supplied with the system. Type "psp info" to see what the current "info" search list is. For more information about search lists, see the search facility commands, and in particular, the add_search_paths description in this manual.

EXAMPLES: To check for info segments modified since the specified date, type:

```
! cis -date "07/01/79 0900."
```

check_info_segs (cis)

To print all modified info segments, type:

```
! cis -call print -brief
```

The -brief control argument is given to check_info_segs to suppress duplicate printing of segment names since the print command types the segment name in the heading.

To print just the first block of any modified info segment, type:

```
! cis -call "answer no help"
```

To check for all modified segments in a project-maintained directory >udd>Project_id>doc as well as the default directories, type the following two command lines:

```
! asp info_segments >udd>Project_id>doc
! cis
```

close_file (cf)

SYNTAX AS A COMMAND:

cf {-control_arg} filenames

FUNCTION: closes specified FORTRAN and PL/I files. It closes all open FORTRAN and PL/I files if the -all control argument is specified.

ARGUMENTS:

filenames
are the names of open FORTRAN or PL/I files.

CONTROL ARGUMENTS:

-all, -a
closes all open files. In this case, no filename appears.

NOTES: The format of a FORTRAN file name is filenn where nn is a two-digit number other than 00; e.g., file05. PL/I file names are selected by the user and can have any format.

If a specified file cannot be found, an error message is printed indicating the name of the file. The rest of the specified files are closed.

For each filename, all PL/I files of that name and, if applicable, the FORTRAN file of that name are closed.

The command "close_file -all" does not affect I/O switches that are not associated with FORTRAN or PL/I files.

SYNTAX AS A COMMAND:

```
cobol path {-control_args}
```

FUNCTION: invokes the COBOL compiler to translate a segment containing the text of a COBOL source program into a Multics object segment.

ARGUMENTS:

path
is the pathname of a COBOL source segment to be translated by the COBOL compiler. If path does not have a suffix of cobol, one is assumed. However, the suffix cobol must be the last component of the name of the source segment. If the expand_cobol_source command is used to create a new segment with the suffix ex.cobol, a check is made to see if this segment exists and it is used.

CONTROL ARGUMENTS:

-brief, -bf
causes error messages written to the user_output I/O switch to contain only an error number and statement identification, once the full message has been given on the first occurrence. In the normal, nonbrief mode, an explanatory message is printed for each occurrence.

-check, -ck
is used for syntactic and semantic checking of a COBOL program. No code is generated.

-expand, -exp
accepts a source segment in the format acceptable to the expand_cobol_source command. It expands the source segment by evaluating COPY and REPLACE statements. If the segment to be translated has the suffix ex.cobol, this control argument is ignored.

-format, -fmt
accepts a source segment in the format acceptable to the expand_cobol_source command. If the segment to be translated has the suffix ex.cobol, this control argument is ignored.

-levelN, -levN
causes severity three L-type diagnostics to be written to the user_output I/O switch whenever a COBOL source line contains a language construct outside the subset specified by N. The

value N can be one through five, corresponding to the four levels specified by the Federal Information Processing Standards Publication, December 1, 1975 (FIPS PUB 21-1) and to the extended version of COBOL supported by Multics. These values are:

1 low level
2 low intermediate level
3 high intermediate level
4 high level
5 Multics COBOL extensions

If a program compiles without any L-type diagnostics, it means the program is an acceptable subset of Multics COBOL at the level requested. The default is level 5.

-list, -ls
produces a source program listing with symbols, followed by an assembly-like listing of the compiled object program. Use of the -list control argument significantly increases compilation time and should be avoided whenever possible by using the -map control argument.

-map
produces a source program listing with symbols, followed by a map of the object code generated by this compilation. The -map control argument produces sufficient information to allow the user to debug most problems online.

-profile, -pf
generates additional code to meter the execution of individual statements. Each statement in the object program contains an additional instruction to increment an internal counter associated with that statement. After a program has been executed, the profile command can be used to print the execution counts.

-runtime_check, -rck
produces an object program in which parameters are validated according to number and type, performs bounds checking on all subscripted referenced, performs string range checking on all variable length string references, and verifies the validity of every index name modification.

-severityN, -svN
causes error messages whose severity is less than N (where N is 1, 2, 3, or 4) to not be written to the user_output I/O switch. All errors are written into the listing. If this control argument is not given, a severity level of 2 is

assumed. See the description of severity levels under "Notes on Error Diagnostics" below.

-table, -tb
generates a full symbol table for use by symbolic debuggers. The symbol table is part of the symbol section of the object program and consists of two parts: a statement table that gives the correspondence between source line numbers and object locations and an identifier table that contains information about every identifier actually referenced by the source program. The table appears in the symbol section of the object segment produced by the compilation. This control argument usually causes the object segment to become significantly longer. If the -format control argument is given with the -table control argument, the symbolic debuggers are not able to display the source statements.

-temp_dir path, -td path
creates the compiler's internal work files in the specified directory rather than in the process directory. This control argument may be necessary for very large source files (over approximately 3000 lines) that incur record quota overflow in the process directory during compilation.

-debug, -db
leaves the work files generated by the compiler intact after a compilation. This control argument is used for debugging the compiler. The command cobol$clean_up can be used to discard these files. Also, this causes severity 4 errors to not unwind and abort the compilation, but rather to invoke a new level of the command processor at the point of the error.

-time, -tm
prints the time (in seconds) and the number of page faults taken by each phase of the compiler; prints the total time at the end of the compilation. This information is directed to the user_output I/O switch.

NOTES: The only result of invoking the cobol command without control arguments is to generate an object segment.

A normal compilation produces an object segment and leaves it in the user's working directory. If an entry with that name already exists in the directory, its access control list (ACL) is saved and given to the new copy of the object segment. Otherwise, the user is given re access to the segment with ring brackets v,v,v where v is the validation level of the process that is active when the object segment is created.

If the user specifies the -map or -list control arguments, the cobol command creates a listing segment in the working directory and gives it a name consisting of the entryname portion of the source segment with a suffix of list rather than cobol (e.g., a source segment named business.cobol would have a listing segment named business.list). The ACL is set as described for the object segment except that the user is given rw access to it when newly created. Previous copies of the object segment and the listing segment are replaced by the new segments created by the compilation.

A listing segment can also be produced. These segments are placed in the user's working directory.

This command cannot be called recursively.

For information on COBOL, refer to the Multics COBOL Users' Guide, Order No. AS43 and the Multics COBOL Reference Manual, Order No. AS44. See the description of the profile command in this document.

NOTES ON ERROR DIAGNOSTICS: The COBOL compiler can diagnose and issue messages for about 800 different errors. These messages are graded in severity as follows:

1 Warning only. Compilation continues without ill effect.

2 Correctable error. The compiler attempts to remedy the situation and continues, possibly without ill effect. The assumptions the compiler makes in remedying the situation, however, do not necessarily guarantee the right results.

3 Uncorrectable but recoverable error. That is, the program is definitely in error and no meaningful object code can be produced, but the compiler can continue executing and diagnosing further errors.

4 Unrecoverable error. The compiler cannot continue beyond this error. A message is printed and control is returned to the cobol command. The command writes an abort message on the error_output I/O switch and returns to its caller.

As indicated above, the user can set the severity level so as not to be bothered by minor error messages. The user can also specify the -brief control argument so that the message is shorter. Since the default severity level is 2, the user must explicitly specify the -severity1 (or -sv1) control argument when invoking the cobol command to have warning messages printed. Neither the -severityN nor -brief control argument has any effect on the contents of the listing segment if one is produced.

An example of an error message in its long form is:

```
22          use after error procedure on extend.
                                                1
** 1 5-250 A use procedure has already been associated with
this processing mode.
```

If the -brief control argument is specified and message 5-250 has previously been given in its long form, the user instead sees:

```
22          use after error procedure on extend.
                                                1
** 1 5-250
```

If the user has set the severity level to 3, no message is printed at all. Notice that the number of asterisks immediately preceding the error indicator corresponds to the severity level of the error.

If a listing is produced, the error messages appear interspersed with the lines of the source program. No more than 300 messages are printed in the listing.

NOTES ON LISTING: The listing created by the cobol command is a line-numbered image of the source segment with diagnostics interspersed. This is followed by a cross-reference table of all the names defined within the program. Following the cross-reference table is the object code map, which gives the starting location in the text segment of the instructions for each statement in the program. The map is sorted by ascending storage locations. Finally, the listing contains an assembly-like list of the object code produced. The executable instructions are grouped under an identifying header, which contains the source statement that produced the

instruction. Opcode, pointer-register, and modifier mnemonics are printed alongside the octal instruction. If the address field of the instruction uses the IC (self-relative) modifier, the absolute text location corresponding to the relative address is printed on the remarks field of the line.

cobol_abs (cba)

SYNTAX AS A COMMAND:

cba paths {cobol_args} {dp_args} {abs_control_args}

FUNCTION: submits an absentee request to perform COBOL compilations. The absentee process for which cobol_abs submits a request compiles the segments named and prints and deletes the listing segment.

ARGUMENTS:

paths
are the pathnames of segments to be compiled.

cobol_args
can be one or more control arguments accepted by the cobol command.

dp_args
can be one or more control arguments (except -delete, -dl) accepted by the dprint command.

LIST OF ABSENTEE CONTROL ARGUMENTS:
abs_control_args can be chosen from the following:

-queue N, -q N
specifies in which priority queue the request is to be placed (N < 3). The default queue is 3; the listing segment is printed in dprint queue N.

-hold, -hd
specifies that cobol_abs should not print or delete the listing segment.

-limit N, -li N
places a limit on the CPU time used by the absentee process. The parameter N must be a positive decimal integer specifying the limit in seconds. The default limit is defined by the site for each queue. An upper limit is defined by the site for each queue on each shift. Jobs with limits exceeding the upper limit for the current shift are deferred to a shift with a higher limit.

-output_file path, -of path
specifies that absentee output is to go to the segment whose pathname is path.

cobol_abs (cba)

NOTES: Control arguments and segment pathnames can be mixed freely and can appear anywhere on the command line after the command. All control arguments apply to all segment pathnames. If an unrecognizable control argument is given, the absentee request is not submitted.

Unpredictable results can occur if two absentee requests are submitted that could simultaneously attempt to compile the same segment or write into the same absout segment.

When doing several compilations, it is more efficient to give several segment pathnames in one command rather than several commands. With one command, only one process is set up. Thus the dynamic intersegment links that need to be snapped when setting up a process and when invoking the compiler need be snapped only once.

If the -output_file control argument is not specified, an output segment, path.absout, is created in the user's working directory (if more than one path is specified, only the first is used).

If none of the segments to be compiled can be found, no absentee request is submitted.

SYNTAX AS A COMMAND:

collate

SYNTAX AS AN ACTIVE FUNCTION:

[collate]

FUNCTION: returns the 128 characters of the ASCII character set in collating sequence.

SYNTAX AS A COMMAND:

collate9

SYNTAX AS AN ACTIVE FUNCTION:

[collate9]

FUNCTION: returns a character string containing all possible 9-bit bit patterns rather than just the 128 ASCII characters, therefore, making the returned string 512 characters long.

SYNTAX AS A COMMAND:

compare path1{|offset1} path2{|offset2} {-control_args}

FUNCTION: compares two segments and lists their differences. The comparison is a word-by-word check and can be made with a mask so that only specified parts of each word are compared.

ARGUMENTS:

path1, path2
are the pathnames of the segments to be compared. The equal convention is allowed for path2.

offset1, offset2
are octal offsets within the segments to be compared. The comparison begins at the word specified or at the first word of the segment if no offset is specified. If an offset is omitted, the vertical bar should also be omitted.

CONTROL ARGUMENTS:

-brief, -bf
prints only the first and last words of each block of discrepancies that is four or more words in length. The default is to print all discrepancy words.

-length N, -ln N
the comparison should continue for no more than N (octal) words.

-long, -lg
prints all discrepancy words, unlike -brief. This is the default.

-mask N
the octal mask N is to be used in the comparison. If N is less than 12 octal digits, it is padded on the left with zeros.

NOTES: The maximum number of words to be compared is the word count of the first segment minus its offset or the word count of the second segment minus its offset, whichever is greater. If the -length control argument is supplied, comparison stops after the specified number of words. If the segments are of unequal length, the remaining words of the longer segment are printed as discrepancies. The word count of a segment is computed by dividing

printed as discrepancies. The word count of a segment is computed by dividing the bit count plus 35 by 36. If the word count minus the offset is less than zero, an error message is printed and the command is aborted.

Any discrepancies found by the command are listed in the following format:

offset	contents	offset	contents
4	404000000002	4	000777000023
6	404000000023	6	677774300100

To compare segments containing only ASCII character-string data, use the compare_ascii command described in this manual.

compare_ascii (cpa)

SYNTAX AS A COMMAND:

cpa paths {-control_args}

FUNCTION: compares ASCII segments and prints any differences.

ARGUMENTS:

paths
are the pathnames of the segments to be compared. Up to six segments can be compared, in addition to the original if one is supplied. The equal convention can be used in any pathname except the first one on the command line, which is assumed to be the original unless otherwise specified.

CONTROL ARGUMENTS:

-original pathA, -orig pathA
specifies the pathname pathA of the original segment of which the others are modified versions.

-no_original, -no_orig
indicates that no original segment is supplied. If neither -no_original nor -original is given, the first pathname on the command line is assumed to be the original.

-minchars NN
specifies the minimum number of characters that must be identical for compare_ascii to assume that it has found the end of a difference. The default is 20 characters. See "Notes" below.

-minlines NN
specifies the minimum number of lines that must be identical for compare_ascii to assume that it has found the end of a difference. The default is two lines. See "Notes" below.

-totals, -tt
prints only the totals line, giving the number of differences and the number of changed lines. The default is to print discrepancies and totals line.

-no_totals, -ntt
does not print the totals line.

-header, -he
prints a heading, giving the full pathname and identifying letter of each segment. This heading is not printed by

default.

-print_new_lines, -pnl
prints only new lines. New lines are lines found in one or more of the modified versions but not in the original. An original must be supplied if this argument is used.

-no_numbers, -nnb
does not print identifying letter and line numbers preceding the lines from the segments being compared. The default is to print them.

NOTES: The output is organized with the assumption that the pathA segment was edited to produce pathB. This command prints lines that were added, replaced, or deleted; it identifies each line by line number within the respective segment and also by the letter A or B to indicate which segment the line is from (A for pathA and B for pathB).

Values for minchars and minlines can be specified without being preceded by control arguments. The order is: minchars minlines.

The values of minchars and minlines control the size of displayed differences. Large values for these parameters cause small, closely-spaced differences to be displayed as one large difference, while very small values (such as -minlines 1 -minchars 2) will cause small changes to be displayed individually but might also cause large differences to be broken down into small parts, thereby giving a misleading picture of what was actually done to produce the modified versions. The user should adjust these parameters to produce the most useful results.

EXAMPLES: The examples of compare_ascii usage below are based on the segments yesterday.menu and today.menu displayed here side by side.

yesterday.menu	today.menu
Breakfast Menu:	Breakfast Menu:
Juice	Juice
Toast	Toast
Eggs	Eggs
Luncheon Menu:	Luncheon Menu:
Hot dogs	Hamburger

compare_ascii (cpa)

```
     Milk                                    Milk
     French fries                            Salad
Supper Menu:                                 French fries
     Steak                              Supper Menu:
     Baked potato                            Chicken
     Coffee                                  Rice
                                             Coffee
```

The default operation of compare_ascii is illustrated by the command line:

```
! cpa yesterday.menu today.menu

   A6          Hot dogs
   A7          Milk
   Changed by B to:
   B6          Hamburger
   B7          Milk
   B8          Salad


   A10         Steak
   A11         Baked potato
   Changed by B to:
   B11         Chicken
   B12         Rice

   Comparison finished: 2 differences, 9 lines.
```

The following command line shows the use of the -original, -header, -minlines, and -minchars control arguments. Notice that the lower values of minlines and minchars isolate the two changes within the Luncheon menu.

```
! cpa today.menu -orig yesterday.menu -he -minchars 5
     -minlines 1

   A >udd>m>Jones>yesterday.menu (original)
   B >udd>m>Jones>today.menu (new)

   A6          Hot dogs
   Changed by B to:
   B6          Hamburger


   Inserted in B:
   B8          Salad
   Preceding:
   A8          French fries
```

compare_ascii (cpa)

```
A10             Steak
A11             Baked potato
Changed by B to:
B11             Chicken
B12             Rice

Comparison finished: 3 differences, 7 lines.
```

In the following example the printing of line numbers, old lines, and the totals line have been suppressed, giving better visibility to what is new in today.menu.

```
! cpa yesterday.menu today.menu -pnl -nnb -ntt -minchars 5
      -minlines 1

Hamburger
Salad
Chicken
Rice
```

SYNTAX AS A COMMAND:

contents path

SYNTAX AS AN ACTIVE FUNCTION:

[contents path]

FUNCTION: returns the contents of a segment as a character string. Newline characters in the segment are changed to blanks in the string.

EXAMPLES: Assume that the segment named distribution contains a list of names (each person's name on a separate line).

The command line:

! dp -ds ([contents distribution]) output

prints one copy of the segment output for each name on the list, using the name as the destination.

convert_characters (cvc)

SYNTAX AS A COMMAND:

```
   cvc key1 {oldpath} {newpath}
or:
   cvc key2 char_string
```

FUNCTION: allows the rapid editing of a segment in the case where a one-for-one replacement of certain characters by certain other characters must be done. An example of its use is the conversion of all uppercase characters in a segment to lowercase characters. Keywords specify the conversion to take place. For certain of the keys convert_characters maintains a from_string and a to_string that define the conversion to be made. The converted segment is the same as the original except that every instance of the i'th character of from_string present in the original segment is replaced by the i'th character of to_string.

The conversion for the key "sp" uses a from_string and to_string that must have been previously set by use of the "from" and "to" keys.

ARGUMENTS:

key1
any of the keys listed below in "List of keywords".

oldpath
the pathname of a segment to be converted. If this argument is omitted, the from_string and to_string related to key1 are printed.

newpath
the pathname of the output segment. If this argument is omitted, newpath is assumed to be the same as oldpath, and the converted copy replaces the original.

key2
either "to" or "from" to set to_string or from_string for the "sp" key.

char_string
is the string to be set as to_string or from_string. If it contains blanks, it must be enclosed in quotes.

convert_characters (cvc)

LIST OF KEYWORDS:

lc
converts alphabetic characters to lowercase.

uc
converts alphabetic characters to uppercase.

mp
converts from Multics PL/1 format to IBM 360 PL/1.

bcd
converts BCD special characters to ASCII/EBCDIC equivalents.

dart
converts Multics special characters to corresponding Dartmouth special characters as follows:

^	'
¬	=
>	"
+	<
=	>
'	:
{	+
"	?
?	

sp
uses conversion strings set earlier by the from and to keys: cvc from char_string1;cvc to char_string2

NOTES: The most recent setting of from_string and to_string in the user's process is used for conversion with the "sp" key. No conversion is attempted for the "sp" key unless both the from_string and the to_string are of the same non-zero length. Any character not present in the from_string is not changed.

SYNTAX AS A COMMAND:

cp path1_1 {path2_1 ... path1_n path2_n} {-control_args}

FUNCTION: causes copies of specified segments and multisegment files to be created in the specified directories with the specified names. Access control lists (ACLs) and multiple names are optionally copied.

ARGUMENTS:

path1_i
is the pathname of a segment or multisegment file to be copied. If path1 is the name of a link, the command copies the target of the link. The star convention is allowed.

path2_i
is the pathname of a copy to be created from path1_i. If the last path2 argument is not given, the copy is placed in the working directory with the entryname of path1_n. The equal convention is allowed.

CONTROL ARGUMENTS:

-acl
copies the ACL.

-all, -a
copies multiple names and ACLs.

-brief, -bf
suppresses the warning messages "Bit count inconsistent with current length..." and "Current length is not the same as records used...".

-chase
copies the targets of links that match path1. See "NOTES" for the default action.

-long, -lg
prints warning messages as necessary. This is the default.

-name, -nm
copies multiple names.

-no_acl
does not copy the ACL. This is the default.

-no_chase
does not copy the targets of links that match path1. See "NOTES" for the default action.

-no_name, -nnm
does not copy multiple names. This is the default.

ACCESS REQUIRED: Read access is required for path1i. Status permission is required for the directory containing path1i. Append permission is required for the directory containing path2i. Modify permission is required if the -name, -acl, or -all control argument is used.

NOTES: The control arguments can appear once anywhere in the copy command line after the command name and apply to the entire copy command line.

The default for chasing links depends on path1. If path1 is not a starname, links are chased by default. If path1 is a starname, links are not chased.

If the ACL of a segment or multisegment file is being copied, the initial ACL of the target directory has no effect on the ACL of the segment or multisegment file after it has been copied into that directory. The ACL remains exactly as it was in the original directory.

Since two entries in a directory cannot have the same entryname, special action is taken by this command if the name of the segment or multisegment file being copied (specified by path1i) already exists in the directory specified by path2i. If the entry being copied has an alternate name, the entryname that would have resulted in a duplicate name is removed and the user is informed of this action; the copying operation then takes place. If the entry being copied has only one entryname, the entry that already exists in the directory must be deleted to remove the name. The user is asked if the deletion should be done; if the user answers "no", the copying operation does not take place.

The copy command prints a warning message if the bit count of path1i is less than its current length or if the current length is greater than the number of records used. These

warnings are suppressed by the use of the -brief control argument.

EXAMPLES:

The command line:

```
! copy >old_dir>fred.list george.=
```

copies segment or multisegment file named fred.list in the directory >old_dir into the working directory as george.list.

SYNTAX AS A COMMAND:

copy_acl path1_1 path2_1 {... path1_n path2_n}

FUNCTION: copies the access control list (ACL) from one file or directory to another, replacing the current ACL if necessary.

ARGUMENTS:

path1_i
is the pathname of a file or directory whose ACL is to be copied. The star convention is allowed.

path2_i
is the pathname of a file or directory onto which the initial ACL is to be copied. The equal convention is allowed.

copy_cards (ccd)

SYNTAX AS A COMMAND:

ccd deck_name {new_deck_name}

FUNCTION: copies specified card image segments from system pool storage into a user's directory. The segments to be copied must have been created using the Multics card input facility.

ARGUMENTS:

deck_name
is the name that was entered on the deck_id card when the card deck was submitted for reading. The star convention is allowed.

new_deck_name
is the pathname of the segment in which the matching card image segment is to be placed. If omitted, the working directory and deck_name are assumed. The equal convention is allowed.

NOTES: See the description of the card input facility in the MPM Reference Guide for the format of the control cards needed when submitting a card deck to be ready by system operations. The user process executing this command must have the proper access to the card image segment in order to perform the copy. When there are multiple copies of the same deck in pool storage, all are copied.

When deck_name is a starname and there are several matching card image segments in pool storage to which the user has access, all are copied.

When an attempt is made to read a card deck having the same name as some previously read deck still in pool storage, a numeric suffix is added to the name of the new deck, e.g., "deck_name.1". Repeated name duplications cause successively larger numeric suffixes to be used. (Name duplications can only occur for decks of the same access class submitted by the same user.) The copy_cards command informs the user of such duplications (if any) and retrieves all copies of the specified deck.

copy_cards (ccd)

Only those card decks having an access class equal to the user's current authorization can be copied. Other decks are not found.

EXAMPLES:

The command line:

```
! ccd my_deck
```

copies the user's card image segment named my_deck from the card pool storage into the user's current working directory.

copy_characters (cpch)

SYNTAX AS A COMMAND:

```
cpch str N
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[cpch str N]
```

FUNCTION: returns a quoted string containing N copies of a specified string.

EXAMPLES:

```
! string [cpch "1 2 3 " 3]
  1 2 3 1 2 3 1 2 3
```

SYNTAX AS A COMMAND:

cpd source_dir {target_dir} {-entry_type_keys} {-control_args}

FUNCTION: copies a directory and its subtree to another point in the hierarchy.

ARGUMENTS:

source_dir
is the pathname of a directory to be copied. The star convention is allowed.

target_dir
is the pathname of the copy of the source_dir. The equal convention is allowed. If target_dir is not specified, the copy is placed in the working directory with the entryname of source_dir. If the target_dir does not exist, it is created.

entry_type_keys
control what type of storage system entries in the subtree are copied. If no entry_type_key is specified, all entries are copied. The keys are:

-branch, -br
-directory, -dr
-file, -f
-link, -lk
-multisegment_file, -msf
-non_null_link, -nnlk
-segment, -sm

If one or more entry_type_keys are specified, but not the -directory key, the subtree of source_dir is not walked.

CONTROL ARGUMENTS:

-brief, -bf
suppresses the printing of warning messages such as "Bit count is inconsistent with current length" and "Current length is not the same as records used".

-force
executes the command, when target_dir already exists, without asking the user. If the -force control argument is not specified, the user is queried.

-replace, -rp
deletes the existing contents of target_dir before the copying begins. If target_dir is non-existent or empty, this control argument has no effect. The default is to append the contents of source_dir to the existing contents of target_dir.

-acl
gives the ACL on the source_dir entry to its copy in target_dir. Although initial ACLs are still copied, they are not used in setting the ACL of the new entries when this control argument is specified. See "Notes on Access Provision" below for further discussion.

-primary, -pri
copies only primary names. If the -primary control argument is not specified, all the names of the selected entries are copied.

-no_link_translation, -nlt
copies links with no change. The default is to translate links being copied. If there are references to the source directory in the link pathname of a link being copied, the link pathname is changed to refer to the target directory.

-chase
copies the target of a link. The default is not to chase links. Chasing the links eliminates link translation.

NOTES: The user can specify that portions of the subtree be copied and can control the processing of links. See also the copy, move, and move_dir commands in this manual.

ACCESS REQUIRED: Status permission is required for source_dir and all of the directories in its tree. Status permission is required for the directory containing source_dir. Read access is required on all files under source_dir. Append and modify permission are required for the directory containing target_dir if target_dir does not exist prior to the invocation of the copy_dir command. Modify and append permission are required on target_dir if it already exists. This command does not force access.

NOTES ON ACCESS PROVISION: If the -acl control argument is not specified, the system default ACLs are added, then the initial ACL for the containing directory is applied (which may change the system supplied ACL). Initial ACLs are always copied for the current ring of execution.

NOTES ON THE EXISTENCE OF target_dir: If target_dir already exists and -force is not specified, the user is so informed and asked if processing should continue. If target_dir is contained in or contains source_dir, an appropriate error message is printed and control is returned to command level. Otherwise, the contents of source_dir either are appended to or replace the contents of target_dir. (See the -replace control argument.)

NOTES ON STAR AND EQUAL CONVENTIONS: The star convention in source_dir matches only directory names and copies them. Matching names associated with other storage types are ignored.

NOTES ON NAME DUPLICATIONS: Since two entries in a directory cannot have the same entry name, this command takes special action if the entryname of the entry being copied already exists in the directory specified by target_dir. If the entry is a directory, it is handled in the same fashion as duplication between source_dir and target_dir is handled, unless the existing entry in target_dir is not also a directory. In this case the entryname duplication is treated the same as non-directory entries. The procedure for non-directory entries is the standard system technique. See the copy command in this manual.

If the -replace control argument is specified or target_dir does not exist, name duplication does not occur.

NOTES ON LINK TRANSLATION: If part of the tree is not copied (by specifying a storage system entry key), problems with link translation may occur. If the link target in the source_dir tree was in the part of the tree not copied, there may be no corresponding entry in the target_dir tree. Hence, translation of the link causes the link to become null.

copy_dir (cpd)

EXAMPLE:

The command line:

```
! cpd old_source new_source -segment -acl
```

copies all the segments with their ACLs in the directory old_source to the directory new_source.

The command line:

```
! cpd old_user new_user -branch
```

copies all the segments, directories and multisegment files from the directory old_user to the directory new_user (no links are copied).

copy_file (cpf)

SYNTAX AS A COMMAND:

cpf in_control_arg out_control_arg {-control_args}

FUNCTION: copies records or lines from an input file to an output file. The copy command makes an exact duplicate of the input file, whereas copy_file produces an output file that has been restructured for maximum compactness. (See the description of the copy command in this manual.)

ARGUMENTS:

LIST OF in_control_args:
the input file from which records or lines are read can be specified by either an I/O switch name or an attach description. (See "Notes" below.)

-input_switch STR, -isw STR
specifies the input file by means of an already attached I/O switch name, where STR is the switch name.

-input_description STR, -ids STR
specifies the input file by means of an attach description STR. STR must be enclosed in quotes if it contains spaces or other command language characters.

LIST OF out_control_args:
the output file to which the records or lines are written can be specified by either an I/O switch name or an attach description. (See "Notes" below.)

-output_switch STR, -osw STR
specifies the output file by means of an already attached I/O switch name, where STR is the switch name.

-output_description STR, -ods STR
specifies the output file by means of an attach description STR. STR must be enclosed in quotes if it contains spaces or other command language characters.

CONTROL ARGUMENTS:

-keyed
copies both records and keys from a keyed sequential input file to a keyed sequential output file. The default is to

copy_file (cpf)

copy records from an input file (either keyed or not) to a sequential output file. (See "Notes on Keyed Files" below.)

-from N, -fm N
copies beginning with the Nth record or line of the input file, where N is a positive integer. The default is to begin copying with the "next record." (See "Notes" below.)

-start STR, -sr STR
copies beginning with the record whose key is STR, where STR is 256 or fewer ASCII characters. The default is to begin copying with the "next record."

-to N
copies until the Nth record or line has been copied or the input file is exhausted, whichever occurs first, where N is a positive integer greater than or equal to the N given with the -from control argument. This control argument can only be specified if -from is also specified. The default is to perform copying until the input file is exhausted.

-stop STR, -sp STR
copies until the record whose key is STR has been copied or the input file is exhausted, whichever occurs first, where STR is 256 or fewer ASCII characters. This control argument can be specified without specifying the -start control argument. However, if -start is specified, the STR given with -stop must be greater than or equal to (according to the ASCII collating sequence) the STR given with -start. The default is to perform copying until the input file is exhausted.

-count N, -ct N
copies until N records or lines have been copied or the input file is exhausted, whichever occurs first, where N is a positive integer. The default is to perform copying until the input file is exhausted.

-all, -a
copies until the input file is exhausted. This is the default.

-brief, -bf
suppresses an informative message indicating the number of records or lines actually copied.

-long, -lg
prints an informative message indicating the number of records or lines actually copied. This is the default.

-input_mode x, -imode x
specifies the opening mode for the input file. If copy_file

copy_file (cpf)

opens the file, it uses this mode. If the file is already open, this mode must be consistent with the open mode of the file (e.g., stream_input is consistent with stream_input_output). Allowable values for x are:

keyed_sequential_input, ksqi	sequential_input_output, sqio
keyed_sequential_output, ksqo	stream_input, si
sequential_input, sqi	stream_output, so
sequential_output, sqo	stream_input_output, sio

-output_mode x, -omode x
specifies the opening mode of the output file. See -input_mode above for restrictions.

-character, -ch
specifies that any positioning of stream files done by copy_file is to be done in terms of characters rather than lines.

NOTES ON UNSTRUCTURED FILES: With the use of -input_mode and -output_mode, it is possible to specify the processing of unstructured (stream) files with copy_file. Three possibilities involving unstructured files exist: stream to record, record to stream, and stream to stream.

Stream to record copying involves reading input lines and writing them as records. Record to stream copying involves reading records, appending a newline character and writing these characters to the output stream. Stream to stream copying involves simply reading characters from the input stream and writing them to the output stream.

In stream to record and record to stream copying, -nnl inhibits the copying or addition of newline characters.

In copies involving stream input, -from, -to, and -count specify positions in terms of lines unless -character has been specified, in which case positioning is in terms of characters.

NOTES ON KEYED FILES: The copy_file command can copy a keyed sequential file to produce an output file that has been restructured for maximum compactness as a keyed file or as though it were purely sequential. By default, the command copies only records and does not place keys in the output

copy_file (cpf)

file. To copy the keys, the -keyed control argument must be used. When -keyed is used, the input file must be a keyed sequential file. Whether keys are copied or not, control arguments can be used to delimit the range of records to be copied (i.e., -start, -stop, -from, -to, -count). Copying is always performed in key order.

NOTES: The input and output files can be any combination of structured or unstructured files. The input file can be copied either partially or in its entirety.

If either the input or output specification is an attach description, it is used to attach a uniquely named I/O switch to the file. The switch is opened, the copy performed, and then the switch is closed and detached. Alternately, the input or output file can be specified by an I/O switch name. Either the io_call command or iox_ subroutine can be used to attach the file prior to the invocation of the copy_file command. (See the description of the io_call command in this manual and the iox_ subroutine in the MPM Subroutines.)

If the input file is specified by an I/O switch name and the switch is not open, the copy_file command opens it for (keyed_)sequential_input or stream_input, performs the copy, and closes it. If the switch is already open when the copy_file command is invoked, the opening mode must be sequential_input, sequential_input_output, keyed_sequential_input, keyed_sequential_update, stream_input, or stream_input_output. The switch is not closed after the copy has been performed.

The "next record" or "next byte" must be defined if neither the -start nor -from control argument is used to specify an absolute starting position within the input file. If the I/O switch is opened by the copy_file command, the next record is the first record of the file; otherwise, the next record is that record at which the file is positioned when the copy_file command is invoked. If the -character control argument has been specified and -from, -to, or -count has been specified and the input file is a stream, positioning is performed in terms of characters rather than lines or records.

If the output file is specified by an I/O switch name and the switch is not open, the copy_file command opens it for (keyed_)sequential_output, performs the copy, and closes it.

copy_file (cpf)

If the switch is already open when the copy_file command is invoked, the opening mode must be sequential_output, sequential_input_output, keyed_sequential_output, keyed_sequential_update, direct_output, direct_update, stream_output, or stream_input_output. (In update mode, output file records with keys that duplicate input file records are rewritten.) The switch is not closed after the copy has been performed.

The -from and -start control arguments are mutually exclusive. The -to, -stop, -count, and -all control arguments are mutually exclusive. The -nnl control argument is mutually exclusive with stream to stream or record to record copying. The -start and -stop control arguments are mutually exclusive with stream input. The -brief and -long control arguments are mutually exclusive. The informative message, printed by default, appears as one of the following:

```
  345 records copied.
  345 records read; 4002 characters written.
 4002 characters read; 345 records written.
 4002 characters copied.
```

EXAMPLES: To copy an entire file from an already attached file to the segment in_copy, type:

```
! cpf -isw in -ods "vfile_ in_copy"
```

To print the first 13 records of a tape file, type:

```
! cpf -ct 13 -ids "tape_ansi_ 887677 -name TEST21 -ret all"
   -osw user_output -ct 13
```

To copy 13 records from an already attached file to another already attached file, starting with the 56th record of the input file, type:

```
! cpf -isw in -osw out -from 56 -ct 13
```

To copy records 43 through 78 from an already attached file to an already attached file, type:

```
! cpf -isw in -osw out -from 43 -to 78
```

copy_file (cpf)

To copy all but the first seven records from segment testdata.11 to an already attached file, type:

```
! cpf -ids "vfile_ testdata.11" -osw out -fm 8
```

To copy an entire keyed sequential file with keys, type:

```
! cpf -isw in -osw out -all -keyed
```

To copy 13 records of a keyed sequential file starting with the record whose key is ASD66 to a sequential output file, the following line is typed. (No keys are copied.)

```
! cpf -isw in -osw out -sr ASD66 -ct 13
```

To copy the records and keys from a keyed sequential file up to and including the record whose key is bb"bb, type:

```
! cpf -keyed -isw in -osw out -sp "bb""bb"
```

To copy a tape_mult_ tape to another tape, type:

```
! cpf -ids "tape_mult_ m2156" -ods "tape_mult_ m2752 -write"
   -imode si -omode so
```

To copy each line in the segment test to a record on an output tape, type:

```
! cpf -ids "vfile_ test" -ods "tape_ansi_ 792561 -write
    -format vb -bk 4000" -imode si
```

To copy 20 records of an already attached keyed sequential file, starting with the record whose key is ASD60 to a stream output file without appending newlines, type:

```
! cpf -isw in -osw out -sr ASD60 -ct 20 -omode so -nnl
```

To copy 27 lines without their newline characters from an already attached stream file to an already attached output file, type:

```
! cpf -isw in -osw out -imode si -nnl
```

SYNTAX AS A COMMAND:

copy_iacl_dir path1_1_ path2_1_ {... path1_n_ path2_n_}

FUNCTION: copies the initial access control list for directories (directory initial ACL) of one directory to another, replacing the current directory initial ACL if necessary.

ARGUMENTS:

path1_i_
is the pathname of a directory. The star convention is allowed.

path2_i_
is the pathname of the target directory. The equal convention is allowed.

NOTES: See the MPM Reference Guide for a description of initial ACL's.

SYNTAX AS A COMMAND:

copy_iacl_seg path1_1_ path2_1_ {... path1_n_ path2_n_}

FUNCTION: copies a segment initial access control list (initial ACL) from one directory to another, replacing the current initial ACL if necessary.

ARGUMENTS:

path1_i_
is the directory from which the initial ACL is to be copied. The star convention is allowed.

path2_i_
is the directory into which the initial ACL is to be copied. The equal convention is allowed.

SYNTAX AS A COMMAND:

cr paths

FUNCTION: causes a segment to be created in a specified directory, or in the working directory. That is, it creates a storage system entry for an empty segment.

ARGUMENTS:

paths
are pathnames of segments to be created.

ACCESS REQUIRED: The user must have append access to a directory in order to create a segment in that directory.

NOTES: If the creation of a new segment would introduce a duplication of names within the directory, and if the old segment has only one name, the user is interrogated whether to delete the segment bearing the old instance of the name. If the old segment has multiple names, the conflicting name is removed and a message to that effect is issued to the user. In either case, since the directory is being changed, the user must also have modify permission for the directory.

The user creating the new segment is given rw access to the segment created.

All directories specified in paths must already exist. That is, only a single level of the storage system hierarchy can be created with this command.

If any one of the paths is the name of an existing link, a segment is created in the place specified by that link. The user must have append access to the directory containing the link target in order to create this segment.

See the description of the create_dir and link commands for an explanation of the creation of directories and links, respectively.

create (cr)

EXAMPLES:

The command line:

```
! cr first_class_mail >udd>Demo>Jones>alpha>beta
```

creates the segment first_class_mail in the working directory and the segment beta in the directory >udd>Demo>Jones>alpha. As explained above, the directory alpha must already exist.

create_data_segment (cds)

SYNTAX AS A COMMAND:

cds path {-control_arg}

FUNCTION: translates a create_data_segment (CDS) source program into an object segment. A listing segment is optionally created. These results are placed in the user's working directory. This command cannot be called recursively.

ARGUMENTS:

path
is the pathname of a CDS segment. If path does not have a cds suffix, one is asssumed. However, the cds suffix must be the last component of the name of the source segment.

CONTROL ARGUMENTS:

-list, -ls
produces a source listing of the CDS program used to generate the data segment followed by object segment information (as printed by the print_link_info command described in the MPM Subsystem Writers' Guide) about the actual object segment created.

NOTES: The source for create_data_segment programs is standard PL/I with the restriction that the program include a call to the create_data_segment_ subroutine. The create_data_segment_ subroutine creates a standard object segment from PL/I data structures passed to it as parameters. These data structures can be initialized with arbitrarily complex PL/I statements in the CDS program. (See the MPM Subroutines for a description of the create_data_segment_ subroutine.)

Since the create_data_segment command invokes the PL/I compiler to first compile the CDS segment, any errors that the compiler finds are reported by its standard technique. If any errors with a severity greater than 2 occur, the CDS run is aborted and an object segment is not created.

SYNTAX AS A COMMAND:

cd paths {-control_args}

FUNCTION: causes a specified directory branch to be created in a specified directory, or in the working directory. That is, it creates a storage system entry for an empty subdirectory. See the description of the create command for information on the creation of segments.

ARGUMENTS:

paths
are pathnames of directories to be created.

CONTROL ARGUEMENTS:

-access_class STR, -acc STR
applies to each pathi and causes each directory created to be upgraded to the specified access class. The access class can be specified with either long or short names.

-logical_volume VOL, -lv VOL
specifies that each directory created is to be a master directory whose segments are to reside on the logical volume named VOL.

-quota N
specifies the quota to be given to the directory when it is created. This argument must be specified if either the -access_class or -logical_volume control argument is specified. If omitted, the directory is given zero quota. The value of N must be a positive integer, and applies to each pathi.

ACCESS REQUIRED: The user must have append permission to a directory in order to create a subdirectory in that directory.

NOTES: If a quota is specified and the directory being created is not a master directory, the containing directory must have sufficient quota to move quota to the directory being created. (See the move_quota command for additional information.)

If the creation of a new subdirectory introduces a duplication of names within the directory, and if the old subdirectory has

create_dir (cd)

only one name, the operation is not performed. If the old subdirectory has multiple names, the conflicting name is removed and a message to that effect issued to the user.

The user is given sma access on the created subdirectory.

All superior directories specified in pathi must already exist. That is, only a single level of storage system directory hierarchy can be created in a single invocation of the create_dir command.

In order to create a master directory, the user must have a quota account on the logical volume with sufficient volume quota to create the directory. A master directory must always have a nonzero quota; therefore, the -quota control argument must always be given when creating a master directory. A master directory can be created even though the logical volume is not mounted.

Each upgraded directory must have a quota greater than zero and must have an access class that is greater than its containing directory. The specified access class must also be less than or equal to the maximum access authorization of the process.

When the -access_class control argument is specified, the command does not create a new directory through a link. Creating through links is allowed only when the access class of the containing directory is taken as the default.

EXAMPLES:

The command line:

```
! cd sub >my_dir>alpha>new
```

creates the directory sub immediately inferior to the current working directory and the directory new immediately inferior to the directory >my_dir>alpha. As noted above, the directories my_dir and alpha must already exist. Both directories are assigned the access class of their containing directory.

create_dir (cd)

The command line:

```
! cd subA -access_class a,c1,c2 -quota 5
```

creates the directory subA with an access class of a,c1,c2 and a quota of 5 pages. The directory subA is created immediately inferior to the working directory. (The access class names a, c1, and c2 used in the example represent possible names defined for the site. See the print_auth_names command for more details on access class names.)

The command line:

```
! cd subB -logical_volume volz -quota 100
```

creates a master directory subB immediately inferior to the working directory. Segments created in this new directory will reside on the logical volume named volz. The directory subB is given a quota of 100 records.

cumulative_page_trace (cpt)

SYNTAX AS A COMMAND:

cpt command_line {-control_args}

FUNCTION: accumulates page trace data so that the total set of pages used during the invocation of a command or subsystem can be determined. The command accumulates data from one invocation of itself to the next. Output from the command is in tabular format showing all pages that have been referenced by the user's process. A trace in the format of that produced by the page_trace command can also be obtained.

ARGUMENTS:

command_line
is a character string to be interpreted by the command processor as a command line. If this character string contains blanks, it must be surrounded by quotes. All procedures invoked as a result of processing this command line are metered by the cumulative_page_trace command.

CONTROL ARGUMENTS:

-count, -ct
prints the accumulated results, giving the number of each page and the number of faults for each page. This control argument cannot be used with -print or -total (see "Notes" below).

-flush
clears primary memory before each invocation of the command line and after each interrupt. This helps the user determine the number of page faults but increases the cost.

-interrupt N, -int N
interrupts execution every N virtual CPU milliseconds for page fault sampling. The default is 500 CPU milliseconds.

-long, -lg
produces output in long format, giving full pathnames.

-loop N
calls the command to be metered N times.

-print, -pr
prints the accumulated results, giving the number of each page referenced. This control argument cannot be used with -count or -total (see "Notes" below).

cumulative_page_trace (cpt)

-print_linkage_faults
prints all accumulated linkage faults and calls to the hcs_$make_ptr entry point.

-reset, -rs
resets the table of accumulated data. If the table is not reset, data from the current use of cumulative_page_trace is added to that obtained earlier in the process.

-short, -sh
formats output for a line length of 80.

-sleep N
waits for N seconds after each call to the command being metered.

-timers
includes all faults between signal and restart.

-total, -tt
prints the total number of page faults, the total number of segment faults, and the number of pages referenced for each segment. This control argument cannot be used with -count or -print (see "Notes" below).

-trace_linkage_faults
accumulates linkage faults information along with page and segment fault information.

-trace path
writes the trace on the segment named path using an I/O switch named cpt.out; cumulative_page_trace attaches and detaches this switch.

NOTES: The cumulative_page_trace command operates by sampling and reading the system trace array after invocation of a command and at repeated intervals. Control arguments are given to specify the detailed operation of the cumulative_page_trace command.

The command line used to invoke the cumulative_page_trace command includes the command or subsystem to be traced as well as optional control arguments.

At least one of three generic operations must be requested. They may all be combined and, if so, are performed in the following order: resetting the table of accumulated data,

cumulative_page_trace (cpt)

calling the command to be metered, applying the specified control arguments, and printing the results in the specified format.

The default mode of operation uses interrupts for page fault sampling with a default sampling time of 500 milliseconds. If this figure is too large, messages indicate that some page faults may have been missed; a smaller value can then be chosen. The cost of a smaller value is high and may cause additional side effects. If the command or subsystem to be metered includes the taking of CPUT interrupts, then the -timers control argument should be specified. This control argument causes some of the page faults of the metering mechanism to be included as well.

Only one of the control arguments -print, -count, or -total can be specified. Each of these control arguments produces printed output in a different format. If more than one format is desired, the command must be invoked once for each format.

EXAMPLES:

The command line:

```
! cpt "pl1 test" -interrupt 400 -trace trace_out
```

calls the pl1 command to compile the program named test, requesting an interrupt every 400 milliseconds to obtain page trace information. Trace information is placed in a segment named trace_out.

The command line:

```
! cpt "list -pn >udd>Multics" -loop 2 -sleep 10
```

calls the list command twice, and sleeps for 10 seconds between calls.

The command line:

```
! cpt -print
```

prints the accumulated results of previous metering.

damaged_sw_off (dsf)

SYNTAX AS A COMMAND:

dsf paths

FUNCTION: resets the damaged switch for segments. See "Notes" below, for an explanation of the damaged switch.

ARGUMENTS:

paths
are pathnames of segments. The star convention is allowed.

NOTES: If a device error or system crash destroys a page of a segment, the supervisor turns on the damaged switch associated with the segment. Two cases of damaged segments can occur. Sometimes, the only valid copy of a page of the segment is destroyed; if so, a page of zeros appears in the segment. In other rare cases, a modified page of a segment cannot be written out due to a paging device error; when this occurs, the original unmodified page may be supplied.

An attempt to reference the contents of a segment whose damaged switch is on causes an error with the message:

Entry has been damaged. Please type "help damaged_segments.gi"

When a damaged segment is detected, the owner of the segment should change the access of the segment so that no other user can reference it, and then reset the damaged switch with the damaged_sw_off command. The owner should then inspect the contents of the segment to determine whether the segment can be re-created or needs to be retrieved.

Users can explicitly turn on the damaged switch of a segment. This is generally done to test recovery procedures. See the description of the damaged_sw_on command.

damaged_sw_on (dsn)

SYNTAX AS A COMMAND:

dsn paths

FUNCTION: sets the damaged segment switch on for a segment. The damaged switch is off for newly created segments. The switch can be turned on by the system (see "Notes" in the damaged_sw_off command for an explanation) or by a user conducting special tests.

ARGUMENTS:

paths
are pathnames of segments. The star convention is allowed.

NOTES: The damaged_sw_on command is primarily useful for testing recovery procedures. It should not be used instead of normal access control procedures for casual denial of normal access.

SYNTAX AS A COMMAND:

date {dt}

SYNTAX AS AN ACTIVE FUNCTION:

[date {dt}]

FUNCTION: returns the date abbreviation for a specified date or the current date.

ARGUMENTS:

dt
is a date-time in a form acceptable to convert_date_to_binary_. If no argument is specified, the current date is returned.

EXAMPLES:

The command line:

! date May 5, 1980

prints:

05/05/80

The command line:

! date Monday

prints the next occurrence of a Monday.

date_compiled (dtc)

SYNTAX AS A COMMAND:

dtc path {-control_arg} {components}

SYNTAX AS AN ACTIVE FUNCTION:

[dtc path {-control_arg} {components}]

FUNCTION: prints the date and time compiled and the compiler identifier for an object segment or an archive of object segments. For a bound object segment, the command prints the date and time compiled for each component.

ARGUMENTS:

path
is the pathname of an object segment, bound object segment, or an archive of object segments.

components
are names of components in a bound object segment or archive of object segments. If component names are specified, information on only these components is listed.

CONTROL ARGUMENTS:

-brief, -bf
lists the date and time compiled only (see "Examples" below).

-long, -lg
lists the date and time compiled, the segment name, the User_id of the person who compiled the segment, and the long form of the compiler identifier (see "Examples" below).

NOTES: If an archive is listed, the bind file is ignored.

If neither control argument is specified, the command lists the date and time compiled, the segment name, the User_id, and the short compiler identifier (see "Examples" below).

Invoked as an active function, dtc returns the first line of output that would be printed if it were invoked as a command.

date_compiled (dtc)

EXAMPLES: To check the compilation date of a private version of the list command in the working directory, type:

```
! date_compiled list -bf
  04/11/77 0922.2
```

To check information on the latest compilation of a Multics system installed command (that is unbound), such as demo_command, type:

```
! date_compiled >system_library_standard>demo_command
  03/09/77 1615.2 demo_command Martinson.SysMaint.a PL/I
```

To get compilation information on an entire bound object segment that is part of the standard Multics system, type:

```
! date_compiled >sss>bound_binder_
  Bound  10/26/77  1337.4  bound_binder_ Martinson.SysMaint.a
binder
  07/26/76 1048.4 bind Martinson.SysMaint.a PL/I
  10/26/77 1328.2 bx_ Martinson.SysMaint.a cds
                        .
                        .
                        .
  12/27/76 1354.3 old_make_bindmap_ Martinson.SysMaint.a PL/I
```

To get detailed information on one component of a bound object segment (in this case, bind in bound_binder_), type:

```
! date_compiled >sss>bound_binder_ -lg bind
  07/26/76  1048.4 bind Martinson.SysMaint.a Multics PL/I
    Compiler, Release 20e, of May 22, 1976
```

SYNTAX AS A COMMAND:

date_time {dt}

SYNTAX AS AN ACTIVE FUNCTION:

[date_time {dt}]

FUNCTION: returns a date and time value for a specified date-time or the current date-time consisting of: a date, a time from 0000.0 to 2359.9, a time zone, and a day of the week. The date and time value is returned as a single, quoted string of the form "mm/dd/yy hhmm.m zzz www" (e.g., "08/17/76 0945.7 est Tue").

ARGUMENTS:

dt
is a date-time in a form acceptable to conver_date_to_binary_. If no argument is specified, the current date-time is returned.

SYNTAX AS A COMMAND:

day {dt}

SYNTAX AS AN ACTIVE FUNCTION:

[day {dt}]

FUNCTION: returns a one- or two-digit number of a day of the month, from 1 to 31.

ARGUMENTS:

dt
is a date-time in a form acceptable to convert_date_to_binary_. If no argument is specified, the current day of the month is returned.

EXAMPLES:

The command line:

! day Friday

prints the one- or two-digit number of the next occurrence of Friday.

SYNTAX AS A COMMAND:

day_name {dt}

SYNTAX AS AN ACTIVE FUNCTION:

[day_name {dt}]

FUNCTION: returns the full name of a day of the week for a specified date or the current date.

ARGUMENTS:

dt
is a date_time in a form acceptable to convert_date_to_binary_. If no argument is specified, the name of the current day is returned.

EXAMPLES:

The command line:

```
! day_name 10/19/79
```

prints:

```
Friday
```

SYNTAX AS A COMMAND:

debug

FUNCTION: is an interactive debugging aid to be used in the Multics environment. It allows the user to look at or modify data or code. The user can stop execution of a program and examine its state by inserting "breakpoints" in the program before and/or during execution. A concise syntax for user requests, coupled with a complete system of defaults for unspecified items, allows the user to make many inquiries with little effort. Symbolic references permit the user to retreat from the machine-oriented debugging techniques of conventional systems and to refer to variables of interest directly by name.

The debug command uses a segment in the home directory to keep track of information about breaks. This segment is named Person_id.breaks, where Person_id is the login name of the user. The break segment is created if not found. If the segment cannot be created, the break features of debug are disabled and unusable.

Users who do not need the sophisticated machine level debugging provided by this command should refer to the probe command in this manual.

With the debug command, the user can:

Look at data or code;

Modify data or code;

Set a break;

Perform (possibly nonlocal) transfers;

Call procedures;

Trace the stack being used;

Look at procedure arguments;

Control and coordinate breaks;

Continue execution after a break fault;

Change the stack reference frame;

Print machine registers; and

Execute commands.

These functions are provided by two types of debug requests: data requests and control requests. The first five functions above are performed by data requests; the others, by control requests. Multiple debug requests (either data or control) can be placed on a line separated by semicolons (;).

Number Representation Conventions

Debug uses both octal and decimal representation of numbers. In general, machine-dependent numbers such as pointers, offsets, and registers are assumed to be octal, while counting arguments (e.g., specifying a source line number, printing the first 20 lines) and variables referenced by name are assumed to be decimal.

A decimal default can be changed to octal by preceding the number with the escape sequence "&o". An octal default can be changed to decimal by preceding the number with "&d".

Example	Description
x = 8	assign the value 8 to the program variable x. Program variables referenced by name are assumed to be decimal; if octal representation is preferred, type: x = &o10
$q = 77	assign the value of 77 to the q-register. Register values are machine dependent and assumed to be octal; if decimal representation is preferred, type: $q = &d63
/test/&a19	print line 19 of the source segment for test.
&a19,s8	print 8 source lines, beginning at line 19.

Data Requests

Data requests consist of three fields and have the following format:

<generalized address> <operator> <operands>

The generalized address defines the actual data or code of interest. It is ultimately reduced to segment number and offset by debug before being used. The operator field indicates to debug which function to perform, e.g., print or modify the data referenced by the generalized address. The operands field may or may not be necessary, depending on the operator. When these fields are specified, they are separated by blanks or commas.

When debug decodes a data request, it parses the generalized address and generates a pointer to the data being referenced. This pointer, called the working pointer, is changed whenever the generalized address is changed. It points into either the working segment, its stack frame, or its linkage section. The actual segment depends on the most recent specification in a generalized address. The form for a generalized address is as follows:

/<segment name>/<offset><segment ID><relative offset>

where each of the four fields is optional. The segment name is either a pathname, a reference name, or a segment number, and defines what is called the working segment. The segment ID specifies which of the data bases associated with the working segment is to be used in setting the working pointer. The segment ID can be one of the following:

&s
refers to the stack frame if the working segment is a procedure segment with an active stack frame.

&l
refers to an active linkage section (i.e., one with an entry in the linkage offset table (LOT) for the user's ring).

&t
refers to the working segment itself.

&a
refers to the source program for the working segment.

&p
refers to the parameters of an active invocation of a procedure.

&i
refers to an active internal static section (i.e., one with an entry in the internal static offset table (ISOT) for the user's ring).

The offset field is used as an offset within the segment referenced by the working pointer. For the working segment, this offset is relative to the base of the segment. If the working pointer points into an active stack frame, the offset is relative to the base of that frame. If the working pointer points into an active linkage section, the offset is relative to the beginning of that linkage section.

The offset can be either a number or a symbolic name. If a symbolic name is specified, a symbol table must exist for the working segment. See the translator commands for descriptions of symbol table creation. If a symbolic name begins with a numeric character, the escape characters &n (for name) must precede the name, to avoid interpreting the name as a number. For example:

```
/test/&n10&t
```

can be used in a debug request to specify the location associated with FORTRAN line number (i.e., label) 10.

The relative offset field allows the user to relocate the working pointer by a constant value or register. For example, a user wishing to reference the fourth word after the stack variable i he could use:

```
/test/i+4
```

as the generalized address. The relative offset can also assume the value of a register. For example, if the a-register contains the value 4 at the time of a break, then:

```
/test/100&s$a
```

sets the working pointer to offset 104 from the base of the stack frame. It is important to note that a + sign is not present when a register is used. (See "Registers" below.)

The three most common values for the segment ID field are &t, &s, and &l. These designate that the working pointer is to refer to, respectively, the working segment itself, its active stack frame, or its active linkage section. In addition, two other possible values of segment ID allow alternate methods of referring to locations in either the working segment or its stack frame.

A segment ID of &a refers to the ASCII source program for the working segment. Associated with this segment ID is a decimal line number, which must immediately follow the &a. This line number is used to generate a working pointer to the first word of code compiled for that line. A relative offset can follow the line number. Note that the line-number/code-location association can only be determined if a symbol table exists for the working segment. This example:

```
/test_seg/&a219+36
```

generates a working pointer that points to the thirty-sixth (octal) word in the text after the first word of code generated for line 219 in the source for the segment test_seg. If an offset field is given before &a, the offset is ignored. The offset of the working pointer is generated solely from the line number and the relative offset.

A segment ID of &p refers to the parameters of an active invocation of a procedure. If the current defaults specify an active stack frame, a number following the &p specifies the parameter that is to be addressed. The offset field is ignored, but a relative offset can be specified. This example:

```
/test_seg/&s;&p4+36,a14
```

causes the stack frame for test_seg to be the working segment, and the first 14 characters of the data contained at a location 36 words after the beginning of the fourth parameter are printed in ASCII format.

It is not necessary to specify all four fields of a generalized address. In fact, every field is optional. If a field is not specified, a default value is assumed that is frequently the last value that the field had. For example:

```
/test_seg/line&s+3
```

followed by the generalized address:

```
+4
```

is acceptable. The latter request is equivalent to:

```
/test_seg/line&s+7
```

One time that the defaults assumed are not the values of the previous data request is when a symbolic variable name or label is specified that causes some field to change. If this is the case, debug might recognize that the segment ID, for example, of the previous data request is not valid and set it appropriately. For example:

```
/test_seg/760&s
```

followed by:

```
regp
```

would cause the defaults to be changed to:

```
/test_seg/140&l
```

if regp is found at a relative offset of 140 (octal) in the linkage section. Note that the segment ID is changed to &l where it remains until explicitly or implicitly changed again.

Defaults are also reset to values different from the previous values when the segment name field is specified in a generalized address. In this case, the following actions are taken:

1) If the segment name begins with &n, take the rest of the characters composing the segment name and go to step 3 below, treating the string as a name. This convention allows the use of debug on segments whose names are composed of numeric characters.

2) If the segment name is really a segment number, this number is used in a search of all active stack frames to see if one exists for this segment. The search is from the highest stack depth (deepest in recursion) to the base of the stack so that if an active stack frame is found, it is the one most recently used. If an active stack frame is found, the generalized address defaults are set as follows:

a. working segment	the one specified by the given segment number.
b. offset	zero.
c. segment ID	&s, i.e., the working pointer points into the latest stack frame for the working segment.
d. relative offset	zero.

If no active stack frame is found, the defaults are set as above except that the segment ID is &t instead of &s, i.e., the working pointer points into the working segment itself.

3) If the segment name is a reference name known in this ring, the segment number for the segment being referenced is found, and then the defaults are calculated as if this segment number were given directly.

4) If the segment name is a pathname, the specified segment is initiated (it can already have been known) and the returned segment number is used as above.

5) If the segment name is of the form segname$entname, the stack is searched from the highest active frame (as in step 2) for the most recent frame associated with the entry point entname in the segment segname. The working segment becomes segname, and the remaining defaults are set as described in step 2.

The entire set of defaults that apply to a debug data request can be determined at any time by issuing the .d control request to print defaults. For the format and use of this request, see the description under "Control Requests" below.

Operator Field of Data Requests

After decoding the generalized address and determining the working pointer, debug checks the operator. The following

five operators are recognized:

1. , (comma) print
2. = assign
3. < set a break
4. > alter program control (i.e., "go to")
5. := call a procedure

If a debug request is terminated before an operator is encountered either by a semicolon or a newline character, the default operator used is ",", i.e., print. The one exception is that a blank line is ignored. The first, second, and fifth operators above have operands.

Print Request

For the print request, there are three optional operands. They are a single character specifying the output mode desired; a number indicating how much output is being requested; and a number in parentheses indicating the size of the output. The size has two meanings that are dependent on the output mode being used.

1) If the mode is comp-8 or comp-5, the size is the number of digits plus the sign, if present.

2) If the mode is not comp-8 or comp-5, the size is the number of bits to use in printing one item.

The size specification is permitted for the following modes: o, h, d, e, f, p, comp-5, comp-8. It is ignored for the following modes: i, l, a, b, comp-6, comp-7. All of the arguments are optional and spaces can appear between arguments. For example:

```
142&s,o(18)12
```

requests that 12 (decimal) half words starting at 142 (octal) in the stack be printed in octal format.

The following output modes are available for print requests (see "Output Modes" below for a full description):

o	octal
h	half-carriage octal
d	decimal
a	ASCII
i	instruction
p	pointer
s	source statement
l	code for line number
n	no output (just change defaults)
e	floating point with exponent
el	long floating point with exponent
f	floating point
fl	long floating point
b	bit string
g	graphic
comp-5	COBOL
comp-6	COBOL
comp-7	COBOL
comp-8	COBOL

The request:

```
+36,a14
```

requests that 14 (decimal) characters starting at 36 (octal) words after the current working pointer be printed in ASCII format. The output might be:

```
1416   1416  ">user_dir_dir>"
```

The two numbers printed in most output modes should be interpreted as follows:

1) If the data is from a stack frame, the first number is the relative offset from the base of the stack segment and the second number is the relative offset within the stack frame. If the second number is negative, the variable does not exist in the current stack frame and is a parameter or a global variable.

2) If the data is from a linkage section, the first number is the offset within the combined linkage segment and the second number is the offset within the linkage section.

3) For all other segments, both numbers are the same and represent the offset within the segment.

If a mode is not specified for output, the last specified mode is used unless debug realizes another mode is more appropriate (e.g., when a symbol specifies a variable of a different type). If the amount of output is not specified, it is assumed to be one unit, i.e., one word for octal output, one line for source output, one character for ASCII output, etc.

Assign Request

When modifying data or code, the operands (at least one is expected) specify the new values to use. For example:

```
i = 8;  p(1) = 206|10, 206|32
```

assigns the decimal value 8 to i and the values 206|10 and 206|32 to p(1) and p(2), respectively. (It is assumed that both are variables that are defined for the current working segment.) If more than one operand is specified in an assignment request, consecutive words starting at the working pointer are changed. This is illustrated by the assignment to the pointer array p.

There are nine acceptable forms for assignment operands:

1. octal number
2. decimal number
3. character string
4. register value (see "Registers" below)
5. instruction format input
6. floating point number
7. pointer
8. bit string
9. variable

Whether a number is assumed to be octal or decimal on input depends on the target. A variable referenced by name is assumed to be decimal unless overridden by "&o". Assignment to a location specified by offset is assumed to take an octal value unless overridden by "&d".

```
x = 99 (decimal)
+2 = 77 (octal)
```

Character strings being input must be bracketed by quote characters ("). Bit strings being input must be bracketed by quote characters and followed by a b. Floating point numbers must not have exponents.

The word-offset portion of a pointer value being input can optionally be followed by either a decimal bit offset in parentheses, a ring number in square brackets, or both. If both a bit offset and a ring number are specified, the ring number must follow the bit offset, with no intervening blanks. For example:

```
p = 206|25(29); q = 252|104[5]; rp = 211|200(3)[4]
```

The format for instruction input is:

(opcode address,tag)

The address can specify a base register or a number. For example:

/test/lab2 = (lda pr6|20) (sta pr0|2,*0) (nop 0)

Some value must be given for the address field. The zero opcode is specified by the opcode arg.

Input of bit strings and character strings changes only those bits or characters specified, i.e., a full word might not be completely changed.

Several types of input can be interspersed in the same assignment request. For example:

/145/13000 = "names" &d16 126

When different types of input are specified in one request, the user should be aware that the bit offset of the temporary working pointer might be ignored for certain types of input. In the example above, the ASCII for "name" is placed at 145|13000 and the ASCII for "s" is placed in the first character position of 145|13001. The next assignment argument (&d16) fills in 145|13001 with the decimal 16 and hence overwrites the "s" of the previous argument.

In order to better specify more complicated assignments, a repetition factor is provided. If a single number (decimal) appears in parentheses in an assignment, the next data item is assigned repeatedly (i.e., the specified number of times), updating the working pointer each time. An example of this is:

string = (32)" " "alpha"

which results in string being modified so that the first 32 (decimal) characters are blanks, and the 33rd through the 37th contain the string "alpha".

Set Break Request

A breakpoint is a special modification to the code of a program that, when executed, causes control to pass to debug. The user is then free to examine and change the states of variables, set other breaks, continue execution, etc. When setting a break, the working pointer is used directly unless it points into the stack. In that case, the working pointer is temporarily forced to the text. To set a break at the label loop_here in the program parse_words, the user types:

```
/parse_words/loop_here<
```

One can also type:

```
/parse_words/loop_here+23<
```

to set the breakpoint 23 (octal) locations after the first word of code for the statement labelled loop_here in the text segment.

One can also set a break by specifying a line number. For example:

```
/rand/&a26<
```

sets a break at the first word of code generated for line 26 (decimal) of the source program.

The break number printed by debug when setting a breakpoint is used as the name of the break when referring to breaks. After a break is reset, the break number is reused. (Resetting a break restores the code to its previous value.)

Once a break has been set at a given location, another break cannot be set there. The list breaks control requests .bl and .bgl can be used to find out which breaks are set.

Alter Program Control Request

To alter program control by issuing an explicit transfer, the user can type:

```
/216/2176>
```

causing debug to search the stack for an active stack frame for the segment 216 (octal) and set the stack pointer to this frame. It then transfers to 2176 (octal) in the text associated with this stack frame.

If no active stack frame is found, debug prints a message and waits for further requests.

Call a Procedure Request

The user can cause debug to call a specified procedure and return values into specified locations. This is done by specifying := as the operator in a data request. This operator expects one operand that is a procedure name with its associated arguments. There are two slightly different ways to invoke this feature: first, to invoke a procedure as a function call (with the argument n+1 being the returned value); and second, to explicitly call a procedure. When a procedure is invoked as a function reference, the current working pointer is used as the last argument in the argument list and, hence, the procedure returns a value into wherever the working pointer is pointing. For example:

```
/test/fi := sqrt_(2.0)
```

causes the sqrt_ function to be called with the first argument 2.0 and the return argument of fi; debug converts the 2.0 into a floating point number before the call.

If no fields are present before the := is encountered, debug does not specify a return argument in the call. (The := can be thought of as "call" in a PL/I program.) For example:

```
:= who
```

sets up a call to who$who with no arguments. The call:

```
:= rename ("foo","moo")
```

and:

```
..rename foo moo
```

are functionally equivalent. (See Multics command execution under "Control Requests" below.)

The method debug uses in setting up the call is to use ten temporary storage areas, one for each of ten possible arguments. debug converts the arguments appropriately and stores the values in these areas. Each area starts on an even location and consists of eight words. These temporary storage areas can be looked at or altered with standard data requests. They are named %1, ..., %10. For example:

```
:= cpu_time_and_paging_(0,0,0)
%1,d
%2,d
%3,d
```

prints three decimal numbers, all being return values from hcs_$usage_values. The actual call that debug made had three arguments that were all 0. (The first words of the first three storage areas were zeroed out prior to the call.) The above call can also be made as follows:

```
%3 := cpu_time_and_paging_(0,0)
```

If this is done, the third argument is not zeroed before the call.

Variables can also be used as arguments. For example:

```
sum := sqrt_(n)
```

No conversion is done by debug if n is fixed and sqrt_ expects a floating argument.

The above mentioned temporaries can be used to do simple mode conversion. For example, to get the floating point representation of 3.7 (in octal) the user can type:

```
%1 = 3.7; ,o
```

To find the ASCII value for 137 (octal) the user can type:

```
%1 = 137137137137 ; ,a4
```

A reference to one of these storage areas causes the working segment to be changed to the stack segment.

If one of the arguments in a procedure call is the character %, the temporary storage for that argument is not changed (e.g., overwritten with the usual argument value). Results from some previous work can be passed in that argument position. For example:

```
%2 := sqrt_(2.0)
:= ioa_("^e",%)
```

Registers

The hardware registers at the time of a fault (in particular a break fault) are available to the user for inspection or change. These registers are referenced by preceding the register name immediately by a dollar sign ($). The register can be looked at by merely typing the register name. For example:

```
$a
```

prints the contents of the a-register at the time of the last fault. The value in the a-register can be changed to octal 146 by typing:

```
$a = 146
```

Decimal input is allowed also:

```
$a = &d19
```

The predefined register names used by debug are:

pr0	pointer register 0
pr1	pointer register 1
pr2	pointer register 2
pr3	pointer register 3
pr4	pointer register 4
pr5	pointer register 5
pr6	pointer register 6

pr7	pointer register 7
prs	all pointer registers
x0	index register 0
x1	index register 1
x2	index register 2
x3	index register 3
x4	index register 4
x5	index register 5
x6	index register 6
x7	index register 7
a	a-register
q	q-register
aq	the a- and q-registers considered as a single register
exp	exponent register
tr	timer register
ralr	ring alarm register
eaq	the exponent, a- and q-registers in floating point format
regs	all the above from x0 through ralr
ppr	procedure pointer register
tpr	temporary pointer register
even	even instruction of Store Control Unit (SCU) data
odd	odd instruction of SCU data
ind	the indicator register
scu	all SCU data
all	all machine conditions

The user can change the above registers at will (with the exception of "ind" and "eaq") with the understanding that if execution continues after the break or transfers directly (via > in a data request), the values of the hardware registers are set to those of the above registers.

The values in the registers are automatically filled in by debug (when it is called or faulted into) with those values associated with the last fault found in the stack. The user can override these values with the fill registers (.f) and crawlout registers (.C) control requests. See "Control Requests" below.

The user can also define registers and use them as a small symbolic memory. For example:

```
$sta1 = 600220757100;    $nop = 11003
```

allows the user to later specify:

```
/test/210&t = $sta1 $nop $nop
```

To print out the contents of all user-defined registers, the user can type:

```
$user
```

The setting and displaying of registers follows the syntax of data requests. However, only the register name and a possible new value can appear in a register request. Registers can be specified in a general data request only in the relative offset field and as operands in assignment requests. Register names must be less than or equal to four characters in length. Some examples of the use of registers follow:

```
/test/i =$q
/test/0 = $x0
/test/46$x0,a5
```

Control Requests

Control requests provide the user with useful functions not necessarily related to any specific data. The format for a control request is:

.<request name>

Control requests and data requests can be freely mixed on a command line if separated by semicolons. However, certain control requests use the entire input line and hence ignore any semicolons found therein. Spaces are not allowed in most control requests.

The following is a list of all control requests and the functions they perform. See "Summary of Data and Control Requests" below for a complete review of all requests.

TRACE STACK

The general form is:

.t*i*,*j*

The stack is traced from frame *i* (counting from 0 at the base of the stack) for *j* frames, where *i* and *j* are decimal integers. If *i* is less than 0, tracing begins at 0; if *i* is greater than the last valid frame, then only the last frame is traced. If *i* is not specified, it is assumed to be 0; if *j* is not specified, all valid stack frames from *i* on are traced. The name printed in the stack trace is the primary segment name unless the segment is a PL/I or FORTRAN program in which case it is the entryname invoked for the stack frame (i.e., the label on the entry or procedure statement).

Examples:

```
.t2,3
.t100
```

POP OR PUSH STACK

The general form is:

.+*i* or .-*i*

debug (db)

The working segment is changed by moving up or down the stack i frames, where i is a decimal integer. For example, if the working segment's active stack frame is at depth 4 in the stack, then:

```
.+3
```

changes the working segment to the segment whose stack frame is at depth 7 in the stack. The defaults for working pointer, segment ID, and offset are reinitialized to the base of the stack frame, &s, and 0, respectively.

SET STACK

The general form is:

```
.i
```

The working segment is set to that of stack frame i (starting at 0), where i is a decimal integer. The defaults are set as in pushing or popping the stack.

EXECUTE MULTICS COMMAND

The general form is:

```
..<Multics command line>
```

The rest of the input line after the .. is interpreted as a standard Multics command line and is passed to the standard command processor with any preceding characters blanked out. Any valid Multics command line can be given. When setting breaks, the program being debugged must be called in this manner because debug sets up a condition handler (for break faults) that is active only as long as debug's stack frame is active.

PRINT DEFAULTS

The general form is:

.d or .D

The output might look like:

```
.vm 6,3,2,20
3 /test_seg/14(0)&t,i  212
```

or:

```
3 />udd>m>foo>test_seg/14(0)&t,i  212
```

The first number (3 above) is the stack frame depth in decimal, unless there is no stack frame for the working segment, in which case the number is -1. The name of the working segment appears between the slashes (test_seg above); if .D is used, the full pathname occurs here. The offset appears next (14 above); the bit offset (in decimal) of the working pointer appears next; the segment ID (&t above) appears next; the operator appears next (, for print); the output mode appears next (i for instruction); finally the segment number of the working segment appears (212 above). To find the name/segment number association for a given segment, for example segment number 206, the user can type:

```
/206/,n;.d
```

yielding:

```
60 /test_caller/0(0)&s,o  206
```

Knowing the name, the user can obtain the same output by typing:

```
/test_caller/,n;.d
```

CONTINUE EXECUTION AFTER A BREAK

The general form is:

.c,i

or:

debug (db)

.ct,i

or:

.cr,i

If i is not specified, it is assumed to be 0. If i is specified, the next i break faults for the current break are skipped. The first instruction executed upon continuation is the instruction on which the break occurred. If a t follows the c, debug continues in temporary break mode (see "Break Requests" below). If an r follows the c, debug resets the mode to normal (not temporary).

Examples:

.c	continue execution.
.c,3	continue execution, but skip the next three break faults for the current break.
.ct	continue execution in temporary break mode.

QUIT

The general form is:

.q

This request returns from debug to its caller. Note that if debug was entered via a break, typing .q returns to the last procedure that explicitly called debug.

CHANGE OUTPUT MODE

Requests pertaining to debug's terminal output begin with ".m".

1) Enter brief output mode:

.mb

This request places debug in brief output mode, which is somewhat less verbose than its normal output mode. In particular, assignment requests and the resetting of breaks are not acknowledged on the user's terminal; the column headings are not printed for a stack trace; the printing of register contents is somewhat more compact; some error messages are abbreviated.

2) Enter long output mode:

```
.ml
```

This returns debug to long output mode, which results in fuller and more explicit terminal output. Long mode is the initial default.

SET I/O SWITCH NAMES

These requests allow a user to debug a program that is run with file output because it generates extensive output or a program that is run from within an exec_com after "&attach" because it requires much input. The general form is:

```
.si switch_name
.so switch_name
```

where switch_name identifies the switch_name to use for input (.si) or output (.so). The named switch must be attached by the user before the request is made. If no switch name is given, debug creates one (either debug_input or debug_output).

1) User makes a switch request but does not give a switch name:

```
.si
.so
```

debug creates a switch named debug_input or debug_output and attaches it to the user_i/o switch. This is the usual request for debugging programs that require the user_input or user_output switches to be attached to a file instead of to user_i/o. Debug detaches the debug_input and debug_output switches when the user quits debug.

2) User makes a switch request and gives the switch name:

```
.si input_switch
.so output_switch
```

The user must attach the switch_name before making the request. This can be used when the user wants to read debug requests from a file. The switches can be restored by typing:

```
.si user_input
.so user_output
```

Examples:

The user has directed the output switch named user_output to a segment, but wants debug diagnostics to be printed on the terminal. This can be done by typing:

```
debug
.so
```

Since a switch name is not given with the request, debug sets up a new I/O switch named debug_output as a synonym for user_i/o, which is the terminal in this case. When the user quits debug, the switch named debug_output is detached.

The user wants to debug a procedure that uses the user_input switch and has a set of debug requests in another segment named debug_macro. An input switch named macro has been attached to the segment of debug requests. The user types:

```
debug
.si macro
```

and debug takes requests from the switch named macro and does not detach the switch when the user exits debug. An attempt by debug to read beyond the end of the macro input stream results in an exit from debug.

BREAK REQUESTS

The following control requests are specific to breaks and begin with ".b". Reference is made to the default object segment, which is merely that segment that debug is currently working with when performing break requests. The default

object segment is generally specified implicitly when a break is set or hit. It can be changed and determined upon request. The default object segment used for break requests is not necessarily the same as the segment addressed by the working pointer used in data requests.

Breaks are numbered (named) sequentially starting at 1 but the numbers are unique only for the object segment in which the break resides. A user can have several breaks with the same number defined in different object segments.

There are two types of global requests that can be performed on breaks. The first, or subglobal requests, refer to all breaks within the default object segment. The second, or global requests, refer to all breaks set by the user (as determined from the break segment in the home directory). The subglobal request is specified by omitting the break number in a break request. The global request is specified by a "g" immediately after the "b" of all break requests (see below).

The general form of all break requests is:

.bgx*i* args

where the "g", the number *i*, and the arguments are optional. The "x" is replaced by the control character for the break request desired. The following break requests are currently defined:

1) Reset a break (or breaks). The forms of the requests are:

.br*i* to reset break *i* of the default object segment.
.br to reset all breaks of the default object segment.
.bgr to reset all breaks known to debug.

2) List (print information about) a break. The forms of the request are:

.bl*i* to list break *i* of the default object segment.
.bl to list all breaks of the default object segment.
.bgl to list all breaks known to debug.

3) Execute a debug request at break time. The forms for this request are:

.bei <rest of line>
.be <rest of line>
.bge <rest of line>

Specifying the above request causes <rest of line> to be interpreted as a debug input line whenever the appropriate break(s) is encountered. If <rest of line> is null, the specified breaks have this execute feature reset to normal.

4) Disable a break (or breaks). The forms of this request are:

.boi disable (turn off) break i of the default break segment.
.bo disable all breaks in the default break segment.
.bgo disable all breaks known to debug.

Disabling a break has the effect of preventing the break from being taken without discarding the information associated with it. A user can disable a break rather than reset it if the break is to be needed again in the future. A disabled break can be eliminated altogether (reset) by the .br request, or reenabled by the .bn request. If the break has already been disabled, these requests has no effect.

5) Enable a break or breaks. The forms of this request are:

.bni enable (turn on) break i of the default break segment.
.bn enable all breaks in the default break segment.
.bgn enable all breaks.

This request restores a previously disabled break. If the break was not disabled, the request has no effect.

6) Establish a temporary command line to be executed whenever breaks are encountered. This request is of the form:

.bgt <rest of line>

This causes <rest of line> to be executed as a debug request whenever any break is encountered during the current process. The difference between this request and .bge is that when .bge is typed, the associated line remains associated with all breaks until they are reset, or until they are changed by .be requests. It is possible to have a temporary global command without removing request lines associated with individual breaks. If <rest of line>

is null, a previously-established temporary command line is disestablished.

7) Break conditionally. The following requests allow the user to change a break into a conditional break, i.e., a break that stops only if a certain condition is met.

```
.bci  arg1  <rel>  arg2
.bc   arg1  <rel>  arg2
```

arg1 and arg2 can be constants or variables; <rel> can be = or ^=. Whenever a specified break is encountered, a test is made to see if the equality exists and breaks according to whether the user specified = or ^= in setting up the conditional break. For example:

```
.bc3   i ^= 0
```

causes break 3 to fault whenever it is encountered and the value of i is nonzero. Another example:

```
.bc3   i = j
```

causes break 3 to fault whenever it is encountered and the value of i is the same as the value of j. The comparison is a bit by bit comparison with the number of bits to compare being determined by the size and type of the second argument.

If no arguments are given to a set conditional request, the specified break is set back to a normal break. For example:

```
.bc
```

causes all breaks of the default object segment to fault normally.

8) Specify the number of times a break should be ignored (skipped). The general form is:

```
.bsi  n
```

This causes the number of skips to be assigned to break i of the default object segment to be n.

9) Print or change the default object segment. The form for this request is:

```
.bd  name
```

debug (db)

where name is the (relative) pathname or segment number of the segment to become the default object segment. If name is not specified, the pathname of the default object segment is printed.

10) List the current segments that have breaks. The form for this request is:

.bp

This request merely interprets the break segment in the initial working directory.

PRINT ARGUMENTS

The general form is:

.a*i*,*m*

Argument *i* for the current stack frame is printed in the mode specified by *m*. If *i* is not specified, all arguments are printed. If *m* is not specified, debug decides the output mode. Valid values for *m* are:

o	full word octal
p	pointer
d	decimal
a	ASCII
b	bit string
l	location of argument
e,f	floating point
?	debug decides (the default value for *m*)

Examples:

```
! .a3
  ARG 3:  ">user_dir_dir"
```

```
! .a3,o
  ARG 3: 076165163145
```

GET FAULT REGISTERS

The general form is:

.f

For register requests debug uses the machine registers of the last fault found in the stack starting at the frame currently being looked at. (This is the default when debug is entered as a result of a break fault.)

CRAWLOUT REGISTERS

The general form is:

.C

For register requests debug uses the fault data associated with the last crawlout (abnormal exit from an inner ring).

Program Interrupt Feature

The user can interrupt debug by pressing the quit button at any time, in particular during unwanted output. To return to debug request level (i.e., to where debug waits for a new request), the user should type:

pi

which is the standard program interrupt manager. (See the description of the program_interrupt command.)

Temporary Break Mode

When debug is in temporary break mode (placed there via a .ct control request), the following actions are taken automatically:

1) When the user continues any break, another (temporary) break is set at the first word of code for the next line of source code after the source statement containing the break being continued. If debug cannot determine the location of the next line of source code, the temporary break is set at the word of object code immediately following the break being continued.

2) A temporary break is restored automatically whenever it is continued. A temporary break must be explicitly reset by the user only when it is not continued.

Since temporary breaks are set sequentially in a program (i.e., at the next statement in the source program), any transfers within a program can either skip a temporary break or cause code to be executed that was stopped earlier with a temporary break. Temporary break mode is designed to be used in programs that are fairly uniform and sequential in their flow of control. A user should list breaks after using temporary break mode to see if any breaks remain active.

Indirection

It is quite often desirable to reference the data pointed to by the pointer that is pointed to by the working pointer, i.e., to go indirect through the pointer. The user can instruct debug to do this by typing * instead of the segment name, offset, and segment ID in a generalized address. For example:

```
/test/regp
```

might print:

```
1260  110 214|2360
```

To find what two octal words begin at 214|2360, the user need type only:

```
*,o2
```

This causes the working pointer to be set to 214|2360 and not necessarily point into the same segment as before the request.

Implementation of Breakpoints

Breakpoints are implemented by using a special instruction (mme2) that causes a hardware fault whenever it is executed. When the fault is first encountered in a process using the standard process overseer, a static handler for the fault is set up which passes control to debug. When debug is entered via a break, it does the following:

1) fills the registers with those of the break fault;

2) prints the location of the break fault;

3) waits for requests.

When continuing after a break fault, debug changes the control unit information so that when it is restarted, it executes the instruction that used to exist where the break word was placed.

The debug command keeps track of a default object segment. All break requests made are relative to the default object segment. For example, any reference to break 3 really means break 3 of the default object segment. To change (or find out) the value of the default object segment, the .bd request should be used.

Variable Names for PL/I and FORTRAN Programs

If a symbol table was created for a PL/I or FORTRAN program using the table option, then names of labels, scalars, structures, and arrays can be used. The only restrictions are:

1) that the entire structure name must be specified;

2) the only expressions that are allowed for subscripts are of the form:

   variable $\pm$ constant

   where variable can be an arbitrary reference as above;

3) all subscripts must appear last. If a variable is based on a particular pointer, that pointer need not be specified. Some examples of valid variable references are:

```
p-> a.b.c(j,3)
a.b
p(3,i+2) -> qp.a.b(x(x(4)+1))->j.a
```

Bit Addressing

When a working pointer is generated to a data item that is based on or is a part of a substructure, a bit offset may be required. This bit offset is indeed kept and used. When making references to data relative to a working pointer with a bit offset, the relocated addresses can still contain a bit offset. For example, if the working pointer has the value:

```
151|3706(13)
```

then the request:

```
+16,b3
```

sets the working pointer to:

```
151|3724(13)
```

and prints the three bits at this location.

Output Modes

The following output modes are acceptable to debug:

o octal
The data pointed to by the working pointer is printed in full word octal format, eight words per line.

h half carriage octal
The data is printed as in o format except that only four words per line are printed.

d decimal
The data is printed in decimal format, eight words per line.

a ASCII
The data is interpreted as ASCII and printed as such. No more than 256 characters are printed in response to a single request.

i instruction
The data is printed in instruction format.

p pointer
The data is printed in pointer format, i.e., segment number and offset (and bit offset if nonzero).

s source statement
One or more source statement lines are printed starting with the line of source code that generated the code pointed to by the working pointer (assumed to be pointing into the text). For example:

```
/test/loop_here+32,s2
```

prints two lines of source code starting with the line that generated the code, 32 (octal) words after the label loop_here.

Another example:

```
/test/&a219,s
```

prints line number 219 (decimal) of test.lang where lang is the appropriate language suffix. Note that if there was no code generated for the specified line, debug prints a message, increments the line number, and tries again for up to 10 lines.

l code for line number
The code associated with the specified line number is printed in instruction format. The line number is determined as in s type output. For example:

```
/test/&a27,l
```

prints the code generated for line 27 (decimal) of test.lang.

n no output
No output. This is used to suppress output when changing defaults.

e floating point with exponent (single precision)

el long floating point with exponent (double precison)

f floating point (single precision)

fl long floating point (double precision)

b bit string
The data is printed as if it were a bit string. No more

than 72 bit positions are printed in response to a single request.

g graphic
The specified number of characters are interpreted as Multics standard graphics code. The type and value of each recognizable item is printed to the terminal. (Refer to the Multics Graphics System manual, Order No. AS40 for details.)

comp-5, comp-6, comp-7, comp-8
The data is printed as if it were a COBOL data type. If the size field is used for comp-5 or comp-8, it is the number of digits plus sign to use in printing the data.

comp-5	byte-aligned packed decimal
comp-6	full-word binary integer
comp-7	half-word binary integer
comp-8	digit-aligned packed decimal

debug (db)

Summary of Data and Control Requests

DATA REQUESTS

/seg name/	offset	seg ID	rel offset	operator	operands
pathname	number	&t	number	,	operands
ref name	symbol	&s	register	=	input list
seg number		&l		<	function list
&n seg name		&an		>	
seg$entry		&pn		:=	
		&i			

Segment ID	Operators	Registers	Output Modes
&t text	, print	$a	o octal
		$q	h half-carriage octal
&s stack	= assign	$aq	d decimal
		$eaq	a ASCII
		$x0	
&l linkage	< set break	.	i instruction
		.	
&i internal static		.	p pointer
&an source line	> transfer	$x7	s source statement
		$pr0	l code for line number
&pn parameter	:= call	.	n no output
		.	
		.	e floating point
		$pr7	el long floating point with exponent
		$exp	f floating point
		$tr	fl long floating point
			b bit string
			g graphic
		$ralr	comp-5 COBOL
		$ppr	comp-6 COBOL
		$tpr	comp-7 COBOL
		$even	comp-8 COBOL
		$odd	
		$ind	
		$prs	
		$regs	

debug (db)

```
$scu
$all
```

debug (db)

CONTROL REQUESTS

.t*i*,*j*	trace stack from frame *i* for *j* frames.
.+*i* or .-*i*	pop or push stack by *i* frames.
.*i*	set stack to *i*'th frame.
..	Multics command.
.d or .D	print default values.
.c,*i*	continue after break fault (ignore next *i* break fault).
.ct,*i*	continue, in temporary break mode.
.cr,*i*	continue, in normal mode.
.q	return from debug to caller.
.br*i*	reset break *i*.
.br	reset the breaks of the default object segment.
.bgr	reset all breaks.
.bl*i*	list break *i*.
.bl	list the breaks of the default object segment.
.bgl	list all breaks.
.be*i* <line>	execution line for break *i*.
.be <line>	execution line for all breaks of the. default object segment.
.bge <line>	execution line for all breaks.
.bo*i*	disable break *i*.
.bo	disable the break of the default object segment.
.bgo	disable all breaks.
.bn*i*	enable break *i*.
.bn	enable the breaks of the default object segment.
.bgn	enable all breaks.
.bgt <line>	establish a temporary global command.
.bc*i* a1 -rel- a2	make conditional break *i*.
.bc a1 -rel- a2	make conditional all breaks of default object segment.
.bs*i* *n*	set skips of break *i* to *n*.
.bd name/no.	set (or print) default object segment .
.bp	print names of all segments with breaks.
.a*i*,*m*	print argument *i* in mode *m*. (modes: o, p, d, a, b, l, e, f, ?).
.f	use registers from last fault.
.C	use crawlout registers .
.mb	change to brief output mode.
.ml	change to long output mode.

SYNTAX AS A COMMAND:

```
decat strA strB C
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[decat strA strB C]
```

FUNCTION: performs operations on bit or character strings. These operations are specified by a three digit bit string given last in the usage line.

ARGUMENTS:

strA, strB
are character strings, or bit strings entered as 0 and 1 characters.

C
where C is any three digit bit string expressed as 0 and 1 characters such as 000,001,...,111.

NOTES: The first occurrence of strB found in strA divides strA into three parts: part prior to strB, part matching strB, and part following strB. The digits given in C correspond to these three parts. The return string contains the parts of strA whose corresponding bit in C is 1. All three parts are returned in their original order of appearance in strA.

EXAMPLES:

Examples of active function usage follow:

```
! string [decat abcdef123ghi 123 110]
  abcdef123
! string [decat decat.incl.pl1 .incl 101]
  decat.pl1
```

SYNTAX AS A COMMAND:

```
dec values
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[dec values]
```

FUNCTION: returns one or more values in decimal.

ARGUMENTS:

value
is a value to be processed. The last character of the value indicates its type. Acceptable types are binary (b), quartenary (q), octal (o), hexadecimal (x), or unspecified (u). Any valid PL/I real value is allowed. The absence of any specifier means decimal. The unspecified value is limited to 8 characters.

EXAMPLES:

```
! string [dec 110.1b]
  6.5
```

SYNTAX AS A COMMAND:

```
decode path1A {path2A ... path1N path2N}
```

FUNCTION: reconstructs an original segment from an enciphered segment according to a key that is not stored in the system. The encode command is used to encipher segments.

ARGUMENTS:

path1A
is the pathname of an enciphered segment. The code suffix should not be specified because the command attaches the code suffix to the path1 argument (e.g., if the user types alpha_x.code as the path1 argument, the command attaches the suffix and looks for a segment named alpha_x.code.code). The star convention is allowed.

path2A
is the pathname of the deciphered segment to be produced. If the last path2 argument is missing, the command constructs a pathname from the path1 argument (see "Notes" below). The equal convention is allowed.

NOTES: The decode command requests the key from the terminal only once. All segments specified in an invocation of decode are deciphered with the same key.

If the last path2 argument is not given, the command places the deciphered segment in a segment whose name is the path1 argument, minus the code suffix.

EXAMPLES:

If the user types the command line:

```
! decode alpha_x
```

the command looks for an enciphered segment named alpha_x.code and places the deciphered segment produced in a segment named alpha_x.

SYNTAX AS A COMMAND:

```
default strA {strB}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[default strA {strB}]
```

FUNCTION: supplies default arguments to commands, and provides a means to override this default when desired. The default active function is designed to be used in conjunction with the abbrev and do commands.

NOTES: If strB is not specified or is the null string, strA is returned (see the second example below).

EXAMPLES: In the first example, the user sets up an abbreviation using the default active function to automatically compile a program with the -map and -table control arguments. The user can override the defaults by specifying more than one argument when using the abbreviation. Assume that comp_pl1 is an abbreviation for:

```
do "pl1 &1 [default ""-map -table"" &2] &f3"
```

Thus, typing "comp_pl1 test" is the same as typing "pl1 test -map -table"; typing "comp_pl1 test -list -profile" is the same as typing "pl1 test -list -profile".

The next example shows the null input feature of the default active function. Assume that my_dp is an abbreviation for:

```
do "dp -he [string [default [entry &1] &r2]]
     -q [default 3 &3] &f4 &1"
```

When the user types the command line:

```
!  my_dp >udd>Demo>Roy>design_memo.runout "" 2 -dl
```

the null input for the second argument means that default uses the default value for this argument (in this case, the entryname portion of the pathname). Thus, the expansion of the command line is:

```
dp -he design_memo.runout -q 2
     -dl >udd>Demo>Roy>design_memo.runout
```

default_wdir (dwd)

SYNTAX AS A COMMAND:

dwd

SYNTAX AS AN ACTIVE FUNCTION:

[dwd]

FUNCTION: returns the pathname of the default working directory of the process in which it is invoked, as set by the change_default_wdir (cdwd) command.

defer_messages (dm)

SYNTAX AS A COMMAND:

dm {destination} {-control_arg}

FUNCTION: prevents messages sent by the send_message command and the "You have mail." notification sent by the send_mail command from printing on the user's terminal. Instead, the user of send_message receives notification of the form "User has deferred messages. User_id.Project_id". The "You have mail" notifications are not saved.

ARGUMENT:

destination
can be of the form Person_id.Project_id to specify a mailbox. The default is the user's default mailbox. If destination contains < or >, it is assumed to be the pathname of a mailbox. This argument and the -pn path control argument are mutually exclusive.

CONTROL ARGUMENTS:

-pathname path, -pn path
specifies a mailbox by pathname. The mbx suffix is assumed. This control argument and the destination argument are mutually exclusive.

NOTES: The print_messages command prints messages that have been deferred.

The immediate_messages command restores the printing of messages as they are received.

For a description of the mailbox, refer to the accept_messages and print_mail commands.

SYNTAX AS A COMMAND:

dl {paths} {-control_args}

FUNCTION: causes the specified segments and/or multisegment files to be deleted. See also the delete_dir and unlink commands.

ARGUMENTS:

paths
are the pathnames of segments or multisegment files. The star convention is allowed.

CONTROL ARGUMENTS:

-brief, -bf
inhibits the printing of an error message if a segment or multisegment file to be deleted is not found.

-force
deletes the specified entries whether or not they are protected, without issuing a query.

-long, -lg
prints a message of the form "Deleted file <path>" for each entry deleted.

-name STR, -nm STR
specifies a nonstandard entry name STR (e.g., invalid starname such as **.**.compout or name containing <.)

ACCESS REQUIRED: The user must have modify permission on the containing directory.

NOTES: At least one path, or -name STR, must be specified.

In order to delete a segment or multisegment file with the delete command, the entry must have both its safety switch and its copy switch off. If either is on, the user is interrogated whether to delete the entry.

Use delete_dir to delete directories. Use unlink to delete links.

SYNTAX AS A COMMAND:

da {path} {User_ids} {-control_args}

FUNCTION: removes entries from the access control lists (ACLs) of segments, multisegment files, and directories.

ARGUMENTS:

path
is the pathname of a segment, multisegment file, or directory. If it is -wd, -working_dir, or omitted, the working directory is assumed. If path is omitted, no User_ids can be specified. The star convention is allowed.

User_ids
are access control names that must be of the form Person_id.Project_id.tag. All ACL entries with matching names are deleted. (For a description of the matching strategy, see set_acl in this manual.) If no User_ids are specified, the user's Person_id and current Project_id are assumed.

CONTROL ARGUMENTS:

-all, -a
deletes the entire ACL with the exception of an entry for *.SysDaemon.*.

-directory, -dr
deletes ACLs for only directories. The default is segments, multisegment files, and directories.

-segment, -sm
deletes ACLs for only segments and multisegment files.

-brief, -bf
suppresses the message "User name not on ACL."

ACCESS REQUIRED: The user must have modify permission on the containing directory.

NOTES: If the delete_acl command is invoked with no arguments, it deletes the entry for the user's Person_id and current Project_id on the ACL of the working directory.

An ACL entry for *.SysDaemon.* can be deleted only by specifying all three components as a User_id. The user should be aware that deleting access to the SysDaemon project prevents Backup.SysDaemon.* from saving the segment or directory (including the hierarchy inferior to the directory) on tape, Dumper.SysDaemon.* from reloading it, and Retriever.SysDaemon.* from retrieving it.

For a description of ACLs, see "Access Control" in the MPM Reference Guide.

EXAMPLES:

The command line:

```
! delete_acl news .Faculty. Jones
```

deletes from the ACL of news all entries with Project_id Faculty and the entry for Jones.*.*.

The command line:

```
! da beta.** ..
```

deletes from the ACL of every segment, multisegment file, and directory (in the working directory) whose entryname has a first component of beta all entries except the one for *.SysDaemon.*.

The command line:

```
! da beta.** .. -sm
```

deletes from the ACL of all segments and multisegment files (in the working directory) whose entrynames have a first component of beta all entries except the one for *.SysDaemon.*.

SYNTAX AS A COMMAND:

dd {paths} {-control_args}

FUNCTION: causes the specified directories and any segments, links, and multisegment files they contain, to be deleted. All inferior directories and their contents are also deleted.

ARGUMENTS:

paths
are pathnames of directories. The star convention is allowed.

CONTROL ARGUMENTS:

-brief, -bf
inhibits the printing of an error message if the directory to be deleted is not found.

-force
deletes the specified directories without issuing a query.

-long, -lg
prints a message of the form "Deleted directory <path>" for each directory deleted.

-name STR, -nm STR
specifies a nonstandard entry name STR (e.g., invalid starname such as **.**.compout or name which contains <.)

ACCESS REQUIRED: The user must have modify permission on both the directory and its superior directory.

NOTES: At least one path or -name must be specified.

Before deleting each specified directory, delete_dir asks the user whether to delete that directory. It is deleted only if the user types "yes".

When deleting a nonempty master directory, or a directory containing inferior nonempty master directories, the user must have previously mounted the logical volume(s). If a nonempty master directory for an unmounted volume is encountered, no

delete_dir (dd)

subtrees of that master directory are deleted, even if they are mounted.

Use delete to delete segments. Use unlink to delete link entries.

WARNING: Protected segments in pathi or any of its subdirectories are not deleted. Segments whose write bracket is less than the current ring (except for mailboxes and message segments) are also not deleted. Consequently, the subtree is not completely deleted if it contains any such segments. For a discussion of protected segments, see the safety switch attribute in the MPM Reference Guide. For a discussion of ring brackets, see "Intraprocess Access Control" in the MPM Reference Guide.

delete_iacl_dir (did)

SYNTAX AS A COMMAND:

```
did {path} {User_ids} {-control_args}
```

FUNCTION: deletes entries from a directory's initial access control list for directories (directory initial ACL). A directory initial ACL contains the ACL entries to be placed on directories created in the specified directory.

ARGUMENTS:

path
specifies a pathname of the directory whose directory initial ACL should be changed. If path is -wd, -working_dir, or omitted, the working directory is assumed. If path is omitted, no User_ids can be specified. The star convention is allowed.

User_ids
are access control names that must be of the form Person_id.Project_id.tag. All entries in the directory initial ACL that match User_id are deleted. (For a description of the matching strategy, refer to the set_acl command.) If no User_ids are specified, the user's Person_id and current Project_id are assumed.

CONTROL ARGUMENTS:

-all, -a
deletes the entire directory initial ACL with the exception of an entry for *.SysDaemon.*.

-ring N, -rg N
identifies the ring number whose directory initial ACL should be affected. N must be an integer such that user's ring $\leq$ N $\leq$ 7 (there is a separate initial ACL for each ring). If this control argument is not specified, the user's ring is assumed.

-brief, -bf
causes the message "User name not on ACL of path" to be suppressed.

ACCESS REQUIRED: The user must have modify (m) permission on the directory.

delete_iacl_dir (did)

NOTES: If the delete_iacl_dir command is given without any arguments, the ACL entry for the user's Person_id and current Project_id is deleted from the directory initial ACL of the working directory.

For a discussion of initial ACLs, see "Access Control" in the MPM Reference Guide.

EXAMPLES:

The command line:

```
! did news .Faculty Jones..
```

deletes from the directory initial ACL of the news directory all entries ending in .Faculty.* and all entries with Person_id Jones.

The command line:

```
! did -a
```

deletes all entries from the directory initial ACL of the working directory.

The command line:

```
! did store Jones -rg 5
```

deletes the entry for Jones.*.* from the ring 5 directory initial ACL of the store directory.

delete_iacl_seg (dis)

SYNTAX AS A COMMAND:

dis {path} {User_ids} {-control_args}

FUNCTION: deletes entries from a directory's initial access control list for segments (segments initial ACL). A segment initial ACL contains the ACL entries to be placed on segments created in the specified directory.

ARGUMENTS:

path
specifies the pathname of a directory whose segment initial ACL should be changed. If path is -wd, -working_dir, or omitted, the working directory is assumed. If path is omitted, no User_ids can be specified. The star convention is allowed.

User_ids
are access control names that must be of the form Person_id.Project_id.tag. All entries in the directory initial ACL that match the given User_ids are deleted. (For a description of the matching strategy, see set_acl in this manual.) If no User_ids are specified, the user's Person_id and current Project_id are assumed.

CONTROL ARGUMENTS:

-all, -a
deletes the entire initial ACL with the exception of an entry for *.SysDaemon.*.

-ring N, -rg N
identifies the ring number whose segment initial ACL should be affected. (There is a separate initial ACL for each ring.) N must be an integer such that user's ring $\leq$ N $\leq$ 7). If this control argument is not specified, the user's ring is assumed.

-brief, -bf
causes the message "User name not on ACL of path" to be suppressed.

ACCESS REQUIRED: The user must have modify (m) permission on the directory.

delete_iacl_seg (dis)

NOTES: If the delete_iacl_seg command is given without any arguments, the ACL entry for the user's Person_id and current Project_id is deleted from the segment initial ACL of the working directory.

For a discussion of initial ACLs, see "Access Control" in the MPM Reference Guide.

EXAMPLES:

The command line:

```
! dis news .Multics. Jones
```

deletes from the segment initial ACL of the news directory all entries with Project_id Multics and the entry for Jones.*.*.

The command line:

```
! dis -a
```

deletes all entries from the segment initial ACL of the working directory.

The command line:

```
! dis store Jones.. -rg 5
```

deletes all entries with Person_id Jones from the ring 5 segment initial ACL of the store directory.

delete_message (dlm)

SYNTAX AS A COMMAND:

dlm {destination} numbers {-control_args}

FUNCTION: deletes a message sent by the send_message command and saved in a mailbox with the -hold control argument to the accept_messages command. (See the accept_messages command for more details.)

ARGUMENT:

destination
can be of the form Person_id.Project_id to specify a mailbox. If destination contains either < or >, it is assumed to be the pathname of a mailbox. This argument and the -pathname control argument are mutually exclusive.

numbers
are message numbers as printed by the print_message command when accept_messages -hold is in effect.

CONTROL ARGUMENTS:

-all, -a
deletes all messages from the mailbox.

-pathname path, -pn path
specifies a mailbox by pathname. The mbx suffix is assumed. This control argument and the destination argument are mutually exclusive.

NOTES: If no mailbox is specified, the user's default mailbox is assumed. For a description of the mailbox, refer to the accept_messages and print_mail commands.

SYNTAX AS A COMMAND:

dn {paths} {-control_arg}

FUNCTION: deletes specified names from segments, multisegment files, links, or directories that have multiple names.

ARGUMENTS:

paths
are pathnames to be deleted. The star convention is allowed.

CONTROL ARGUMENTS:

-brief, -bf
suppresses error messages when entries are not found with specified pathnames. The default is -long (-lg).

-long, -lg
prints a message of the form "Deleted <path>" for each name deleted.

-name STR, -nm STR
specifies a nonstandard entry name STR (e.g., invalid starname such as *.compout or name containing <).

ACCESS REQUIRED: The user must have modify permission on the parent directory.

NOTES: At least one path or -name STR must be specified. In keeping with standard practice, each path can be a relative pathname or an absolute pathname; its final portion (the storage system entryname in question) is deleted from the storage system entry it specifies, provided that doing so does not leave the segment or directory without a name. If the entryname to be deleted is the only name on the storage system entry, an error message is printed.

See the descriptions of the add_name and rename commands for adding and changing names, respectively, on storage system entries.

delete_name (dn)

EXAMPLES:

The command line:

```
! dn alpha >my_dir>beta
```

deletes the name alpha from the list of names for the appropriate entry in the current working directory and also deletes the name beta from the list of names for the appropriate entry in the directory >my_dir. Neither alpha nor beta can be the only name for their respective entries.

SYNTAX AS A COMMAND:

dsp search_list search_paths {-control_arg}

FUNCTION: allows a user to delete one or more search paths from the specified search list.

ARGUMENTS:

search_list
is the name of the search list from which the specified search paths are deleted. It must be quoted if it contains spaces or other command language characters.

search_pathi
specifies a search path to be deleted. The search path can be an absolute or relative pathname or a keyword. It is necessary to use the same name that appears when the print_search_paths command is invoked.

CONTROL ARGUMENTS:

-all, -a
specifies that the search list itself is to be deleted. Any search paths specified are ignored. This control argument must be used to delete all the search paths in a search list.

NOTES: For a complete list of the search facility commands, see add_search_paths in this manual.

delete_search_rules (dsr)

SYNTAX AS A COMMAND:

dsr paths

FUNCTION: deletes search rules for object segments.

ARGUMENTS:

paths
are usually directory pathnames (relative or absolute) to be deleted from the current search rules. One of the paths can be the keyword working_dir (see "Note" below).

NOTES: Site-defined keywords and the home_dir and process_dir keywords are not accepted by delete_search_rules although they are accepted by the add_search_rules command. Although the delete_search_rules command does accept the keywords initiated_segments and referencing_dir, their deletion is discouraged and may lead to unpredictable results.

detach_audit (dta)

SYNTAX AS A COMMAND:

dta {switchname}

FUNCTION: removes audit_ from the specified switch and restores the switch to the state of the switch prior to invoking attach_audit.

ARGUMENTS:

switchname
is the switch from which audit_ is to be removed. If switchname is not specified, user_i/o is assumed.

NOTES: For further information about the audit facility, see audit_ in the MPM Subroutines, and attach_audit in this manual.

detach_lv (dlv)

SYNTAX AS A COMMAND:

dlv volume_names

FUNCTION: detaches one or more logical volumes that have been attached for the user's process by the resource control package (RCP).

ARGUMENTS:

volume_names
specify the volumes to be detached.

NOTES: A user can detach all logical volumes attached for the process by specifying "all" rather than any volume names.

The detaching of a logical volume involves telling the storage system that the logical volume is no longer attached for this process. The detaching of a logical volume does not affect the attached/detached state of the logical volume for any other process.

SYNTAX AS A COMMAND:

```
dirs star_names {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[dirs star_names {-control_arg}]
```

FUNCTION: returns the entrynames or absolute pathnames of directories that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames. The default is to return entrynames.

NOTES: Only one name per directory is returned; i.e., if a directory has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by directories is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

EXAMPLES:

```
! pwd
>udd>Apple>Jones
! ls -a

Segments = 3, Lengths = 4.

re      1  prog
r w     1  prog.list
r w     2  prog.pl1
```

directories (dirs)

```
      Multisegment-files = 2, Lengths = 770.

      r w  513  prog.output
      r w  257  prog.data

      Directories = 2.

      sma  prog_stuff
      sma  documents

      Links = 1.

      prog.temp1                  >udd>Apple>Jones>temp_seg_1

!     string [dirs **]
      documents prog_stuff
!     string [dirs prog*.**]
      prog_stuff
```

SYNTAX AS A COMMAND:

```
directory path
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[directory path]
```

FUNCTION: returns the directory portion of path, after it has been expanded into an absolute pathname.

NOTES: See the description of entry in this manual.

EXAMPLES:

In the working directory >udd>m>Jones:

```
! string [directory >udd>m>Jones>foo]
>udd>m>Jones

! string [directory bar>foo]
>udd>m>Jones>bar
```

discard_output (dco)

SYNTAX AS A COMMAND:

dco {-control_arg} command_line

FUNCTION: executes a command line while temporarily suppressing output on specified I/O switches.

CONTROL ARGUMENTS:

-output_switch STR, -osw STR
where STR is the name of an I/O switch. If no control arguments are specified, output on the user_output I/O switch is suppressed.

ARGUMENTS:

command_line
is a command line. It need not be quoted.

NOTES: If the command specified in command_line cannot be executed, an error message is printed.

display_audit_file (daf)

SYNTAX AS A COMMAND:

daf {path} {-control_args}

FUNCTION: displays the file produced by the audit_ I/O module.

ARGUMENTS:

path
is the pathname of the audit file to be displayed. If path is not specified, the audit file associated with the user_i/o switch is assumed. If user_i/o is not being audited, the audit file currently in use is the one that is displayed.

CONTROL ARGUMENTS:

-from STR, -fm STR
specifies the first audit file entry to be displayed. If STR is a positive integer, it is interpreted as an entry number. If STR is a positive number containing a decimal point, it is interpreted as a time in 24 hour format. If STR does not fit either of the above cases, the audit file is displayed from the first entry which matches STR. If -from is not specified, the audit file is displayed from the beginning.

-to STR
stops the display of the audit file at the point specified by STR where STR can have any of the values as described for the -from control argument. If -to is not specified, the audit file is displayed up to the end.

-next STR
displays a specified number of entries from an explicit point in the file to the point specified by STR. If STR is in entry number format, the next STR entries are displayed. If STR is in time format, the entries within the next STR period of time after the beginning entry are displayed. If STR is a character string, the entries up to the next match of STR are displayed. If -next is not specified, all entries to the end of the file are displayed.

-last STR
starting at the end of the audit file, the -last control argument displays entries beginning at the point specified by STR. If STR is in entry number format, the first entry displayed is STR entries back from the end of the file. If STR is in time format, the first entry is STR hours and

minutes from the end of the file. If STR is a character string, the first entry contains a match for STR searching from the end of the file. If this control argument is not specified, the audit file is displayed from the beginning.

-match STR1 { ... STRn}
prints entries containing strings matching any of the STRs. If this control argument is not specified, all otherwise selected entries are printed.

-exclude STR1 { ... STRn}, -ex STR1 { ... STRn}
excludes any entries containing strings matching any of the STRs. If this control argument is not specified, all otherwise selected entries are printed.

-class STR1 { ... STRn}
prints the entries having a class identifier matching any of the STRs. Class indentifiers are as follows:

EL - edit line
IL - input line
IC - input characters
OC - output characters
TM - trace of modes operations
TC - trace of control operations

If STR contains only one character, it is matched on the first character of the class identifiers. For example, if STR is I, entries having either IL and IC class identifiers are displayed. If -class is not specified, the audit file is displayed without class identifiers.

-reverse
prints the entries in reverse chronological order. If this control argument is not specified, the entries are printed in chronological order.

-switch STR
displays the audit file associated with the I/O switch specified by STR if the I/O switch is currently attached. If the I/O switch is not attached, an error message is printed. If this control argument is not specified, the audit file associated with the user_i/o switch is displayed.

-entry_numbers, -etn
prints the entry numbers before each entry.

-metering, -mt
displays the audit file with metering information at the beginning of each line, preceding the class identifiers if the

display_audit_file (daf)

-class or -class_identifiers control arguments are also specified.

-class_identifiers, -cli
displays the audit file with the class identifiers before each entry. If the -metering control argument is also specified, the metering information precedes class identifiers.

-string STR, -str STR
uses STR as a character string with no special interpretation. This is useful for preventing STR from being interpreted as a control argument, a time, or an entry number. It can be used with the -from, -to, -next, -last, -match, and -exclude control arguments, for example "-from -string 81".

-line_length N, -ll N
inserts a newline after the character specified by N if a line of output is greater than N characters in length. A continuation line is indented to allow for any entry descriptors produced by the -metering, -entry_numbers, or -class_identifiers control arguments and preceded by an "*" to indicate it is a continuation of the previous line.

NOTES: The format of the output, which entries are selected to be output, and the file to which the output is directed are all specified by the user. For more information on the audit facility, see the attach_audit description in this manual and the audit_ I/O module in MPM Subroutines.

The audit_meter mode must have been on for there to be any metering information in the audit file. Without this information, time arguments are invalid. See the audit_ I/O module in the MPM Subroutines.

The -string control argument is useful in the following situations. To pass 1005.2 as a character string to be matched, rather than a value of time for the control argument -from type:

```
! daf -from -string 1005.2
```

To pass -last as a character string to the control argument -match type:

```
! daf -match -string -last
```

display_audit_file (daf)

EXAMPLES:

```
!  daf -from 5 -to 20 -ll 50 -metering -class_identifiers

  time  cpu usage   paging class
1306.0    0.037        8  IL *pr dump_truck.pl1 1 5
1306.0    0.019       13  OC  dump_truck: proc;

                           dcl expand_pathname_$add_suff
                          *ix entry (char (*), char (*),
                          * char (*), char (*), fixed bi
                          *n (35));
                           dcl get_truck_$plates entry (
                          *char (*), char (*), ptr, fixe
                          *d bin (35));
                           dcl dump_truck_ entry (ptr, f
                          *ixed bin (35));
1306.0    0.021        7  IL  rdy
1306.0    0.019        0  OC  r 12:07 0.186 level 2,15
1306.0    0.076        6  IL  ls *.compin -prn
1306.5    0.014        0  OC
                           Segments = 2, Lengths = 2.
1306.5    0.011        0  OC  re      1  ci_setup.compin  (1
                          *3)
1306.5    0.008        0  OC  re      1  hex.compin
1306.5    0.016       13  OC
1306.5    0.046       17  IL  la hex.compin
1306.5    0.011        0  OC  rew   Carry.Multics.*
1306.5    0.010        0  OC  re    *.SysDaemon.*
1306.5    0.009        0  OC  re    *.*.*


!  daf -last 0010.0 -entry_numbers

   entry
   22  fo out -last 0010.0 -entry_numbers;ro
   23  file_output: Specified control argument is not
          implemented by this command. -last
   24  revert_output: No previous attachment of user_output
   25  r 1345 0.111 106
   26  fo out;daf -last 0010.0 -entry_numbers;ro
```

display_audit_file (daf)

```
! daf -from cwd -next cwd -reverse -switch foo -entry_numbers

   entry

   6 cwd use>AG92r2B
   5 r 945 0.075 5.580 112

   4 when the cart stops
     do you whip the cart
     or whip the ox?

   3 say
   2 r 945 0.042 87

   1 cwd
```

In the example above, the display_audit_file command locates the first occurrence of cwd in the switch foo. It then goes to the next occurrence of cwd and prints the results in reverse order with the entry_numbers displayed on the left.

display_cobol_run_unit (dcr)

SYNTAX AS A COMMAND:

dcr {-control_args}

FUNCTION: displays the current state of a COBOL run unit.

CONTROL ARGUMENTS:

-long, -lg
displays more detailed information about each COBOL program in the run unit.

-files
displays information about the current state of the files that have been referenced during the execution of the current run unit.

-all, -a
prints information about all programs in the run unit, including those that have been cancelled.

NOTES: The minimal information displayed tells which programs compose the run unit. Optionally, more detailed information can be displayed concerning active files, data location, and other aspects of the run unit. Refer to the run_cobol command for information concerning the run unit and the COBOL runtime environment.

The user is also referred to stop_cobol_run (scr) and cancel_cobol_program (ccp) in this manual.

display_pl1io_error (dpe)

SYNTAX AS A COMMAND:

dpe

FUNCTION: describes the most recent file on which a PL/I I/O error was raised and displays diagnostic information associated with that type of error. The display_pl1io_error command is designed to be invoked after the occurrence of an I/O error signal during a PL/I I/O operation.

EXAMPLE:

The command line:

```
! dpe
```

might respond with the following display:

```
Error on file afile
Title: vfile_ afile
Attributes: open input keyed record sequential
Last i/o operation attempted: write from
Attempted "write" operation conflicts with file "input"
   attribute.
Attempted "from" operation conflicts with file "input"
   attribute.
```

SYNTAX AS A COMMAND:

```
divide numA numB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[divide numA numB]
```

FUNCTION: returns the integer part of the decimal quotient of numA divided by numB.

NOTES: See the description of mod and quotient in this manual.

EXAMPLES:

```
! string [divide 5 4]
  1
```

SYNTAX AS A COMMAND:

```
do {command_string} {args}
  or
do {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[do command_string args]
```

FUNCTION: substitutes arguments into a command string. The expanded command line is then passed to the current Multics command processor for execution. If abbreviations are being expanded in the user's process, any abbreviations in the expanded command line are expanded. Since the command line supplied to the do command is enclosed in quotation marks, abbreviations in it are not expanded before do operates on it. Control arguments can be used to set the mode of operations.

As an active function, evaluates to the expanded command line, without executing it.

ARGUMENTS:

command_string
is a command line enclosed in quotation marks. Each instance of the parameter designator &i (where i is a number from 1 to 9) found in command_string is replaced by argi. If any argi is not supplied, each instance of &i in command_string is replaced by the null string. Each instance of the parameter designator &fi is replaced by the arguments argi through the last argument supplied, separated by single spaces. Each instance of the string &n is replaced by the number of arguments supplied. The parameters &qi, &ri, &qfi, and &rfi are replaced by quoted arguments. (See "Quote-Doubling and Requoting" below.) Each instance of the unique-name designator &! found in command_string is replaced by a 15-character identifier unique to the particular invocation of the do command. Finally, each instance of the ampersand pair && is replaced by a single ampersand. Any other ampersand discovered in command_string causes an error message to be printed and the expansion to be terminated.

argi
is a character string argument. Any argument supplied but not referenced in a parameter designator is ignored.

CONTROL ARGUMENTS:
set the mode of operation of the do command. Control arguments can only be specified if neither a command_string nor args are given. Control_args can be one or more of the following:

-long, -lg
prints the expanded command line on error_output before it is executed or passed back.

-brief, -bf
suppresses printing of the expanded command line. This is the default.

-nogo
does not pass the expanded command line to the command processor. This control argument is ignored if do is invoked as an active function.

-go
passes the expanded command line to the command processor. This is the default. This control argument is ignored if do is invoked as an active function.

-absentee
establishes an on unit for the any_other condition during the execution of the expanded command line. See "Notes on Modes" below for additional information about the -absentee control argument.

-interactive
does not catch any signals. This is the default. (See "Notes on Modes" below.)

NOTES ON MODES: The do command has three modes, the long/brief mode, the nogo/go mode, and the absentee/interactive mode. These modes are kept in internal static storage and are thus remembered from one invocation of do to the next within a single process. The modes are set by invoking the do command with control arguments and are described under "Control Arguments" above.

The absentee mode is mainly of use in an absentee environment, in which any invocation of the default any_other on unit terminates the process. In the absentee mode, any signal caught by the do command merely terminates execution of the command line, not the process. A number of conditions, however, are not handled by the do command but are passed on

for their standard Multics treatment; they are quit, program_interrupt, command_error, command_query_error, command_question, and record_quota_overflow. (For a description of these conditions see "List of System Conditions and Default Handlers" in the MPM Reference Guide.)

NOTES ON QUOTE-DOUBLING AND REQUOTING: In addition to the parameter designators &1 ... &9, the do command also recognizes two more sets of parameter designators. They are &q1 ... &q9, to request quote-doubling in the actual argument as it is substituted into the expanded command line, and &r1 ... &r9, to request that the actual argument be requoted as well as have its quotes doubled during substitution.

Quote-doubling can be described as follows. Each parameter designator in the command_string to be expanded is found nested a certain level deep in quotes. If a designator is found to not be within quotes, then its quote-level is zero; if it is found between a single pair of quotes, then its quote-level is one; and so on. If the parameter designator &qi is found nested to quote-level L, then, as argi is substituted into the expanded command line each quote character found in argi is replaced by 2**L quote characters during insertion. This permits the quote character to survive the quote-stripping action to which the command processor subsequently subjects the expanded command line. If &qi is not located between quotes, or if argi contains no quotes, then the substitutions performed for &qi and for &i are identical. The string &qfi is replaced by a list of the ith through last arguments with their quotes doubled.

If the parameter designator &ri is specified, the substituted argument argi is placed between an additional level of quotes before having its quotes doubled. More precisely, if the parameter designator &ri is found nested to quote-level L, 2**L quotes are inserted into the expanded line, argi is substituted into the expanded line with each of its quotes replaced by 2**(L+1)quotes, and 2**L more quotes are placed following it. If argument argi is not supplied, nothing is placed in the expanded line; this provides a way to distinguish between arguments that are not supplied and arguments that are supplied but are null. If argument argi is present, the expansions of &ri, and of &qi written between an additional level of quotes, are identical. The string &rfi is replaced by a list of the ith through last arguments, requoted.

NOTES ON ACCESSING MORE THAN NINE ARGUMENTS: In addition to the normal parameter designators in which the argument to be substituted is specified by a single integer, the do command accepts the designators &(d...d), &f(d...d), &r(d...d), and &q(d...d) where d...d denotes a string of decimal digits. An error message is printed and the expansion is terminated if any character other than 0 ... 9 is found between the parentheses.

NOTES: For a description of abbreviation expansion, see abbrev in this manual.

EXAMPLES: The do command is particularly useful when used in conjunction with the abbreviation processor, initialized by the abbrev command. Consider the following abbreviations:

```
ADDPLI   do "fo &1.list;ioa_ ^|;pli &1;ro"
AUTHOR   do "ioa_$nnl &1;status -author &1"
CREATE   do "cd &1;sis &1 re *.Demo rew Jay.*"
LIST     do "fo Jay.list;LISTAB;ws &1 LISTAC;ro;
              dp -dl Jay.list"
LISTAB   do ".l"
LISTAC   "la;ls -dtem -a"
P        do "pl1 &1 -list &2 &3"
P2       do "pl1 &1 -list &f2"
```

The command line:

```
! ADDPLI alpha
```

expands to:

```
fo alpha.list;ioa_ ^|;pli alpha;ro
```

The command line:

```
! AUTHOR beta
```

prints beta and the author of segment beta.

The command line:

```
! CREATE games
```

expands to:

```
cd games;sis games re *.Demo rew Jay.*
```

This shows an easy method of automatically setting initial access on the segments that will be cataloged in a newly created directory.

The command line:

```
! LIST >udd>Demo>Jay
```

expands to:

```
fo Jay.list;LISTAB;ws >udd>Demo>Jay LISTAC;ro;dp -dl Jay.list
```

that is expanded by abbrev to:

```
fo Jay.list;do ".l";ws >udd>Demo>Jay "la;ls -dtem -a";ro;
     dp -dl Jay.list
```

This shows how do can be used at several levels and how it allows abbreviations to be used within abbreviations.

The command line:

```
! P alpha
```

generates the expansion:

```
pl1 alpha -list
```

whereas the command line:

```
! P alpha -table
```

expands to:

```
pl1 alpha -list -table
```

This shows how references to unsupplied arguments are deleted.

The abbreviation P2 is equivalent to P for three or fewer arguments. The command line:

```
! P2 alpha -table -sv3 -optimize
```

executes the pl1 command with the -list, -table, -sv3, and -optimize control arguments, whereas:

```
! P alpha -table -sv3 -optimize
```

omits the -optimize control argument.

SYNTAX AS A COMMAND:

dp {-control_args} {paths}

FUNCTION: queues specified segments and/or multisegment files for printing on one of the Multics line printers. The output is by default identified by the requestor's Person_id. This command does not accept standard object segments.

ARGUMENTS:

paths
are pathnames of segments and/or multisegment files. The star convention is NOT allowed.

CONTROL ARGUMENTS:

-access_label, -albl
for each pathi specified, uses the access class of that segment as a label at the top and bottom of every page (see "Notes" below).

-brief, -bf
suppresses the message "j requests signalled, k already queued. (request_type queue)." This control argument cannot be overruled later in the command line. (See the -request_type and -queue control arguments below.)

-bottom_label STR, -blbl STR
uses the specified string as a label at the bottom of every page (see "Notes" below).

-copy N, -cp N
prints N copies ($N \leq 4$) of specified paths. This control argument can be overruled by a subsequent -copy control argument. If pathi is to be deleted after printing, all N copies are printed first. If this control argument is not specified, one copy is made.

-delete, -dl
deletes (after printing) specified paths.

-destination STR, -ds STR
labels subsequent output with the string STR, which is used to determine where to deliver the output. If this control argument is not specified, the default is the requestor's

Project_id. This argument can be overruled by a subsequent -destination control argument.

-header STR, -he STR
identifies subsequent output by the string STR. If this control argument is not specified, the default is the requestor's Person_id. This argument can be overruled by a subsequent -header control argument.

-indent N, -in N
prints specified paths so that the left margin is indented N columns. If this control argument is not specified, no indentation occurs.

-label STR, -lbl STR
uses the specified string as a label at the top and bottom of every page (see "Notes" below).

-line_length N, -ll N
prints specified paths so that lines longer than N characters are continued on the following line, i.e., no line of output extends past column N. If this control argument is notspecified, a line length of 136 characters is used.

-no_endpage, -nep
prints specified paths so that the printer skips to the top of a page only when a form-feed character is encountered in the input path. This argument causes the -page_length control argument, if present, to be ignored.

-no_label, -nlbl
does not place any labels on the printed output.

-non_edited, -ned
prints nonprintable control characters as octal escapes rather than suppressing their printing.

-notify, -nt
sends a confirming message when the requested output is done, showing the pathname and charge.

-page_length N, -pl N
prints specified paths so that no more than N lines are on a page. If this control argument is notspecified, a page length of 60 lines is used.

-queue N, -q N
prints specified paths in priority queue N. This control argument can be overruled by a subsequent -queue control

argument. If this control argument is notspecified, queue 3 is assumed. (See "Notes" below.)

-request_type STR, -rqt STR
places specified paths in the queue for requests of the type identified by the string STR (see "Notes" below). If this control argument is notspecified, the default request type is "printer".

-single, -sg
prints specified paths so that any form-feed or vertical-tab character in any of the paths is printed as a single newline character.

-top_label STR, -tlbl STR
uses the specified string as a label at the top of every page (see "Notes" below).

-truncate, -tc
prints specified paths so that any line exceeding the line length is truncated rather than "folded" onto subsequent lines.

NOTES: If the dprint command is invoked without any arguments, the system prints a message giving the status of queue 3.

If control arguments are present, they affect only paths specified after their appearance in the command line. If control arguments are specified without a following pathi argument, they are ignored for this invocation of the command and a warning message is printed.

The -queue 1 control argument places requests in the top priority queue, -queue 2 places them in the second priority queue, and -queue 3 (or not specifying a queue) places them in the third priority queue. The existence of lower priority queues for a specified request type is determined by the site. All requests in the first queue are processed before any requests in the other queues, and so on. Higher priority queues usually have a higher cost associated with them.

The -brief, -delete, -single, -truncate, and -no_endpage control arguments cannot be reset in aspecified invocation of the command; e.g., once -delete appears in a line, all subsequently specified paths are deleted after printing.

The -request_type control argument is used to ensure that a request is performed by a member of a particular group of printers, e.g., to distinguish between onsite printers and remote printers at various locations, or between printers being charged to different projects. Only request types of generic type "printer" can be specified. Request types can be listed by the print_request_types command.

If a requested output operation cannot be done, the daemon process sends a message to the user of the form:

Request path reason.

The -label, -top_label, -bottom_label, and -access_label control arguments allow the user to place labels on each page of printed output. The default labels are access labels, i.e. the -access_label control argument is assumed. These control arguments are read, in sequence, from left to right by the dprint command. For example, if -access_label is specified, it is printed at the top and bottom of the page. If the next control argument is -top_label STR, then the top access label becomes STR but the bottom label remains the same. Each label control argument can override the preceding one. The label lines are printed on the second line of the page and on the next to last line of the page. Note that if the access class of pathi is system_low and the access class name defined for system_low is null, then the default access label is blank. The default access label can be overridden by the -no_label control argument if labels are not wanted or by one of the other label-related control arguments.

The top and bottom labels are treated independently. Thus, use of the -top_label control argument alone leaves an access label as the default bottom label. A page label that exceeds 136 characters is truncated to that length. Only the first line of a page label is printed, i.e., a new line terminates the page label. Form feeds and vertical tabs are not permitted. The various label control arguments are incompatible with the -no_endpage control argument and they are ignored independent of the position in the command line of the -no_endpage control argument.

Segments and multisegment files cannot be printed unless appropriate system processes have sufficient access. The process that runs devices of the specified class (normally IO.SysDaemon) must have read access to all paths to be printed

and status permission on the containing directory. Pathi cannot be deleted after printing unless its safety switch is off and the system process has at least sm access on the containing directory. Also, pathi is not deleted if it has a date-time-contents-modified value later than the date-time-contents-modified value at the time of the dprint request.

The dprint command does not accept the star convention. It prints a warning message if a name containing asterisks is encountered, and continues processing its other arguments.

If pathi specifies a standard Multics storage system object segment, the dprint command prints a warning message and continues processing its other arguments.

EXAMPLES:

The command line:

```
! dp -he Jones -ds BIN-5 -cp 2 -dl test1 test7 -he Doe
      text.runout
```

causes two copies of each of the segments named test1 and test7 in the current working directory to be printed with the header "Jones" and the destination "BIN-5", and then deleted. It also causes two copies of the segment named text.runout in the current working directory to be printed with the header "Doe" and destination "BIN-5", then deleted.

SYNTAX AS A COMMAND:

dpn {-control_args} {paths}

FUNCTION: queues specified segments and/or multisegment files for punching by the Multics card punch. It is similar to the dprint command.

ARGUMENTS:

paths
are pathnames of segments and/or multisegment files. The star convention is NOT allowed.

CONTROL ARGUMENTS:

-brief, -bf
suppresses the message "j requests signalled, k already queued. (request_type queue)." This control argument cannot be overruled later in the command line. (See the -request_type and -queue control arguments below.)

-copy N, -cp N
punches N copies ($N \leq 4$) of specified paths. This control argument can be overruled by a subsequent -copy control argument. If pathi is to be deleted after punching, all N copies are punched first. If this control argument is not specified, one copy is made.

-delete, -dl
deletes (after punching) all specified paths.

-destination STR, -ds STR
uses the string STR to determine where to deliver the deck. If this control argument is not specified, the default is the requestor's Project_id. This control argument can be overruled by a subsequent -destination control argument.

-header STR, -he STR
identifies subsequent output by the string STR. If this control argument is not specified, the default is the requestor's Person_id. This control argument can be overruled by a subsequent -header control argument.

dpunch (dpn)

-mcc
punches the specified paths using character conversion. This control argument can be overruled by either the -raw or -7punch control arguments.

-notify, -nt
sends a confirming message when the requested output is done, showing the pathname and charge.

-queue N, -q N
punches psecified paths in priority queue N ($N \leq 3$). This control argument can be overruled by a subsequent -queue control argument. If this control argument is not specified, queue 3 is assumed. (See "Notes" below.)

-raw
punches the specified paths using no conversion. This control argument can be overruled by either the -mcc or -7punch control arguments.

-request_type STR, -rqt STR
places specified paths in the queue for requests of the type identified by the string STR (see "Notes" below). If this control argument is not specified, the default request type is "punch."

-7punch, -7p
punches the specified paths using 7-punch conversion. For a description of conversion modes, see "Bulk Input/Output" in the MPM Reference Guide.

NOTES: See "Input and Output Facilities" in the MPM Reference Guide for information on the input/output system.

If the dpunch command is invoked without any arguments, the system prints a message giving the status of queue 3.

If control arguments are present, they affect only paths specified after their appearance on the command line. If control arguments are specified without a following pathi argument, they are ignored for this invocation of the command and a warning message is printed.

dpunch (dpn)

The -queue 1 control argument places requests in the top priority queue, -queue 2 places them in the second priority queue, and -queue 3 (or not specifying a queue) places them in the third priority queue. The existence of lower priority queues for a specified request type is determined by the site. All requests in the first queue are processed before any requests in the second queue, and so on. Higher priority queues usually have a higher cost associated with them.

The -delete control argument is the only control argument affecting segments that cannot be reset in a specified invocation of the command. Once -delete appears in a line, all subsequent segments are deleted after punching.

The -request_type control argument is used to ensure that a request is performed by a member of a particular group of punches, for example, to distinguish between onsite punches and remote punches at various locations, or between punches being charged to different projects. Only request types of generic type "punch" can be specified. Request types can be listed by the print_request_types command.

If a requested output operation cannot be done, the daemon process sends a message to the user of the form:

"Unable to punch" path reason.

A segment or multisegment file cannot be punched unless appropriate system processes have sufficient access. The process (normally IO.SysDaemon) that runs devices of the specified class must have read access to all files to be punched and status permission on the containing directory. A file cannot be deleted after punching unless its safety switch is off and the system process has at least sm permission on the containing directory. Also, a file is not deleted if it has a date-time-contents-modified value later than the date-time-contents-modified value at the time of the dpunch request.

The dpunch command does not accept the star convention; it prints a warning message if a name containing asterisks is encountered and continues processing its other arguments.

It is suggested that the user, before deleting a file that has been punched, read the deck back in and compare it with the original (using the compare command) to ensure the absence of errors.

EXAMPLES:

The command line:

```
! dpunch a b -mcc -he Doe c.pl1 -dl -7p -he "J. Roe" alpha
```

causes segments a and b in the current working directory to be punched using 7-punch conversion; segment c.pl1 to be punched using character conversion with "for Doe" added to the heading; and segment alpha to be punched using 7-punch conversion (and then deleted) with "for J. Roe" added to the heading.

dump_segment (ds)

SYNTAX AS A COMMAND:

ds path {offset} {length} {-control_args}
or
ds seg_no {offset} {length} {-control_args}

SYNTAX AS AN ACTIVE FUNCTION:

[ds path {offset} {-control_args}]
or
[ds seg_no {offset} {-control_args}]

FUNCTION: prints, in octal or hexadecimal format, selected portions of a segment. It prints out either four or eight words per line and can optionally be instructed to print out an edited version of the ASCII, BCD, EBCDIC (in 8 or 9 bits), or 4-bit byte representation.

The active function returns a single word in octal or hexadecimal representation.

ARGUMENTS:

path
is the pathname or (octal) segment number of the segment to be dumped. If path is a pathname, but looks like a number, the preceding argument should be the -name (or -nm) control argument (see below). The star convention is allowed for the command only.

offset
is the (octal) offset of the first word to be dumped. If both offset and length are omitted, the entire segment is dumped.

length
is the (octal) number of words to be dumped. If offset is supplied and length is omitted, 1 word is dumped.

segno
is the octal segment number of a segment to be dumped.

CONTROL ARGUMENTS:

-4bit
prints out, or returns, a translation of the octal or hexadecimal dump based on the Multics unstructured 4-bit byte.

The translation ignores the first bit of each 9-bit byte and uses each of the two groups of four bits remaining to generate a digit or a sign.

-address, -addr
prints the address (relative to the base of the segment) with the data. This is the default.

-bcd
prints the BCD representation of the words in addition to the octal or hexadecimal dump. There are no nonprintable BCD characters, so periods can be taken literally. This control argument causes the active function to return BCD.

-block N, -bk N
dumps words in blocks of N words separated by a blank line. The offset, if being printed, is reset to initial value at the beginning of each block.

-character, -ch, -ascii
prints the ASCII representation of the words in addition to the octal or hexadecimal dump. Characters that cannot be printed are represented as periods. This control argument causes the active function to return ASCII.

-ebcdic9
prints the EBCDIC representation of each 9-bit byte in addition to the octal or hexadecimal dump. Characters that cannot be printed are represented by periods. This control argument causes the active function to return 9-bit EBCDIC.

-ebcdic8
prints the EBCDIC representation of each eight bits in addition to the octal or hexadecimal dump. Characters that cannot be printed are represented by periods. If an odd number of words is requested to dump, the last four bits of the last word do not appear in the translation. This control argument causes the active function to return 8-bit EBCDIC.

-header, -he
prints a header line containing the pathname (or segment number) of the segment being dumped as well as the date-time printed. The default is to print a header only if the entire segment is being dumped, i.e., if neither the offset nor the length argument is specified.

-hex8
prints the dumped words in hexadecimal with nine hexadecimal digits per word rather than octal with 12 octal digits per word.

-hex9
prints the dumped words in hexadecimal with eight hexadecimal digits per word rather than 12 octal digits per word. Each pair of hexadecimal digits corresponds to the low-order eight bits of each 9-bit byte.

-long, -lg
prints eight words on a line. Four is the default. This control argument cannot be used with -character, -bcd, -4bit, -ebcdic8, -ebcdic9, or -short. (Its use with these control arguments, other than -short, results in a line longer than 132 characters.)

-name PATH, -nm PATH
indicates that PATH is a pathname even though it may look like an octal segment number.

-no_address, -nad
does not print the address.

-no_header, -nhe
suppresses printing of the header line even though the entire segment is being dumped.

-no_offset, -nofs
does not print the offset. This is the default.

-offset N, -ofs N
prints the offset (relative to N words before the start of data being dumped) along with the data. If N is not given, 0 is assumed.

-short, -sh
compacts lines to fit on a terminal with a short line length. Single spaces are placed between fields, and only the two low-order digits of the address are printed, except when the high-order digits change. This shortens output lines to less than 80 characters.

NOTES: Only one of the control arguments: -ebcdic8, -ebcdic9, -character, -bcd, or -4bit can be specified.

When invoked as an active function dump_segment returns only one word of information, which is located at offset within the segment. If the -4bit, -bcd, -character, -ebcdic9, -ebcdic8, -hex8, or -hex9 control arguments are invoked, the information is returned in the specified format only. All other arguments are ignored in active function invocation.

SYNTAX AS A COMMAND:

edm {path}

FUNCTION: invokes a simple Multics context editor. It is used for creating and editing ASCII segments. This command cannot be called recursively.

ARGUMENTS:

path
specifies the pathname of a segment to be created or edited. If path is not specified, edm begins in input mode (see "Notes" below), ready to accept whatever is subsequently typed as input. If path is specified, but the segment does not yet exist, edm also begins in input mode. If path specifies a segment that already exists, edm begins in edit mode.

LIST OF EDITOR REQUESTS:

-	backup
=	print current line number
,	comment mode
.	mode change
b	bottom
c	change
d	delete
E	execute
f	find
i	insert
k	kill
l	locate
merge	insert segment
move	move lines within segment
n	next
p	print
q	quit
qf	quitforce
r	retype
s	substitute
t	top
updelete	delete to pointer
upwrite	write to pointer (upper portion of segment)
v	verbose
w	write

NOTES: This command operates in response to requests from the user. To issue a request, the user must cause edm to be in edit mode. This mode is entered in two ways: if the segment already exists, it is entered automatically when edm is invoked; if dealing with a new segment (and edm has been in input mode), the mode change character must be issued. The mode change character is the period (.), issued as the only character on a line. The command announces its mode by typing "Edit." or "Input." when the mode is entered. From edit mode, input mode is also entered via the mode change character.

The edm requests are predicated on the assumption that the segment consists of a series of lines to which there is a conceptual pointer that indicates the current line. (The "top" and "bottom" lines of the segment are also meaningful.) Various requests explicitly or implicitly cause the pointer to be moved; other requests manipulate the line currently pointed to. Most requests are indicated by a single character, generally the first letter of the name of the request. For these requests only the single character (and not the full request name) is accepted by the command. Certain requests have been considered sufficiently dangerous, or likely to confuse the unwary user, that their names must be specified in full.

If the user issues a quit signal while in edit mode and then invokes the program_interrupt command, the effect of the last request executed on the edited copy is nullified. (See description of program_interrupt in this manual.) In addition, any requests not yet executed are lost. If program_interrupt is typed after a quit in comment or input modes, then all input since last leaving edit mode is lost. A user wishing to keep the input must invoke the start command following the quit.

See the FAST Subsystem User's Guide, Order No. AU25 for an introduction to the use of edm.

SYNTAX AS A COMMAND:

emacs {-control_args} {paths}

FUNCTION: enters the Emacs text editor, which has a large repertoire of requests for editing and formatting text and programs. Emacs is a display-oriented editor designed for use on CRT terminals. Several modes of operation for special applications (e.g., RMAIL, PL/I, FORTRAN) are provided; the default mode entered is Fundamental major mode, whose requests are listed below.

ARGUMENTS:

paths
are pathname(s) of segments to be read in. Each is put into its own appropriately named buffer.

CONTROL ARGUMENTS:

-terminal_type STR, -ttp STR
specifies your terminal type to Emacs, where STR is any recognized editor terminal type or the pathname of a control segment to be loaded. The terminal type is set permanently; changing the Multics terminal type during a login session does not affect the type "remembered" by Emacs. If STR is not a recognized type, Emacs queries you after entry, providing a list of recognized types. This control argument must precede any other arguments and cannot be used with -reset or -query.

-reset
specifies that Emacs disregard the terminal type set by the -ttp control argument and set it in accord with the Multics terminal type instead. This control argument must precede any other arguments, and cannot be used with -ttp or -query.

-query
causes Emacs to query the user for a terminal type without checking the Multics terminal type first. The query response can be any recognized editor terminal type. This control argument must precede any other arguments and cannot be used with -ttp or -reset.

-line_speed N
indicates linespeed to obtain proper padding (for ARPANet users), where N is the output line baud rate in bits/second. This control argument can follow any one of the above three, but must precede paths and any of the remaining control arguments.

-no_startup, -ns
prevents use of the user's startup (start_up.emacs). This control argument must precede -mc, -ap, or paths, but must follow any of the other control arguments.

-macros path, -mc path
loads the segment, specified by path, as Lisp, so that features therein are available.

-apply function_name arg1 arg2 ... argi,
-ap function_name arg1 arg2 ... argi
evaluates (function_name 'arg1 'arg2 ... 'argi), where the args are arguments to the named Lisp function (e.g., an Emacs request). This is valuable for constructing abbrevs. This control argument must be the last argument.

NOTES: None of the terminal-type control arguments (-ttp, -reset, -query, -line_speed) are generally necessary; they are only used for solving various communications problems. However, if used, they must precede all other control arguments.

A detailed description of the Multics Emacs text editor can be found in the Emacs Text Editor Users' Guide, Order No. CH27. Information about extensions and instructions for writing them are provided in the Emacs Extension Writers' Guide, Order No. CJ52.

LIST OF FUNDAMENTAL MODE REQUESTS: The following is a list of Emacs Fundamental mode requests, alphabetized by the last character. Everything preceding the last character of each request is arranged in this suborder, under that last character: ^, ESC, ESC^, ^X, ^X^, ^Z, ^Z^. Extended requests are listed separately at the end.

#	rubout-char
ESC #	rubout-word
^X#	kill-backward-sentence
@	kill-to-beginning-of-line
^@	set-or-pop-the-mark
^Z^@	set-named-mark
CR	new-line
ESC CR	cret-and-indent-relative
^XCR	eval-multics-command-line

ESC	escape
ESC ESC	eval-lisp-line
^XESC	escape-dont-exit-minibuf
\	escape-char
ESC \	delete-white-sides
\177	rubout-char
ESC \177	rubout-word
^X\177	kill-backward-sentence
^X(	begin-macro-collection
^X)	end-macro-collection
^X*	show-last-or-current-macro
^X.	set-fill-prefix
ESC ;	indent-for-comment
^X;	set-comment-column
^Z;	kill-comment
^X=	linecounter
ESC %	query-replace
^_	help-on-tap
ESC _	underline-word
^Z_	remove-underlining-from-word
ESC /	regexp-search-command
ESC <	go-to-beginning-of-buffer
ESC >	go-to-end-of-buffer
ESC ?	describe-key
ESC [	beginning-of-paragraph
ESC]	end-of-paragraph
ESC ^	delete-line-indentation
ESC ~	unmodify-buffer
^X0	remove-window
^X1	expand-window-to-whole-screen
^X2	create-new-window-and-go-there
^X3	create-new-window-and-stay-there
^X4	select-another-window

^A	go-to-beginning-of-line
ESC A	backward-sentence
^B	backward-char
ESC B	backword-word
ESC ^B	balance-parens-backward
^XB	select-buffer
^X^B	list-buffers
^Z^B	edit-buffers
^C	re-execute-command
ESC C	capitalize-initial-word
^X^C	quit-the-editor
^D	delete-char
ESC D	delete-word
^XD	edit-dir
^E	go-to-end-of-line
ESC E	forward-sentence
^XE	execute-last-editor-macro
^X^E	comout-command
^F	forward-char
ESC F	forward-word
ESC ^F	balance-parens-forward
^XF	set-fill-column
^X^F	find-file
^Z^F	get-filename
^G	command-quit
ESC G	go-to-line-number
ESC ^G	ignore-prefix
^XG	get-variable
^X^G	ignore prefix
^ZG	go-to-named-mark
^Z^G	ignore-prefix
ESC H	mark-paragraph
^XH	mark-whole-buffer
ESC I	indent-relative
ESC ^I	indent-to-fill-prefix
^XI	insert-file
^J	noop
^K	kill-lines
ESC K	kill-to-end-of-sentence

^XK	kill-buffer
^L	redisplay-command
ESC L	lower-case-word
^X^L	lower-case-region
ESC M	skip-over-indentation
^XM	send-mail
^N	next-line-command
ESC N	down-comment-line
^O	open-space
ESC ^O	split-line
^XO	select-another-window
^X^O	delete-blank-lines
^P	prev-line-command
ESC P	prev-comment-line
^Q	quote-char
ESC Q	runoff-fill-paragraph
^XQ	macro-query
^R	reverse-string-search
ESC R	move-to-screen-edge
^XR	rmail
^X^R	read-file
^S	string-search
ESC S	center-line
^XS	global-print-command
^X^S	save-same-file
^T	twiddle-chars
^X^T	toggle-redisplay
^U	multiplier
ESC U	upper-case-word
^X^U	upper-case-region
^V	next-screen
ESC V	prev-screen
ESC ^V	page-other-window
^XV	view-lines
^Z^V	scroll-current-window
^W	wipe-region
ESC W	copy-region

ESC ^W	merge-last-kills-with-next
^XW	multi-word-search
^X^W	write-file
^Z^W	edit-windows
ESC X	extended-command
^XX	put-variable
^X^X	exchange-point-and-mark
^Y	yank
ESC Y	wipe-this-and-yank-previous
ESC ^Y	yank-minibuf
^Z^Z	signalquit

LIST OF EXTENDED REQUESTS:

ESC X	accept-msgs
ESC X	alm-mode
ESC X	apropos <string>
ESC X	describe <extended-request>
ESC X	edit-macros
ESC X	electric-pl1-mode
ESC X	filloff
ESC X	fillon
ESC X	fortran-mode
ESC X	fundamental-mode
ESC X	ldebug
ESC X	lisp-mode
ESC X	list-named-marks
ESC X	loadfile <path>
ESC X	loadlib <library>
ESC X	lvars
ESC X	make-wall-chart
ESC X	opt <option>
ESC X	pl1-mode
ESC X	replace
ESC X	reset-minibuffer-size <size>
ESC X	reset-screen-size <size>
ESC X	runoff-fill-region
ESC X	save-macro
ESC X	set-comment-prefix "string"
ESC X	set-compile-options "option string"
ESC X	set-compiler <compiler>
ESC X	set-key <keyname> <command-name>
ESC X	set-minibuffer-size <size>
ESC X	set-permanent-key <keyname> <command-name>
ESC X	set-screen-size <size>
ESC X	set-search-mode <search-mode>
ESC X	setab <abbrev1> <expansion1> <abbrevn> <expansion>

emacs

ESC X	show-macro <macro-name>
ESC X	speedtype
ESC X	speedtypeoff

SYNTAX AS A COMMAND:

encode path1A {path2A ... path1N path2N}

FUNCTION: enciphers a segment's contents according to a key that is not stored in the system. The enciphered segment has the same length as the original segment. The encode command provides additional security for data stored in a Multics segment.

ARGUMENTS:

path1
is the pathname of a segment to be enciphered. The star convention is allowed.

path2
is the pathname of an enciphered segment to be produced. If the last path2 is not specified, it is assumed to be the same as path1. The equal convention is allowed. This command always appends the code suffix to path2 to produce the name of the enciphered segment.

NOTES: The encode command requests an encipherment key (1-11 characters not including space, semicolon, or tab) from the terminal. Printing on the terminal is suppressed (the printer is turned off) while the key is typed. The command then requests that the key be typed again, to guard against the possibility of mistyping the key. If the two keys do not match, the key is requested twice again.

All segments specified in an invocation of encode are enciphered with the same key.

To reconstruct the original segment from the enciphered segment, see the decode command.

enter_abs_request (ear)

SYNTAX AS A COMMAND:

ear path {-control_args}

FUNCTION: allows a user to request that an absentee process be created. An absentee process executes commands from a segment and places the output in another segment. The user can delay the creation of the absentee process until a specified time.

ARGUMENTS:

path
specifies the pathname of the absentee control segment associated with this request. The absin suffix is assumed. The first argument to the command must be path.

CONTROL ARGUEMENTS:

-argument STR, -ag STR
indicates that the absentee control segment requires arguments. STR can be one or more arguments. All arguments following -ag on the command line are taken as arguments to the absentee control segment. Therefore -ag, if present, must be the last control argument to the enter_abs_request command.

-output_file path, -of path
specifies the pathname of the output segment. (See "Notes" below.)

-restart, -rt
specifies that the computation of this request should be started over again from the beginning if interrupted (for example, by a system crash). The default is to not restart the computation.

-limit N, -li N
places a limit on the CPU time used by the absentee process. The parameter N must be a positive decimal integer specifying the limit in seconds. The default limit is defined by the site for each queue. An upper limit is defined by the site for each queue on each shift. Jobs with limits exceeding the upper limit for the current shift are deferred to a shift with a higher limit.

-queue N, -q N
specifies that absentee queue N should contain the request to be entered, where N is an integer specifying the number of the

queue. The default queue is 3. There are four background queues with queue one having the highest priority. The highest numbered queue processed on each shift is determined by the site. For convenience in writing exec_coms and abbreviations, the word foreground or fg following the queue control argument performs the same function as the -foreground control argument.

-foreground, -fg
places the request in the foreground queue, rather than in one of the numbered background queues. Jobs in the foreground queue are treated, for load control and charging purposes, as interactive logins. That is, a foreground job is logged in if the user could have logged in interactively, and while logged in, it occupies a primary slot in the user's load control group. Also see the -secondary control argument below.

-secondary
logs in a foreground job as a secondary user (subject to preemption) if there are no primary slots available in the user's load control group. By default, a foreground job is only logged in if a primary process can be created for the user.

-time DT, -tm DT
delays creation of the absentee process until a specified date-time, where DT must be a character string acceptable to the convert_date_to_binary_ subroutine (described in the MPM Subroutines). If the DT string contains blanks, it must be enclosed in quotes.

-deferred_indefinitely, -dfi
does not run a job until the operator releases it.

-comment STR, -com STR
associates a comment with the request. If STR contains blanks or other command language characters, it must be enclosed in quotes. The comment is printed whenever the user or the operator lists the request. It can indicate to the operator the time or circumstances when a deferred job should be released such as when a specified reel of tape is delivered to the computer room.

-resource STR, -rsc STR
specifies resources given in STR, for example, one or more tape drives, and should not be started until they are available. The resource description must be enclosed in quotes if it contains blanks or other command language characters. For more information on resource description, see the reserve_resource command in this manual.

enter_abs_request (ear)

-brief, -bf
suppresses the message "ID: HHMMSS.f; N already requested."

-long_id, -lgid
prints the long form of the request identifier in the normal message:
ID: yymmddHHMMSS.ffffff; N already requested

-notify, -nt
notifies the user (by means of an interactive message sent to the user's mailbox) when the job is logged in, when it is logged out, or when it is deferred for any reason other than the user's request. The latter might occur because of the unavailability of resources or a time limit higher than the maximum for the shift.

-proxy User_id
enters the request on behalf of the specified user. An absentee process of that User_id will be logged in to run the job. Use of this control argument is controlled by the system administrator by means of an access control segment.

-sender STR
specifies that only requests from sender STR should be entered. In most cases, the sender is an RJE station identifier.

NOTES: The principal difference between an absentee process and an interactive one is that in an absentee process the I/O switch user_input instead of being attached to a terminal is attached to an absentee control segment containing commands and control lines, and the I/O switch user_output instead of being attached to a terminal is attached to an absentee output segment. The absentee control segment has the same syntax as an exec_com segment. (See exec_com in this manual.)

An error message, unless it says otherwise, indicates that the request has not been submitted.

If the pathname of the output segment is not specified, the output of the absentee process is directed to a segment whose pathname is the same as the absentee control segment, except that it has a suffix of absout instead of absin. If the absout suffix is omitted from the output segment pathname, the suffix is assumed. The named output segment may or may not already exist.

If the absout segment exists, the absentee user (Person_id.Project_id.m or, in the case of a proxy request, Person_id.Project_id.p) must have write access to the segment. If the absout segment does not exist, the absentee user requires append permission to the directory in which it is to be created.

The command checks for the existence of the absentee input segment and rejects a request for an absentee process if it is not present.

The effect of specifying the -time control argument is as if the enter_abs_request command were issued at the deferred time.

See also the descriptions of the list_abs_requests and cancel_abs_request commands for information on displaying and cancelling outstanding absentee requests.

If an absentee job cannot be run or if it terminates abnormally, the system sends an interactive user message to the submitter's mailbox, whether or not the -notify control argument is given.

EXAMPLES: Suppose that a user wants to request an offline compilation. A control segment can be constructed called absentee_pl1.absin containing:

```
change_wdir current_dir
pl1 x -table -map
dprint -delete x.list
logout
```

enter_abs_request (ear)

The command line:

```
! ear absentee_pl1
```

creates an absentee process (some time in the future) that does the following:

1. sets the working directory to the directory named current_dir, which is inferior to the user's normal home directory.
2. compiles a PL/I program named x.pl1 with two control arguments.
3. dprints one copy of the listing segment and then deletes it.
4. logs out.

The output of these tasks appear in the directory containing absentee_pl1.absin in a segment called absentee_pl1.absout.

Suppose that an absentee control segment, trans.absin, contains the following:

```
change_wdir &1
&2 &3 -map &4
dprint -delete &3.list
&goto &2.b
&label pl1.b
&3
&label fortran.b
logout
```

The command line:

```
! ear trans -li 300 -rt -ag work pl1 x -table
```

requests a restartable absentee process in queue 3 having a CPU limit of 300 seconds, that does the following:

1. sets the working directory to the directory named work, which is inferior to the normal home directory.

2. compiles a PL/I program x.pl1 in that directory and produces a listing segment containing a map and with an object segment containing a symbol table.

3. issues a dprint request for the listing segment.

4. executes the program x just compiled in the absentee process.

5. logs out.

The command line:

```
! ear trans -rt -tm "Monday 2300.00" -q 2 -ag comp fortran yz
```

creates a request for a restartable absentee process in queue 2 at the first occurrence of Monday, 11 P.M., that does the following:

1. sets the working directory to the directory named comp, which is inferior to the home directory.

2. compiles a FORTRAN program named yz.fortran and produces a listing segment.

3. issues a dprint request for the listing segment.

4. logs out.

All of the commands used in the above examples are described in this document under the name of the particular command.

enter_retrieval_request (err)

SYNTAX AS A COMMAND:

err path {-control_args}

FUNCTION: queues volume retrieval requests for specific segments, directories, multisegment files, and subtrees.

ARGUMENTS:

path
is the pathname of a segment, directory, or node of a subtree. The star convention is not allowed.

CONTROL ARGUMENTS:

-multisegment file, -msf
specfies that the object named in path is a multisegment file and that all of its components are to be recovered.

-subtree, -subt
specifies that the subtree inferior to the directory specified in path as well as the directory is to be retrieved. If a subtree is found intact after a directory is recovered, then no further action is taken, unless a time interval has been specified. See "Notes" for more information. The default is not to retrieve subtrees.

-queue N, -q N
queues requests in priority queue N. The default is queue 3.

-to DT
specifies that the search for path and all inferior branches, if specified, proceeds from time DT backwards. Thus, objects dumped later than time DT are not recovered. Time DT must be acceptable to the convert_date_to_binary_ subroutine. (See MPM Subroutines for a complete description of convert_date_to_binary_.) If this control argument is not specified, time DT is assumed to be the start of the retrieval operation.

-from DT, -fm DT
specifies that the search for path and all inferior branches, if specified, stops at time DT. Thus, objects dumped before time DT are not recovered. Time DT must be acceptable to the convert_date_to_binary_ subroutine. (See MPM Subroutines for a complete description of convert_date_to_binary_.) If the

enter_retrieval_request (err)

control argument is not specified, all valid dump volumes are searched.

-new_path newpath
specifies that if the requestor has the correct access to retrieve the segment specified in path above (which must already exist) and the correct access to create a segment with the pathname newpath, then the object described/identified by path is retrieved into newpath.

-notify, -nt
specifies that the user is to be notified by online mail of the success or failure of the request. The default is to not notify the user.

-previous, -prev
specifies that the object to be retrieved is the one dumped prior to the object presently online. The default is to always retrieve the most recent copy. By specifying this control argument, the requestor can retrieve successively earlier copies of an object.

ACCESS REQUIRED: The user must have write access or modify permission to an object in order to retrieve it. If an object has been deleted, then append permission on the containing directory is also required.

NOTES: In certain cases where a directory is damaged, the inferior subtree may be unavailable until the directory is recovered. When a directory is recovered, and the subtree control argument is specified, a check is made to see if the subtree is available, and if so, retrieval is assumed complete.

Retrieval requests of objects for which the online copy is more recent than the dump copy are refused, unless the -previous, -from, or -to control arguments are used.

The pathnames of the segments and directories to be retrieved need not be specified as a set of primary names. Any set of valid entrynames is acceptable.

SYNTAX AS A COMMAND:

```
entries star_names {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[entries star_names {-control_arg}]
```

FUNCTION: returns the entrynames or absolute pathnames of segments, directories, multisegment files, and links that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per entry is returned; i.e., if an entry has more than one name that matches a star_name, only the first match found is returned.

Since each entryname (or pathname) returned by entries is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

See the description of directory, directories, and entry in this manual.

EXAMPLES:

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w    0  empty_seg
  re     1  test
  r w    1  test.list
  r w    1  test.pl1
  re     1  prog
  r w    1  prog.list
  r w    1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w  513  prog.output
  r w  257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                >udd>Apple>Jones>temp_seg_2
  prog.temp1                >udd>Apple>Jones>temp_seg_1
  junk                      >udd>Apple>Jones>empty_seg

! string [entries prog.*]
  prog.pl1 prog.list prog.data prog.output prog.temp1
     prog.temp2
! string [entries *.pl1]
  prog.pl1 test.pl1
```

SYNTAX AS A COMMAND:

```
entry path
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[entry path]
```

FUNCTION: returns the entryname portion of path, after it has been expanded into an absolute pathname.

ARGUMENTS:

path
is the pathname whose entryname portion is to be returned.

NOTES: See the description of directory in this manual.

EXAMPLES:

In the directory >udd>m>Jones:

```
! string [entry >udd>m>Jones>foo]
foo
! string [entry [wd]]
Jones
```

SYNTAX AS A COMMAND:

```
equal strA strB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[equal strA strB]
```

FUNCTION: returns true if strA is equal to strB; otherwise it returns false.

ARGUMENTS:

strA, strB
are character strings to be compared.

NOTES: The strings are compared character by character according to their ASCII code value (i.e., if the first character in each string has the same ASCII code value, compare the second character; if their values are identical, compare the third character; etc.). Strings of unequal length are compared by padding with blanks.

EXAMPLES:

```
! string [equal Ab ab]
  false

! string [equal this this]
  true
```

If the current working directory is the default working directory:

```
! equal [wd] [dwd]
  true
```

SYNTAX AS A COMMAND:

enm path equal_path

SYNTAX AS AN ACTIVE FUNCTION:

[enm path equal_path]

FUNCTION: returns an entryname or absolute pathname, constructed by applying the equal convention to the specified arguments.

ARGUMENTS:

path
is any pathname. The equal convention is applied to its entryname portion.

equal_path
is a pathname whose entryname portion is an equal name. If equal_path is only any entryname (i.e., contains no < or > characters) then the result is an entryname. Otherwise, the result is an absolute pathname.

NOTES: This active function provides a means of applying equal names within abbreviations and exec_com segments.

For a complete description of the equal convention see "Equal Names" in the MPM Reference Guide.

EXAMPLES: The following abbreviation creates a link and adds names to it in a single command line. Assume that linkn is an abbreviation for:

```
do "link &1 &2; if [exists argument &3] -then
  ""an [directory &2]>[equal_name &1 &2] &f3"""
```

then the single command line:

```
! linkn generate_report.ec old_=.= old_grpt.ec old_grpt
```

is equivalent to the following command lines:

```
! link generate_report.ec old_=.=
! add_name old_generate_report.ec old_grpt.ec old_grpt
```

SYNTAX AS A COMMAND:

ec path {optional_args}

FUNCTION: used to execute a sequence of command lines contained in a segment. This command allows the user to construct command sequences that are invoked frequently without retyping the commands each time. In addition, control strings can be used to substitute argument values into the executed text, manage I/O switches, and execute portions of the text conditionally.

ARGUMENTS:

path
is pathname of a segment containing commands to be executed and control statements to be interpreted. The entryname of the segment must have the ec suffix, although the suffix can be omitted in the command invocation. If an entryname only is specified, i.e., one containing no < or > characters, the exec_com search list is used to locate the segment. See "Notes on Search List" section below.

optional_args
are character strings to be substituted for special strings in the exec_com segment. (See "Notes on Argument Substitution" below.)

NOTES ON INPUT SEGMENT: The exec_com segment should contain only command lines, input lines, and control statements. Normally it is created using a text editor, such as qedx. The exec_com command can be used in conjunction with the abbrev command to form abbreviations for command sequences that are used frequently.

When the ampersand character (&) appears in the exec_com segment, it is interpreted as a special character. It is used to denote a string used for argument substitution and to signify the start of a control statement.

NOTES ON ARGUMENT SUBSTITUTION: Strings of the form &i in the exec_com segment are interpreted as dummy arguments and are replaced by the corresponding arguments to the exec_com

command. For instance, optional_arg1 is substituted for the string &1 and optional_arg10 is substituted for &10.

The strings &qi, &ri, &fi, &qfi, and &rfi also indicate argument substitution. The string &qi is replaced by the ith argument to the exec_com command with quotes doubled. The string &ri is replaced by the ith argument, requoted. Refer to do in this manual for a description of quote doubling and requoting and for examples of the use of &qi, &ri, &fi, &qfi, and &rfi. The string &fi is replaced by a string of the ith through last arguments to exec_com, separated by blanks. Likewise, &qfi is replaced by a string of the ith through last arguments with quotes doubled and &rfi is replaced by a string of the ith through last arguments, requoted.

The string &n is replaced by the number of arguments to the exec_com command. The string &f&n, therefore, is replaced by the last argument to exec_com. The string &ec_name is replaced by the entryname portion of the exec_com pathname without the ec suffix. The string &ec_dir is replaced by the directory name portion of the exec_com pathname.

Argument substitution can take place in command lines, input lines or in control statements, since the replacement of arguments is done before the check for a control statement.

NOTES ON CONTROL STATEMENTS: Control statements permit more variety and control in the execution of the command sequences. Currently the control statements are: &label, &goto, &attach, &detach, &input_line, &command_line, &ready, &print, &quit, &if, &then, and &else.

Control statements generally must start at the beginning of a line with no leading blanks. Two exceptions to this rule are the &then statement which can follow an &if clause, and the &else statement, which can follow a &then clause. Any control statement other than &label, &if, &then, and &else is allowed to follow the control words &then and &else.

&label and &goto

These statements permit the transfer of control within an exec_com segment.

&label location
identifies the place to which a goto control statement transfers control. The location is any string of 32 or fewer characters, unique within the exec_com segment.

&goto location
causes control to be transferred to the place in the exec_com segment specified by the label location. Execution then continues at the line immediately following the label.

&attach, &detach, and &input_line

&attach
causes the user_input I/O switch to be attached to the exec_com segment. This means that if this control statement is executed, all input read by subsequent commands is taken from the segment rather than from the previous source of data to which the user_input I/O switch was attached.

&detach
causes the user_input I/O switch to be reverted to its original value. By default, the user_input I/O switch is left attached to its original source.

&input_line on
causes input lines returned when using the attach feature to be written on the user_output I/O switch. This is the default.

&input_line off
causes input lines to not be written out.

&command_line, &ready, and &print

These statements allow the control of the user_output I/O switch. They are useful as tools in observing the progress of the exec_com execution and in printing messages.

&command_line on
causes subsequent command lines to be written on the user_output I/O switch before they are executed. This is the default.

&command_line off
causes subsequent command lines to not be written out.

&ready on
causes the invocation of the user's ready procedure after the execution of each command line.

&ready off
causes the user's ready procedure to not be invoked. This is the default.

&print char_string
causes the character string following &print to be written out on the user_output I/O switch. The character ^ is treated as a special character in a print statement. The following is a list of strings that can appear and the characters that replace them:

string	replacement
^/ or ^n/	newline character
^| or ^n|	form feed (new page)
^- or ^n-	horizontal tab
^^	^

where n expresses the number of special characters to be written out.

No other characters should appear following the ^ character in the print statement.

&quit

This statement causes the current invocation of exec_com to return to its caller and not to execute subsequent command lines.

&if, &then, and &else

These statements provide the ability to have command lines, input lines, and control statements interpreted conditionally.

The format of these control statements is:

```
&if [ACTIVE_FUNCTION {arg1} ... {argn}]
&then THEN_CLAUSE
&else ELSE_CLAUSE
```

The active function reference in an &if control statement is evaluated. If the value of the active function is the string true, THEN_CLAUSE is executed. If the value is false, ELSE_CLAUSE is executed.

&if [ACTIVE_FUNCTION {arg1} ... {argn}]
This statement must start at the beginning of a line. The active function is any active function (user-provided or system-supplied) that returns as its value a string with the value true or false. The arguments to the active function can themselves be active functions. (Nesting of active functions is permitted.) The active function and its optional arguments, enclosed in brackets, must be on the same line as the &if string.

&then THEN_CLAUSE
This statement must immediately follow the &if statement; it can appear on the same line or on the following line. THEN_CLAUSE is an exec_com statement, and can include a command line, an input line, the null statement and most control statements. The &label, &if, &then, and &else control statements are not allowed. (Nesting of &if statements is not permitted.) THEN_CLAUSE must be on the same line as &then.

&else ELSE_CLAUSE
This statement is optional. When it appears, it must immediately follow the &then statement; it can appear on the same line or on the following line. ELSE_CLAUSE is an exec_com statement and can include a command line, an input line, the null statement and most control statements. The &label, &if, &then and &else control statements are not allowed. ELSE_CLAUSE must be on the same line as &else.

NOTES ON SEARCH LIST: The exec_com command uses the exec_com search list that has the synonym ec. Type:

```
! psp ec
```

to see what the current exec_com search list is. The default exec_com search list is the working directory. For more information on the search facility, see the description of the add_search_paths command in this manual.

NOTES ON HANDLING CONDITIONS: The on command and active function can be used to handle conditions raised during the execution of an exec_com. To handle command_error when executing the copy command, for example, an exec_com can say:

```
&if [on command_error "" -bf copy PROJ_DIR>&1 MY_DIR>=]
  &then &goto copy_failed
an MY_DIR>&1 &1.[date]
...
&label copy_failed
&print PROJ_DIR>&1 not copied
...
```

The -bf control argument suppresses a message printed by on when the condition is raised.

The discard_output command can be used to suppress output from the command whose success is being tested, for example:

```
&if [on command_error "" -bf dco -osw error_output -osw
  user_output archive tb source &1.pl1]
&then &goto no_component
&print &1.pl1 in source.archive
...
&label no_component
&print &1.pl1 not found in source.archive
...
```

The on command can be used to execute another exec_com, or a recursive entry point in the current one, with a handler in effect. For example:

```
on any_other "ec handler" ec test_ms
...
&quit
&label handler
tmr mail mbsa mailbox_
in >sss>mail
&quit
&label test_ms
tmr mail mbsa mailbox_
in MS>mailbox_
MS>mbsa test.mbx adros
MS>mail test
&quit
```

For more information, see the description of on in this manual.

NOTES ON HANDLING QUESTIONS: The answer command can be used to supply preset answers to questions asked by commands invoked in an exec_com. (It is not recommended that answers be supplied on successive lines of the exec_com with &attach on.) The following exec_com prints only the first three sections of an info segment by answering "yes" twice and then "no":

```
answer yes -times 2 -then no help &1
&quit
```

The following example prints the first three sections of an info segment, then prints the next three only if the user answers yes:

```
answer yes -times 2 -then -query -then yes -times 2
   -then no help &1
&quit
```

For more information, see the description of the answer command in this manual.

NOTES: If a line begins with the & character but is not one of the current control statements, the entire line is ignored. This is one way of including comments in the exec_com segment. The user is cautioned to leave a blank immediately following the & to ensure compatibility with control requests to be added to exec_com in the future.

The segment executed by exec_com can contain calls to exec_com. The user must exercise caution when invoking this feature in conjunction with the &attach feature. When exec_com is called from an exec_com using this feature, the input read by commands in the second exec_com is read from the first exec_com segment. Generally, if the &attach feature is used, all calls to exec_com should be preceded by &detach control statements.

Several exec_coms can be combined into one segment, by using the dummy argument &ec_name together with the &label and &goto statements. If exec_coms are grouped together, the exec_com segment should have all the names (concatenated with an ec suffix) on its storage system entry that can replace &ec_name.

Examples

EXAMPLES: Assume that the segment a.ec in the user's working directory contains:

```
pl1 &1 -table -list
dprint -delete &1.list
&quit
```

The command line:

```
! exec_com a foo
```

causes the following commands to be executed:

```
pl1 foo -table -list
dprint -delete foo.list
```

Assume that the segment b.ec in the user's working directory has an additional name a.ec and contains:

```
&goto &ec_name
&
&label b
print &1 1 99
&quit
&
&label a
pl1 &1 -table -list
dprint -delete &1.list
&quit
```

The command line:

```
! exec_com b my_file
```

causes the following command to be executed:

```
print my_file 1 99
```

The command line:

```
!  exec_com a foo
```

causes the following commands to be executed:

```
pl1 foo -table -list
dprint -delete foo.list
```

Assume that the segment d.ec in the user's working directory contains the following:

```
&if [exists segment &1.pl1] &then
&else &goto not_found
pl1 &1 -table -list
dprint -delete &1.list
&quit
&label not_found
&print &1.pl1 not found
&quit
```

If the segment foo.pl1 exists, the command line:

```
!  exec_com d foo
```

causes the following commands to be executed:

```
pl1 foo -table -list
dprint -delete foo.list
```

If the segment foo.pl1 does not exist, the command line:

```
!  exec_com d foo
```

outputs the following:

```
foo.pl1 not found
```

Assume that the segment test.ec in the user's working directory contains:

```
&print begin &ec_name exec_com
&command_line off
create &1.pl1
&attach
edm &1.pl1
i &1: proc;
&input_line off
i end &1;
w
q
&detach
&goto &2
&label compile
pl1 &1
&label nocompile
&print end &ec_name &1 &2 exec_com
&quit
```

The command line:

```
! exec_com test x compile
```

produces the following output:

```
begin test exec_com
Edit.
i x: proc;

PL/I
end test x compile exec_com
```

SYNTAX AS A COMMAND:

```
   exists argument {str_args}
or:
   exists key star_name {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
   [exists argument {str_args}]
or:
   [exists key star_name {-control_arg}]
```

FUNCTION: checks for the existence of various types of items depending on the value of the first argument (key).

ARGUMENTS:

argument
   is the key "argument" described below in "List of Keys".

key
   is any key as described below in "List of Keys".

str_args
   are character string arguments.

CONTROL ARGUMENTS:

-chase
   specifies that any keyword that looks for branch entries chase links and look at the link targets.

LIST OF KEYS:

argument
   true if any str_args are specified; otherwise false.

branch
   true if any branches (segments, multisegment files, or directories) with a pathname matching star_name exist; otherwise false.

directory, dir
   true if any directories with a pathname matching star_name exist; otherwise false.

entry
true if any entries (segments, multisegment files, directories, or links) with a pathname matching star_name exist; otherwise false.

file
true if any segments or multisegment files with a pathname matching star_name exist; otherwise false.

link
true if any links with a pathname matching star_name exist; otherwise false.

master_directory, mdir
true if any master directories with a pathname matching star_name exist; otherwise false.

msf
true if any multisegment files with a pathname matching star_name exist; otherwise false.

nonbranch
true if any links with a pathname matching star_name exist; otherwise false.

nonfile
true if any links or directories with a pathname matching star_name exist; otherwise false.

nonlink
true if any directories, segments, or multisegment files with a pathname matching star_name exist; otherwise false.

nonmaster_directory, nmdir
true if any directories which are not master directories with a pathname matching star_name exist; otherwise false.

nonmsf
true if any directories, segments, or links with a pathname matching star_name exist; otherwise false.

non_null_link, nonnull_link, nnlink
true if any links with a pathname matching star_name exist and point to an existing segment, directory, or multisegment file; otherwise false.

nonsegment, nonseg
true if any links, directories, or multisegment files with a pathname matching star_name exist; otherwise false.

nonzero_file, nzfile
   true if any nonzero length segments or multisegment files with a pathname matching star_name exist; otherwise false.

nonzero_msf, nzmsf
   true if any nonzero length multisegment files with a pathname matching star_name exist; otherwise false.

nonzero_segment, nzseg
   true if any nonzero length segments with pathname matching star_name exist; otherwise false.

null_link
   true if any links with a pathname matching star_name exist and point to nonexistent entries; otherwise false.

segment, seg
   true if any segments with a pathname matching star_name exist; otherwise false.

zero_segment, zseg
   true if any zero length segments with a pathname matching star_name exist; otherwise false.

expand_cobol_source (ecs)

SYNTAX AS A COMMAND:

ecs path {-control_args}

FUNCTION: translates a segment containing the text of a standard format COBOL source program containing COPY and REPLACE statements to equivalent source programs not containing these statements.

ARGUMENTS:

path
is the pathname of the input segment to be modified. If path does not have a suffix of cobol, one is assumed. However, the cobol suffix must be the last component of the name of the input segment. The translated segment created by ecs is the first component of the entryname with the suffix ex.cobol added.

CONTROL ARGUMENTS:

-format, -fmt
converts pseudo free form COBOL source programs to the standard fixed format COBOL source programs processed by the COBOL compiler.

-upper_case, -uc
produces a formatted output file entirely in uppercase except for the contents of nonnumeric literals, which are left exactly as specified in the input file. If this control argument is not specified, the file is produced as typed.

-lower_case -lc
produces a formatted output file entirely in lowercase except for the contents of nonnumeric literals, which are left exactly as specified in the input file. If this control argument is not specified, the file is produced as typed.

NOTES: If the -fmt, -lc, or -uc control arguments are used, the expand_cobol_source command assumes that the input file is in free form (as would be typically typed in from a terminal) and attempts to reformat each line into the standard COBOL reference format described in the Multics COBOL Users' Guide, Order No. AS43. Statements in a COBOL source program generally begin in area B (column 12 and beyond). However, certain entries must begin in area A (column 8 through 11).

These are COBOL-defined division names, section names, paragraph names, level indicators, and certain level numbers, as well as user-defined section names and paragraph names. Additionally, certain characters have special meaning when appearing in the indicator area (column 7), such as the asterisk, slash, hyphen, and letter "d".

The expand_cobol_source command recognizes all COBOL-defined names that are required to appear in area A and reformats lines containing them to guarantee that they do so. User-defined section names are recognized by the appearance of the word "section" on the line while words beginning the line and followed immediately by a period are assumed to be user-defined paragraph names. Source lines containing either of these are reformatted similarly to lines containing COBOL-defined sections and paragraphs. Lines beginning with level numbers 01, 66, 77, 88 are reformatted to begin in area A (at column 8) as required in standard American National Standard (ANS) COBOL. Lines beginning with level numbers 02 through 49 are indented a number of spaces identical to the numeric value of the level number plus seven (e.g., 02 begins at column 9, 05 at column 12).

Certain characters force special interpretation when they begin a free form source line. The slash (/) and asterisk (*) when used in this way denote a comment line with or without page eject, respectively; the hyphen (-) denotes a continuation line. Such lines are reformatted so that these special characters appear in the indicator area followed by the rest of the line. Additionally, for continuation lines, the remainder of the line following the hyphen is shifted to begin in area B as COBOL prohibits use of area A in this case.

Debugging lines are probably of little interest for Multics COBOL users due to the powerful symbolic debugging facilities available on an interactive basis, but they can be specified in free form source by beginning the line with "d*". In rare instances, in which a user-defined section or paragraph name is specified in a way not contextually recognizable by the expand_cobol_source command, the user can force reformatting beginning in area A by beginning the line with "a*" (or "da*" in the case of debugging lines).

All other source lines (i.e., those not beginning with special character(s) and not containing entries required to begin in area A) are reformatted by insertion of eleven blanks forcing

expand_cobol_source (ecs)

commencement in area B. Any indentation already existing in the free form file is thereby maintained relative to column 12.

The expand_cobol_source command also converts all horizontal tab characters (ASCII code 011) not contained in nonnumeric literals to spaces. The number of spaces is determined by subtracting the position of the tab character on the source line modulo 10 from 10. In this way, the user can input the source program using the tab character as a formatting tool, yet avoid the fact that this is not part of the standard COBOL character set.

The COBOL source program output is acceptable to any ANS compiler with regard to reference format. (Actually, Multics COBOL relaxes many of these format requirements. However, it is usually desirable to eliminate the warnings and observations issued when such ANS rules are violated.) For transportability purposes, the output file can be created entirely in uppercase or lowercase (with the contents of nonnumeric literals left as is) by use of the -upper_case and -lower_case control arguments. If neither is specified, the case of all words remains the same as in the input file. Notice, all COBOL-defined names and characters with special meaning are recognized regardless of case, i.e., they may be all in uppercase, all in lowercase, or in mixed case.

For those users wishing to keep source files in free form, identical function described above is available on a per use basis via the -format control argument of the cobol command. Refer to the description of the cobol command for further information.

SYNTAX AS A COMMAND:

fast

FUNCTION: causes the user to enter the FAST subsystem.

NOTES: For a description of the commands available under FAST, see the Multics FAST Subsystem Users' Guide, Order No. AU25.

To exit the subsystem and return to Multics system command level the user should type quit (q).

file_output (fo)

SYNTAX AS A COMMAND:

```
fo {path} {-control_args}
ro {-control_args}
so target_sw {-control_args}
to {-control_args}
```

FUNCTION: The file_output (fo) command directs I/O output switches to a specified file. The terminal_output (to) command directs I/O output switches to the user's terminal. The syn_output (so) command directs output I/O switches to another already open I/O switch. The effects of the first three commands can be stacked. The revert_output (ro) command reverts the effect of these other commands, i.e., releases the most recent, preceding command.

ARGUMENTS:

path
is the pathname of a segment. If the segment does not exist, it is created. If path is not specified, the segment output_file in the working directory is assumed.

target_sw
is the name of an open I/O switch to which output is to be redirected. It must be open for stream_output, stream_input_output, or IOS (the older version of the I/O system) compatibility.

CONTROL ARGUMENTS:

-source_switch STR, -ssw STR
specifies the name of an I/O switch to be redirected. The default is user_output.

-all, -a
reverts all file_output, terminal_output, and syn_output attachments for specified I/O switches or for all switches if none are specified. This control argument is applicable to the revert_output command only.

-extend
extends the output file (default).

-truncate, -tc
truncates an existing output file for file_output. The default is to extend the output file.

NOTES: Each command invocation of file_output, terminal_output, or syn_output stacks up another attachment for each of the specified switches. The revert_output command pops and restores one attachment from the stack. It does not revert attachments made, for example, by the io_call command.

The command line:

```
! revert_output -ssw STR
```

reverts the latest attachment by one of the following command lines:

```
file_output -ssw STR
terminal_output -ssw STR
syn_output target -ssw STR
```

To avoid getting ready messages in the output file, the file_output (or syn_output) and revert_output commands should appear on the same command line.

EXAMPLES: The command line:

```
! fo text.cpa;cpa text.old text.new;ro;dp text.cpa
```

makes a comparison of two text segments names text.old and text.new, places the results of that comparison in the output file named text.cpa, and dprints the file text.cpa on a remote printer.

The sequence of commands within an exec_com segment:

```
fo segs_and_links
ls -seg
to
ls -directory
ro
ls -link
ro
```

lists segments and links in the output file named segs_and_links and lists directories on the terminal.

file_output (fo)

The sequence of lines within an exec_com segment:

```
&if [equal &1 tape] &then io attach s1 tape_mult_ &2;
   io open s1 so
&if [equal &1 file] &then io attach s1 vfile_ &2;
   io open s1 so
&if [equal &1 tty] &then io attach s1 syn_ user_i/o
   syn_output s1;
so s1; ws -wd "list -all";ro
&if [not [equal &1 tty]] &then io close s1
io detach s1
```

outputs a listing of all segments in a subtree to a file, a tape, or the terminal as specified by the first exec_com argument.

SYNTAX AS A COMMAND:

```
files star_names {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[files star_names {-control_arg}]
```

FUNCTION: returns the entrynames or absolute pathnames of segments and multisegment files that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per file is returned; i.e., if a file has more than one name that matches a star_name, only the first match found is returned.

Since each entryname (or pathname) returned by files is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

EXAMPLES:

```
! pwd
>udd>Apple>Jones
! ls -a

   Segments = 7, Lengths = 6.

   r w    0  empty_seg
   re     1  test
   r w    1  test.list
```

```
r w     1  test.pl1
re      1  prog
r w     1  prog.list
r w     1  prog.pl1

Multisegment-files = 2, Lengths = 770.

r w   513  prog.output
r w   257  prog.data

Directories = 2.

sma  prog_stuff
sma  documents

Links = 3.

prog.temp2                    >udd>Apple>Jones>temp_seg_2
prog.temp1                    >udd>Apple>Jones>temp_seg_1
junk                          >udd>Apple>Jones>empty_seg

! string [files **]
prog.pl1 prog.list prog.data prog test.pl1 test.list test
   prog.output empty_seg

! string [files *.pl1]
prog.pl1 test.pl1
```

SYNTAX AS A COMMAND:

```
floor num
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[floor num]
```

FUNCTION: returns the largest decimal integer less than or equal to its argument.

EXAMPLES:

```
! string [floor 4.7]
4.0
! string [floor -4.7]
-5.0
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[fl control_string {args}]
```

FUNCTION: returns a single, quoted character string that is formatted from an ioa_ control string and other optional arguments.

ARGUMENTS:

control_string
is an ioa_ control string that is used to format the return value of the active function. The following ioa_ control codes are allowed (for detailed descriptions, see ioa_ in this manual):

Control		Acceptable Arguments
^a	^Na	any character string
^d	^Nd	a character representation of a number, including optional exponent (e.g., 315.44 or .2789e+2 or 1101b)
^i	^Ni	same as ^d
^f	^Nf	same as ^d
^.df	^N.df	same as ^d
^e	^Ne	same as ^d
^o	^No	same as ^d
^[... ^]		"true", "false", or an integer character string
^(	^N(	an integer character string
^)		an integer character string
^;		
^s	^Ns	

In addition, any of the following carriage movement controls may be used:

```
     ^N/ ^N| ^N- ^Nx ^N^ ^R ^B
or
      ^/ ^| ^- ^x ^^
```

where N is an integer count or a "v". When "v" is given, an integer character string from the args is used for count. For a complete description of these control strings see the description of the ioa_ subroutine in the MPM Subroutines.

args
are character strings substituted in the formatted return value, according to the ioa_ control string.

NOTES: If no optional arguments are given, the value returned depends on the ioa_ control string that was specified.

See the description of the ioa_ subroutine in the MPM Subroutines.

EXAMPLES:

In an exec_com segment, the lines:

```
&if [query [fl "^a copies already exist.^/Do you want to build
   another?"] &2]
&then ec build_new_data [plus 1 &2]
```

might be expanded when &2 is 3 to:

```
3 copies already exist.
Do you want to build another?
```

The lines:

```
! string [fl "Insurance option: <2 spaces>
             ^[no fault^;regular^]" [query "No Fault?"]]
```

prints the following if the user answers "yes" to the query:

```
Insurance option:  no fault
```

NOTES: The Multics system currently supports two FORTRAN compilers (see the new_fortran and old_fortran commands). The name fortran (ft) is added to either the new_fortran or old_fortran command, depending on which compiler the site wants to use as their "standard" FORTRAN compiler. By default, the names fortran and ft are associated with the new FORTRAN compiler.

SYNTAX AS A COMMAND:

fa paths {ft_args} {dp_args} {-control_args}

FUNCTION: submits an absentee request to perform FORTRAN compilations on the site's standard FORTRAN compiler.

ARGUMENTS:

paths
are pathnames of segments to be compiled.

ft_args
are one or more control arguments accepted by the fortran command.

dp_args
are one or more control arguments (except -delete) accepted by the dprint command.

CONTROL ARGUMENTS:

-queue N, -q N
specifies in which priority queue the request is to be placed ($N \leq 3$). The default queue is 3; the listing segment is also dprinted in queue N.

-hold
specifies that fortran_abs should not dprint or delete the listing segment.

-limit N, -li N
places a limit on the CPU time used by the absentee process. The parameter N must be a positive decimal integer specifying the limit in seconds. The default limit is defined by the site for each queue. An upper limit is defined by the site for each queue on each shift. Jobs with limits exceeding the upper limit for the current shift are deferred to a shift with a higher limit.

-output_file path, -of path
specifies that absentee output is to go to the segment whose pathname is path.

fortran_abs (fa)

NOTES: The absentee process for which fortran_abs submits a request compiles the segments named and dprints and deletes the listing segment. If the -output_file control argument is not specified, an output segment, path.absout, is created in the user's working directory (if more than one path is specified, only the first is used). If none of the segments to be compiled can be found, no absentee request is submitted.

Control arguments and segment pathnames can be mixed freely and can appear anywhere on the fortran_abs command line after the command. All control arguments apply to all segment pathnames. If an unrecognizable control argument is given, the absentee request is not submitted.

Unpredictable results can occur if two absentee requests are submitted that could simultaneously attempt to compile the same segment or write into the same absout segment.

When doing several compilations, it is more efficient to give several pathnames in one command rather than several commands. With one command, only one process is set up. Thus the dynamic intersegment links that need to be snapped when setting up a process and when invoking the compiler need be snapped only once.

FUNCTION: invokes the GCOS environment simulator to run a single GCOS job, immediately, in the user's process.

NOTES: Related facilities include the GCOS daemon, which provides batch processing for GCOS jobs under Multics and the following commands, which may be used to manipulate GCOS format files that reside in the Multics storage system:

gcos_sysprint, gsp
gcos_syspunch, gspn
gcos_card_utility, gcu

These commands and the gcos command are fully described in the Multics GCOS Environment Simulator manual, Order No. AN05.

SYNTAX AS A COMMAND:

gr {-control_args}

SYNTAX AS AN ACTIVE FUNCTION:

[gr {-control_args}]

FUNCTION: prints a ready message containing specified values in a specified format.

LIST OF CONTROL ARGUMENTS BY FUNCTION:

Prefix control arguments
-string
-control

Format control arguments
-inc_vcpu
-total_vcpu
-total_mem_units
-inc_cost
-total_cost
-inc_pf
-total_pf
-level
-date
-date_time
-day
-day_name
-hour
-minute
-month
-time
-year
-zone

Operation control arguments
-set
-revert
-reset
-call

LIST OF PREFIX CONTROL ARGUMENTS: These arguments must occur prior to any of the format control arguments described below.

The two prefix arguments allow the user to override the default formats for the contents of the ready message. They are:

-string
allows the user to specify the character string at the beginning of the ready message. The argument following this control argument is used instead of "r " at the beginning of the ready message. Since it is put into the ioa_ control string, "^/", "^R", and "^B" can be used to cause new lines, red ribbon shifts, and black ribbon shifts, respectively.

-control
allows the user to specify the entire ioa_ control string used to format the ready message. The string is passed to ioa_$nnl without change so it must contain specifications for each of the various values to be included in the ready message. The ioa_ control string formats for the various values that can be inserted into the ready message are given below for each type of value (See "List of Format Control Arguments"). This control argument overrides any format arguments that would normally affect the format of the ready message. However, format keywords must still be specified to indicate which values are to be output and the order in which these values correspond to the ioa_ control characters in the control string.

LIST OF FORMAT CONTROL ARGUMENTS: The format and content of the ready message are controlled by format arguments. These arguments include: control arguments that identify values to be included in the ready message; optional precision numbers following some of these control arguments that control the number of digits after the decimal point in numeric values; and literal character strings that are inserted in the ready message. The format arguments are combined in the order of their appearance in the general_ready command to form an ioa_ control string that controls the format of the ready message.

Six types of values can be used in a ready message:

processor usage values (virtual CPU seconds);
memory usage values (memory units);
paging operations (both bulk store reads and demand page faults);
usage cost values (dollar charges);
command processor level numbers;
and date/time values (date, time of day, day of the week, etc).

Both total usage values (total usage accrued during this process) and incremental usage values (usage accrued since the last ready message printed by general_ready) can be output in the same ready message. The values are selected for use in the ready message by format control arguments to general_ready. The format control arguments are listed below by type.

processor usage values
These control arguments can be followed by a single numeric digit from 1 to 9, to indicate the number of digits that should appear to the right of the decimal point in the number that is output. The default is three digits. The output format of the value can be described by the ioa_ control string " ^.nf" where n is 3 by default.

-inc_vcpu {N}
incremental virtual CPU value.

-total_vcpu {N}
total virtual CPU value.

memory usage values
These control arguments are used in the same manner as the control arguments for processor usage values above.

-inc_mem_units {N}
incremental units.

-total_mem_units {N}
total memory units.

usage cost values
These control arguments are used in the same manner as the control arguments for processor usage values with the following differences. The default number of digits following the decimal point is two. The output format of the value can be described by the ioa_ control string " $^.nf" where n is 2 by default.

-inc_cost {N}
incremental cost charges.

-total_cost {N}
cost charges.

paging values
These control arguments are output by the ioa_ control string " ^d+^d", where the first number is the number of bulk store

pages read (formerly the number of prepages) and the second is the number of demand page faults.

-inc_pf
incremental paging values.

-total_pf
paging values.

command processor level numbers
This control argument indicates that the command processor and stack frame level numbers should be included in the ready message. The number of digits is not settable. The level numbers are output by the ioa_ control string "^a", but the printed format can be described by " level ^d,^d" where the first number is the number of command processor invocations and the second is the stack frame depth of the ready message procedure's stack frame. If the command processor level is 1, the printed format is the null string.

-level
command processor level numbers.

date values
These values can be described by the ioa control string " ^a" except for the -min, -day, and -year control arguments which use the ioa_ control string "^a" (without a leading space). The number of digits is not settable.

-date
eight-character date (mm/dd/yy).

-date_time
date and time (mm/dd/yy hhmm.m zzz www).

-day
two-digit day (dd).

-day_name
three-character day of the week (www).

-hour
two-digit hour (hh).

-minute
two-digit minute (mm)

-month
two-digit month (mm).

-time, -tm
six-character time of day (hhmm.m).

-year
two-digit year (yy)

-zone
three-character time zone (zzz).

Notice that all values except those date values mentioned above are preceded by a space and none of the values are suffixed by a space. Any nonkeyword argument (other than a single numeric digit following a floating point or dollar keyword) is assumed to be a literal string that is inserted in the ioa_ control string being built by general_ready. Refer to the examples below.

LIST OF OPERATION CONTROL ARGUMENTS: The following control arguments affect the operation of general_ready, but do not change the format of ready messages.

-set
establishes general_ready as the current ready message procedure. The command processor then calls general_ready to print a ready message after each command line is complete. In addition, the system commands ready, ready_on, and ready_off, determine the operation of general_ready. This control argument also causes general_ready to set an alarm timer to catch shift changes.

-revert
makes the system ready procedure the current ready message procedure and resets any timer alarms established by general_ready to catch shift changes.

-reset
resets incremental usage values to zero without printing a ready message.

-call cmdline
when used with the -set control argument, causes general_ready to call the command processor to execute cmdline after the completion of every command line. cmdline is a single argument to general_ready and therefore, must be enclosed in quotes if it contains any blanks. A frequent use of -call is "-call print_messages". cmdline is executed even if the printing of ready messages has been inhibited by executing the ready_off command.

The -set and -revert control arguments are mutually exclusive. A general_ready command that includes -set does not print a ready message. Instead, it saves the ioa_ control string built from the format and prefix control arguments in the command, and uses this ioa_ string to control the format of ready messages printed when command lines complete execution or when a ready command is issued.

A general_ready command that includes -revert prints a single ready message, only if format or prefix control arguments appear in the command with the -revert control argument. Otherwise, no ready message is printed.

If neither -set nor -revert is given, then general_ready prints one ready message according to the format and prefix arguments given in the command.

NOTES: The program is designed to allow an almost arbitrary format at no additional cost (relative to the system's ready procedure) other than the cost associated with the general_ready command, which sets up the ready message. Once a ready message is specified, the ready_on, ready, and ready_off commands control the printing of the ready message in the normal manner.

The general_ready command builds up an ioa_ control string (described in the MPM Subroutines) from the order of the keywords passed to it. The keywords specify which values to output in the ready message. Virtual CPU usage and dollar cost can be printed. Both incremental usage (usage accrued since the last ready message produced by general_ready) and total usage (usage accrued during this process) can be in the same ready message with the precision of the output (the number of decimal places to the right of the decimal point) specified by the user. As a command, general_ready can be used to either print a single ready message or define the contents of the ready message printed by the ready command (and after every command line if the ready_on command is executed). When used as an active function, the return value of general_ready is the ready message.

The values for total virtual CPU time and total memory units is valid across new processes. The value for cost is valid unless a shift change occurred during a previous process. When general_ready is invoked for the first time in a process,

the dollar cost of all usage (in that process) up to that point in time is computed at the rates then in effect.

Due to the manner in which ready message procedures and procedures that set up alarm timers are invoked, such procedures should not be terminated (by the terminate and terminate_refname commands). If it is desired to terminate general_ready, general_ready should be invoked with the -revert control argument before it is terminated.

EXAMPLES: The following examples illustrate some of the facilities of general_ready:

```
gr -string READY -date ^xTIME -time ^xVCPU -inc_vcpu
   -total_vcpu -set
```

establishes general_ready as the current ready procedure since the -set keyword appeared. Each ready message has the format:

```
READY 01/15/74 TIME 1234.3 VCPU 3.456 23.349
```

If the -set keyword had not appeared, a single ready message having the above format is printed. The ioa_ control string that general_ready uses to generate the above ready message is:

```
"READY ^a^xTIME ^a^xVCPU ^.3f ^.3f^2/"
```

The command line:

```
! gr -string READY -date -hour : -minute ^xVCPUI -inc_vcpu
   ^xVCPUT -total_vcpu 2
```

results in a single ready message of the form:

```
READY 01/15/74 09:46 VCPUI 2.345 VCPUT 34.21
```

using the ioa_ control string:

```
"READY ^a ^a:^a^xVCPUI ^.3f^xVCPUT ^.2f^2/"
```

The above ready message can also be specified by the command line:

```
! gr -control "READY ^a ^a:^a VCPUI ^.3f VCPUT ^.2f^2/" -date
   -hour -minute -inc_vcpu -total_vcpu
```

SYNTAX AS A COMMAND:

gpn {-control_arg} arg

SYNTAX AS AN ACTIVE FUNCTION:

[gpn {-control_arg} arg]

FUNCTION: returns the absolute pathname of the segment designated by a specified reference name or segment number. (Reference names are discussed in the MPM Reference Guide.) If the reference name or segment number is not in use, an error message is printed.

ARGUMENTS:

arg
is a reference name or octal segment number known to this process.

CONTROL ARGUMENTS:

-name, -nm
indicates that arg (which happens to look like an octal segment number) is to be interpreted as a reference name. If this control argument is not specified, the system assumes arg is a reference name only if arg is not a valid octal number.

get_quota (gq)

SYNTAX AS A COMMAND:

gq {paths} {-control_arg}

FUNCTION: returns information about the secondary storage quota and pages used for a specified directory.

ARGUMENTS:

paths
are pathnames of directories for which quota information is desired. If one of the paths is -wd or -working_directory, the working directory is used. If no paths are specified, the working directory is assumed. The star convention is allowed.

CONTROL ARGUMENTS:

-long, -lg
prints output in long format. See "Notes" below.

NOTES: The short form of output (the default case) prints the number of pages of quota assigned to the directory and the number of pages used by the segments in that directory and any inferior directories that are charging against that quota. The output is prepared in tabular format, with a total, when more than one pathname is specified. When only one pathname is specified, a single line of output is printed.

The long form of output gives the quota and pages-used information provided in the short output. In addition, it prints the logical volume identifier of segments, the time-record product in units of record-days, and the date that this number was last updated. Thus, a user can see what secondary storage charges the user's accounts are accumulating. If the user has inferior directories with nonzero quotas, it is necessary to print this product for all these directories in order to obtain the charge.

get_system_search_rules (gssr)

SYNTAX AS A COMMAND:

gssr

FUNCTION: prints the definitions of site-defined search rule keywords acceptable to the set_search_rules command.

NOTES: The get_system_search_rules command prints a list of standard search rule keywords and directories, each one followed by one or more site-defined keywords. If the user includes a site-defined keyword in the search segment accepted by the set_search_rules command, the site-defined keyword expands into its definition in the order printed by get_system_search_rules.

See print_search_rules, add_search_rules, delete_search_rules, and set_search_rules in this manual.

EXAMPLES:

```
! gssr
  initiated_segments, default
  referencing_dir, default
  working_dir, default
  >system_library_standard, default, system_libraries
  >system_library_unbundled, default, system_libraries
  >system_library_1, default, system_libraries
  >system_library_tools, default, system_libraries
  >system_library_auth_maint, default, system_libraries
  <ready>
```

In the example above, default and system_libraries are site-defined keywords. If the user includes system_libraries in the search segment accepted by the set_search_rules command, system_libraries expands into:

```
>system_library_standard
>system_library_unbundled
>system_library_1
>system_library_tools
>system_library_auth_maint
```

SYNTAX AS A COMMAND:

greater strA strB

SYNTAX AS AN ACTIVE FUNCTION:

[greater strA strB]

FUNCTION: returns true if strA is greater than strB according to ASCII collating sequence; otherwise it returns false.

NOTES: The strings are compared character by character according to their ASCII code value (i.e., if the first character in each string has the same ASCII code value, compare the second character; if their values are identical, compare the third character; etc.).

To make numeric comparisons of strings, see ngreater and nless in this manual.

SYNTAX AS A COMMAND:

have_mail {path}

SYNTAX AS AN ACTIVE FUNCTION:

[have_mail {path}]

FUNCTION: as a command prints the terminal display "You have mail" if there is mail in the mailbox specified by path; otherwise it prints "No mail". As an active function returns true is there is mail in the mailbox specified by path; otherwise it returns false. If path is not specified, the user's current default mailbox is assumed.

EXAMPLES: The following lines from an exec_com segment print the mail at the user's terminal and then dprint the mail on a line printer.

```
&if [not [have_mail]] &then &goto skip_mail
answer no print_mail
do "file_output &1; answer yes print_mail; revert_output;
     dp -he &1 -dl &1" [date].[time].mail
&label skip_mail
```

SYNTAX AS A COMMAND:

```
help {info_names} {-control_args}
```

FUNCTION: prints information, in the form of online documentation called info segments, including descriptions of system commands, active functions, and subroutines, miscellaneous information about system status, system changes, and general information. In addition, each site or user can construct info segments to be printed by this command.

ARGUMENTS:

info_names
are the pathnames or entrynames of infos to be printed. Pathnames contain greater-than (>) or less-than (<) characters, or follow a -pathname control argument (described below). If a pathname is specified, it identifies the segment containing the info to be printed. Otherwise, help searches for segments matching an entryname using the "info_segments" search list. For subroutines, an entry point name can be included in the info_name (e.g., subroutine_$entry_point). A suffix of info is assumed. The star convention is allowed for info_names, except when an entry point name is specified or when the -entry_point control argument (described below) is used. (See "Notes" below for information about the use of the star convention with the help command.)

CONTROL ARGUMENTS:

-pathname path, -pn path
specifies the pathname of a segment containing the info to be printed. It is useful when the info to be printed is in the working directory, or when the pathname begins with a minus (-) character. For subroutines, an entry point name can be included with the final entryname of path (e.g., -pn >udd>Proj>Pers>info>get_subr$entry_value). A suffix of info is assumed. The star convention is allowed except when an entry point name or -entry_point control argument is used. (See "Notes" below for information about the use of the star convention with the help command.)

LIST OF CONTROL ARGUMENTS BY FUNCTION: The control arguments are arranged here according to the function they perform. The categories and their respective control arguments are listed

below (detailed descriptions follow the list, in the same order):

info selection

-pathname path, -pn path
   selects an info segment
-entry_point, -ep
   selects main subroutine info entry point

information selection
-header, -he
   prints only a heading line
-brief_header, -bfhe
   prints brief heading with info
-title
   prints section titles
-brief, -bf
   prints summary of command, active function, or subroutine info
-control_arg STRs, -ca STRs
   prints only description of an argument
-all, -a
   prints entire info without questions

starting paragraph

-section STRs, -scn STRs
   selects by section title
-search STRs, -srh STRs selects by words in paragraph

paragraph grouping
-minlines I
   sets minimum paragraph size
-maxlines J
   sets maximum paragraph grouping size

LIST OF QUERY RESPONSES: The user can reply to questions asked by the help command with any of the responses described in detail under "Responses to Questions Asked by the help Command" below. Possible responses include:

yes	print the next paragraph
skip	skip the next paragraph
rest	print remaining paragraphs
brief	print a brief summary of command, active function or subroutine usage information
no	stop printing information

The question/answer dialogue continues until all of the information is printed, or until the user replies "no".

EXAMPLES: When the help command is invoked without any arguments, it prints information describing how to use the help command. Other topics are requested by means of info_name arguments. For example, to print information about the list command, type:

```
help list
```

The help command begins by printing a heading which identifies the information being printed. It then prints the first paragraph of information. Then help asks if the user wants more help, using a question of the form:

```
XXX (7 lines).  More help?
```

where XXX is the title of the next section of information, or

```
8 more lines.  More help?
```

if the next paragraph contains more information from the current section.

The help command accepts several control arguments which select information to be printed. For example,

```
! help fortran -brief
```

briefly describes how to use the fortran command without asking the user any questions.

```
! help fortran*.** -header
```

prints the names of info segments whose names begin with "fortran". These info segments contain information about Multics FORTRAN.

```
! help acl_matching.gi -all
```

prints information about how Access Control List entries are matched with the User_ids given in access control commands such as set/acl. The entire info is printed without asking any questions. The complete list of control arguments

accepted by the help command is described below under "Info Selection", "Information Selection", "Starting Paragraph", and "Paragraph Grouping".

Contents of Info Segments

The information printed by help is stored in formatted segments called info segments. Each segment contains one or more blocks of information, called infos, which describe a particular command active function, subroutine, or topic.

An info begins with a heading line, consisting of a date on which the info was last modified and a brief title identifying the info. For command and active function infos, the program name (including any short name) is used as the title. For example:

```
04/10/77  list, ls
```

For subroutine infos, the subroutine name is used as the title.

Information in an info is divided into paragraphs. A paragraph is a group of information lines. The paragraphs in an info are separated from one another by two blank lines. The help command uses this separation to determine where one paragraph ends and the next begins.

Each paragraph contains a logically complete unit of information. Control arguments and responses are available to search for and print a particular paragraph. To avoid printing unnecessary information when such searches are performed, paragraphs are short (1 to 15 lines long) and deal with only a single subject.

The paragraphs describing a given topic are grouped together into a section. The first paragraph of each section begins with a title that names the topic described in that section. Section titles are short, usually consisting of one or two words followed by a colon (:).

Standard section titles are used in info segments provided with the Multics system so that users can search for a

particular information topic. For command and active function infos, the section titles in their standard order are:

Syntax:
shows how the program is invoked. Arguments are given a generic name (e.g., paths indicates that one or more pathnames are allowed). Optional arguments are shown in braces (e.g., {paths}). If the program allows control arguments, they are shown as -control_args in the syntax line.

Function:
gives a brief description of what the program does.

Arguments:
gives a brief description of each argument.

Control arguments:
gives a brief description of each control argument.

Notes:
gives comments, clarifications, or any special case information.

Examples:
gives sample invocations of the program.

The descriptions of arguments and control arguments are formatted in a special way so that help can print a list of all argument and control argument names, or can find and print the description of an individual argument. Each description begins with a line naming the argument or control argument. This naming line includes the short name of a control argument, and names any operands required by a control argument. The description continues on subsequent lines by defining the meaning and function of the argument or control argument. Lines following the naming line are indented three spaces from the left margin so that help can identify the naming lines. A sample description for the section control argument of the help command is shown below.

-section STRs, -scn STRs
begins printing at the section whose title contains all strings STRs. By default, printing begins at the top.

Subroutines are described by a series of information blocks (infos), one describing each subroutine entry point. The first info describes the general purpose of the subroutine as a whole, and can include control information and notes common to all entry points. It includes the following sections.

Funtion:
describes the overall function performed by the subroutine. This section is optional.

Entry points in xxx:
lists the entry points defined in the subroutine.

Each entry point info includes a Syntax section, giving PL/I declare and call statements for the subroutine entry point. In addition, information from the first info that is common to all entry points is copied by help into each entry point info. This allows each entry point info to be a complete description of that entry point.

Infos can contain section titles other than those listed above. A user can issue the help command with the -title control argument, or use the title response from within help, to list the section titles used in a particular info.

INFO SELECTION

The following control arguments select the information blocks (infos) to be printed. If no control arguments from this section are specified, and no info_name arguments are specified, information about the help command is printed.

-pathname path, -pn path
specifies the pathname of a segment containing the info to be printed. It is useful when the info to be printed is in the working directory, or when the pathname begins with a minus (-) character. For subroutines, an entry point name can be included with the final entryname of path (e.g., -pn >udd>Project_id>Person_id>info>get_subr$entry_value). A suffix of info is assumed if one is not given. The star convention is allowed except when an entry point name or -entry_point control argument is used. (See "Notes" below for information about the use of the star convention with the help command.)

-entry_point, -ep
selects the info describing the main entry point of a subroutine. For example,

```
! help ioa_ -ep
```

prints the info describing the ioa_$ioa_ subroutine entry point. When the -entry_point control argument is omitted and no entry point name is specified by an info_name identifying a subroutine info segment, help prints the info describing the general purpose of the subroutine. For example,

```
! help ioa_
```

prints the info describing the general purpose of all ioa_ entry points.

Two other control arguments, -section and -search, further select the infos to be printed. The major purpose of these control arguments is to select the paragraph to be printed first. However, when searching an info selected by info_names and the -pathname and -entry_point control arguments, if no paragraph matching the -section or -search criteria is found, that info is passed over without comment. Thus -section and -search serve as a secondary info selection mechanism. Refer to "Starting Paragraph" below for a description of the -section and -search control arguments. Refer to "Notes" below for a complete description of this selection mechanism.

INFORMATION SELECTION

The following control arguments select the kind of information that help prints. If no information selection control argument is specified, help prints a long info heading line, followed by the first paragraph in the info. At the end of each paragraph, help asks the user if "More help" is needed.

-header, -he
prints only a long heading line, including: pathname of the info, info heading, and line count. No other information is printed. This control argument conflicts with all other information selection control arguments.

-brief_header, -bfhe
shortens the long heading line that is printed by default. Instead, help prints a brief heading line, followed by information selected by the other information selection control arguments or by the first paragraph if no other

information selection control arguments are specified. A brief heading line includes: info heading, and line count.

-title
lists the section titles used in the info (including section line counts), then asks if the user wishes to see the first section.

-brief, -bf
prints a brief summary of a command, active function or subroutine info segment without asking questions. The summary includes the Syntax section, and (for commands and active functions) a list of control arguments. This control argument conflicts with -title and -all.

-control_arg STRs, -ca STRs
prints only the descriptions of the control (or other) arguments whose names contain one of the strings STRs. STRs must not include a leading minus sign (-). For example,

```
! help mail -ca brief match exclude
```

prints descriptions of the -brief, -match and -exclude control arguments of the mail command. All arguments following -ca until the next control argument are treated as STRs. The help command prints no other information besides the argument descriptions and asks no questions of the user. This control argument conflicts with -title and -all.

Often, a control argument name following -ca is typed erroneously with a leading minus sign. Because this error occurs frequently, help ignores the minus sign on the first argument following -ca. However, subsequent arguments following -ca having leading minus signs are treated as control arguments for the help command, rather than as operands for -ca. For example, help treats:

```
! help mail -ca -brief
```

as if the user had typed:

```
! help mail -ca brief
```

and gives a description of the -brief control argument of the mail command. However, help reports an error for the command:

```
! help mail -ca -brief -match -exclude
```

because -match and -exclude are not valid control arguments to the help command. Note that, if the command:

```
! help mail -ca -header -brief
```

is typed, help does not report an error. The -brief causes help to print a summary of the mail command. The -ca -header then prints a description of mail's -header control argument.

These cases point out that the best procedure is to omit leading minus signs from the operands of -ca.

-all, -a
prints the entire info or subroutine entry point description without asking the user questions.

STARTING PARAGRAPH

Normally, help begins printing the first paragraph in the info. The control arguments below can select a particular section and/or a particular paragraph at which printing is to start.

-section STRs, -scn STRs
begins printing the section whose title contains the strings STRs. The entire section title is not required. Instead, the first section whose title contains all of the strings STRs is selected. The strings can appear in the section title in any order. The strings can be typed in lowercase, since case is ignored during matching operations. All arguments following the -section control argument until the next control argument are treated as STRs.

-search STRs, -srh STRs
begins printing with the first paragraph containing the strings STRs. All of the strings must appear in the selected paragraph, but they can appear in any order. The strings can be typed in lowercase, since case is ignored when matching. All arguments following the -search control argument are treated as STRs, so -search must be the last control argument to the help command. The search usually begins with the first paragraph, but when -section is also specified it begins with the matching section and continues to the last paragraph (i.e., without wraparound).

When -section or -search control arguments are specified and no matching paragraph is found in one of the infos selected by an info_name or info selection control argument, that info is passed over without comment. Thus, the starting paragraph control arguments serve as a secondary info selection mechanism.

The starting paragraph control arguments can be used with any of the information selection control arguments listed above, but its effect differs depending upon which of them are used. When -section or -search is used with -header, only the heading lines for infos containing a matching paragraph are listed. The matching paragraph itself is not printed. When they are used with -brief or -control_arg, help prints a heading line and then the information selected by -brief or -control_arg. The matching paragraph is not printed.

When -section or -search is used with -no_header, a brief heading line is printed preceding the matching paragraph. When used with -title, help prints a heading line, then the list of section titles, and finally the matching paragraph. When used with -all, the entire info is printed for infos containing a matching paragraph.

PARAGRAPH GROUPING

The following control arguments determine how much information help prints before asking if the user wants to see more.

-minlines I
sets the minimum paragraph size to I lines. Paragraphs smaller than this size are printed with preceding paragraphs. The default is 4.

-maxlines J
sets the maximum paragraph grouping size to J lines. When paragraphs are grouped together, the number of grouped lines cannot exceed this size. The default is 15.

The -minlines I control specifies the length in lines of the smallest paragraph that help treats as a distinct unit. Paragraphs shorter than I lines are printed as part of the preceding paragraph.

The -maxlines J control limits the number of short paragraphs (those shorter than I lines) that are grouped together. No more than J lines of information are printed before asking if the user wants more help.

For example, consider an info divided into paragraphs as follows:

```
Paragraph 1 (8 lines)
 (2 blank lines)
Paragraph 2 (3 lines)
 (2 blank lines)
Paragraph 3 (4 lines)
```

With -minlines 4 and -maxlines 15, help treats paragraph 2 as a short paragraph and prints it with paragraph 1 (total lines = 13). However, paragraph 3 is 4 lines long, and is treated as a distinct paragraph.

With -minlines 5 and -maxlines 10, help prints paragraph 1 separately, since grouping short paragraph 2 with paragraph 1 prints 13 lines, exceeding -maxlines. Paragraphs 2 and 3 are grouped together (total lines = 9) because paragraph 3 is shorter than 5 lines.

Paragraphs that have been seen are not grouped with unseen paragraphs. Similarly, paragraphs at the end of one section of info are not grouped with those beginning another section. Paragraphs are not grouped when the -section or -search control arguments are used to find a particular starting paragraph. If the wrong paragraph is found by the search, grouping might compound the error by printing more of the wrong information. For similar reasons, grouping is suppressed when the section and search responses are used.

Responses to Questions Asked by the help Command

The responses accepted when help questions the user are given in the list below. Those responses that search the info or list section titles operate from the current paragraph to the end of the info. No wraparound feature is employed. However, -top or -t can be used with these responses to cause searching or listing from the top of the info, rather than from the current paragraph.

The help command remembers which paragraphs the user has seen and which have been skipped or not yet reached. It asks the user to "Review" paragraphs seen before, but asks if "More help" is needed for unseen paragraphs. It stops printing if all paragraphs have been seen when the end of info is reached. However, if any paragraphs were skipped, help asks if user wants to see them. If the response is "yes", the first unseen paragraph is printed. The user can then answer "skip -seen" to view subsequent unseen paragraphs.

The responses to all questions asked by the help command can be chosen from the following:

yes, y
prints the next paragraph of information, then asks whether the user wants more help.

no, n
exits from the current info, and begins printing the next info selected by info_names given in the help command. Returns from the help command if all selected infos have been printed.

quit, q
causes the help command to return without printing the remaining infos selected by the info_names.

rest {-scn}, r {-scn}
prints the rest of the info without intervening questions. If -section or -scn control arguments are specified, help prints only the rest of the current section without questions. When the section has been printed, help then asks whether the user wants to see the next section.

top, t
skips to the beginning of the info, prints the heading line, and asks whether the user wants to see the first section.

title {-top}
lists titles and line counts of all sections remaining in the current info. If -top or -t is specified, help lists all section titles.

section {STRs} {-top}, scn {STRs} {-top}
skips to the next section whose title contains all strings STRs. Title matching is performed as described above for the -section control argument. If -top or -t is specified, title searching starts at the beginning of the info. If STRs are omitted, help uses the search strings from the previous section response or -section control argument. If the search fails, help prints the message:

No matching section found.

and repeats the previous question.

search {STRs} {-top}, srh {STRs} {-top}
skips to the next paragraph containing all strings STRs. Paragraph selection is performed as described above for the -search control argument. If -top or -t is specified,

searching starts at the beginning of the info. If STRs are omitted, help uses the strings from the previous search response or -search control argument. If the search fails, help prints the message:

No matching paragraph found.

and repeats the previous question.

skip {-scn} {-rest} {-seen} {-ep},
s {-scn} {-rest} {-seen} {-ep}
skips the next paragraph and asks whether the user wants to see the paragraph following it. If -section or -scn is specified, help skips all paragraphs of the current section. If -rest or -r, -entry_point or -ep are specified, help skips the rest of this info or subroutine entry point description, continuing with the next. If -seen is specified, help skips to the next paragraph that the user has not seen. Only one of these control arguments can be used at a time.

brief, bf
prints a summary of a command, active function or subroutine info, including Syntax section and a list of control arguments, then repeats the previous question.

control_arg STRs, ca STRs
prints descriptions of control (or other) arguments whose names contain one of the strings STRs, then repeats the previous question.

entry_point {EP_NAME}, ep {EP_NAME}
skips to the description of subroutine entry point EP_NAME. The EP_NAME can be specified as entry_point_name or subroutine_$entry_point_name. For example,

! ep rsnnl

when in the info segment describing the ioa_ subroutine, skips to the description of the ioa_$rsnnl entry point. If EP_NAME is omitted, help skips to the description of the subroutine_$subroutine_ entry point.

?
prints a list of available responses.

.
prints "help" to identify the current interactive environment.

.. command_line
passes the remainder of the response to the Multics command processor as a command line.

header, he
prints a long heading line to identify the current info. The line includes: pathname of the info, info heading, and line count.

Search List

The help command uses the "info_segments" search list, which has synonyms of "info_segs" and "info". For more information about search lists, see the descriptions of the search facility commands and, in particular, the add_search_paths command description in this manual. Type:

```
! psp info_segments
```

to see what the current "info_segments" search list is. The default search list is:

```
>doc>iml_info
>doc>info
```

NOTES: When the star convention is used, the help command performs the following steps:

1. The info segments whose entrynames match any of the star names are alphabetized within their directory and scanned in that order.

2. When -section and -search control arguments are specified, help scans the matching infos until the desired section and/or paragraph is found. If a matching paragraph is found, help prints it. Then help asks the user whether to print remaining paragraphs. Note that any section and search responses given at this point scan only the current info. If a matching paragraph is not found in one of the infos selected by a star name, that info is passed over without comment. Thus, it is possible to scan all info segments and print only those containing certain section titles or certain words.

3. When -section and -search control arguments are not specified, help begins printing the first paragraph of each

info that matches any of the starnames. Then help asks the user whether to print the remaining paragraphs.

4. The -title, -all, -brief and -control_arg control arguments apply to each info selected by the starnames and -section/-search string matching. Section titles, a brief summary or particular control argument descriptions are printed before the matching paragraph. When -all is combined with -section or -search, the entire info selected by the string matching is printed without questions.

5. The yes, no, rest, and skip responses operate on the next selected paragraph. This paragraph can be the first paragraph of the next selected info, or even the first paragraph that matches the -section and -search criteria in the next selected info.

6. If the user issues a quit signal, the program_interrupt command can be used to reenter the interactive help environment. The question asked previously is repeated.

Info Naming Conventions

Infos for Multics commands, active functions and subroutines are given the name of the particular system module with a suffix of info. For example, the info describing the pl1 compiler command is called pl1.info.

Information about changes made to a command or active function from one release to the next are given the name of the particular system module with a suffix of changes.info. For example, changes to the fortran compiler are described in fortran.changes.info.

General information describing features or use of the system is included in infos whose names end with a suffix of gi.info (gi for general information). For example, acl_matching.gi.info describes how Access Control List entries are matched with User_ids in access control commands such as set_acl.

More than 500 infos are available. To find information about a particular area of the system, use list_help, described in this manual, or the -header control argument with an entryname containing stars to list the names of available infos. For

example, to list info_names related to the FORTRAN compiler, the user can type:

```
! help fortran*.** -he
```

To get a list of all general information segments, type:

```
! help *.gi -he
```

Info Segment Format

Users can create info segments describing their own commands, exec_coms and application programs. Info segments must be formatted in a special way so that the help command can parse them into paragraphs. For information about this format, type:

```
! help info_seg.gi
```

Examples

In the examples given below, the lines typed by the user are indicated by an exclamation point at the beginning of the line or immediately preceding a request. In the first example, the user wants to see list.info.

```
! help list
  (6 lines follow;  131 lines in info)
  04/10/77  list, ls

  Syntax:  ls {entrynames} {-control_args}

  Arguments (9 lines).  More help?  ! yes

  Arguments:

  entrynames
     are the names of entries to be listed. The star
     convention can be used. If no entrynames are specified,
     all entries in the directory (of the default types or
     the types specified by control arguments) are listed. A
     pathname can be specified instead of an entryname,
     causing the entries specified by its entryname portion
     to be listed, in the directory specified by its
     directory portion. It is an error to specify more than
     one directory to be listed in a single invocation of the
     list command.
```

```
   Control arguments (5 lines).  More help?  ! no
   r 1459 0.753 744
```

In the following example, the user knows what the "Syntax" and "Function" sections are but wants to see the control arguments section for the list command. Note that the argument for the -section control argument can be upper or lower case.

```
!  help list -scn Control
   (7 lines follow;  131 lines in info)
   04/10/77  list, ls

   Control arguments:  described below according to their
   functions.
   DIRECTORY
   -pathname path, -pn path
      list entries in the directory named path.  Note the
      restriction described above under Arguments.

   15 more lines.  More help?  ! yes

   ENTRY TYPE
   -segment, -sm
      list segments.
   -multisegment_file, -msf
      list multisegment files.
   -file, -f
      list files (segments and multisegment files).
   -directory, -dr
      list directories.
   -branch, -br
      list branches (segments, multisegment files, and
      directories).
   -link, -lk
      list links.
   -all, -a
      list all four entry types.

   12 more lines.  More help?  ! no
   r 1500 0.215 264
```

In the following example, the user is searching for all the list commands that have the word request anywhere in the info.

```
!  help list_*.info -srh request
   >doc>info>list_abs_requests.info (6 lines follow; 56 in
   info)
```

```
07/20/78  list_abs_requests, lar
          list_daemon_requests, ldr
          list_retrieval_requests, lrr

Syntax:  (lar ldr lrr) {rel_path} {-control_args}

Function & Arguments (9 lines).  More help?  ! yes

Function:  these commands list requests in the absentee,
I/O daemon, and retrieval queues, respectively.


Arguments:
rel_path
   is relative pathname of request(s) to be listed.  It can
   end in a starname.  Default is to list requests of all
   pathnames.  See also the -entry control argument.

Control arguments (34 lines).  More help?  ! no

>doc>info>list_carry_requests.info  (3 lines follow; 22 in
info)
07/12/78  list_carry_requests, lcr

Syntax:  lcr {-control_args}

r 1732 2.585 647
```

If the user wishes to see the info for the help_ subroutine with all the entry points described, he would do the following:

```
!  help help_
   >doc>info>help_.info
   (3 lines follow,  9  in  introduction; 74  lines,  4 entry
      points in info)
   12/11/78  help_

   Entry points in help_ (4 lines).  More help?  ! yes

   Entry points in help_:
      12/11/78  help_$init        12/11/78  help_$check_info_segs
      12/11/78  help_ (entry      12/11/78  help_$term
         point)

   Entry:  12/11/78  help_$init
   (10 lines  follow; 16 lines  in entry point)  More help?  !
      yes
```

```
   Syntax:
   declare help_$init entry (char(*), char(*), char(*),
      fixed bin, ptr, fixed bin (35));

   call help_$init (caller, search_list_name,
      search_list_ref_dir, required_version, Phelp_args, code);


   Notes:  The structure pointed  to by Phelp_args is declared
      in help_args_.incl.pl1.

   Entry points in help_ (4 lines).  Review?  ! yes

   Entry points in help_:
      12/11/78  help_$init          12/11/78  help_$check_info_segs
      12/11/78  help_ (entry        12/11/78  help_$term
         point)

   Entry:  12/11/78 help_ (entry point)
   (8 lines follow; 14 lines in  entry point).  More help ?  !
      no
   r 1644 0.374 567
```

If just the help entry point to the help_ subroutine is required, type:

```
!  help  help_ -ep
   >doc>info>help_.info
   (10 lines follow, 16 in entry point;
   74 lines, 3 other entry points in info)
   12/11/78  help_ (entry point)

   Syntax:
   declare  help_  entry  (char(*), ptr,  char(*),  fixed bin,
      fixed bin(35));

   call help_ (caller, Phelp_args, suffix, progress, code);


   Notes:  The structure pointed  to by Phelp_args is declared
      in  help_args_.incl.pl1, and  is obtained  to by calling
      help_$init.

   Entry points in help_ (4 lines).  More help?  ! no
   r 1642 0.684 920
```

To print just a syntax line and a list of control arguments for delete_acl, type:

```
! help da -bf
Syntax:  da {path} {User_ids} {-control_args}

Control arguments:     -directory, -dr     -brief, -bf
-all, -a               -segment, -sm
```

hexadecimal (hex)

SYNTAX AS A COMMAND:

hex values

SYNTAX AS AN ACTIVE FUNCTION:

[hex values]

FUNCTION: returns one or more values in hexadecimal.

ARGUMENTS:

value
is a value to be processed. The last character of the value indicates its type. Acceptable types are binary (b), quartenary (q), octal (o), hexadecimal (x), or unspecified (u). Any valid PL/I real value is allowed. The absence of any specifier means decimal. The specified value is limited to 8 characters.

EXAMPLES:

```
! string [hex 377o]
ff
```

SYNTAX AS AN ACTIVE FUNCTION:

[high N]

FUNCTION: returns a specified number of copies of the last (highest) character in the ASCII character set, the PAD character or 177 octal.

SYNTAX AS AN ACTIVE FUNCTION:

[high9 N]

FUNCTION: returns a specified number of copies of the last (highest) 9-bit bit pattern, 777 octal.

home_dir (hd)

SYNTAX AS A COMMAND:

hd

SYNTAX AS AN ACTIVE FUNCTION:

[hd]

FUNCTION: returns the pathname of the user's home directory (usually of the form >user_dir_dir>Project_id>Person_id).

SYNTAX AS A COMMAND:

hour {dt}

SYNTAX AS AN ACTIVE FUNCTION:

[hour {dt}]

FUNCTION: returns the one- or two-digit number of an hour of the day, from 0 to 23.

ARGUMENTS:

dt
is a date_time in a form acceptable to convert_date_to_binary_. If no argument is specified, the current hour is returned.

how_many_users (hmu)

SYNTAX AS A COMMAND:

hmu {-control_args} {optional_args}

FUNCTION: tells how many users are currently logged in on the system.

CONTROL ARGUMENTS:

-long, -lg
prints additional information including the name of the installation, the time the system was brought up, the time of the next shutdown, if it has been scheduled, and the time of the last shutdown or crash. Load information on absentee users is also printed.

-absentee, -as
prints load information on absentee users only, even if the absentee facility is not running.

-brief, -bf
suppresses the printing of the headers. Only used in conjunction with one of the optional_args.

LIST OF OPTIONAL ARGUMENTS:
specifies that only selected users are to be listed and can be one of the following:

Person_id
lists a count of logged in users with the name Person_id.

.Project_id
lists a count of logged in users with the project name Project_id.

Person_id.Project_id
lists a count of logged in users with the name and project of Person_id and Project_id.

NOTES: In addition to how many users are currently logged in, this command prints the name of the system, the current load on the system, and the maximum load. If the absentee facility is running, the number of absentee users and the maximum number of absentee users is printed also.

how_many_users (hmu)

If this command is invoked without any arguments, basic summary information is printed (see the first example below).

Absentee counts in a selective use of how_many_users (i.e., when an optional_arg is specified) are denoted by an asterisk (*).

Up to 20 classes of selected users are permitted.

EXAMPLES: To print summary information, type:

```
! hmu
  Multics MR8.0, load 15.0/50.0; 15 users, 6 interactive,
     9 daemons.
```

To print summary information on absentee users, type:

```
! hmu -as
  Absentee users 0/2
```

To print the additional information provided by the -long control argument, type:

```
! hmu -lg
  Multics 8.0: PCO, Phoenix, Az.
  Load = 13.0 out of 110.0 Units; users = 13,
     4 interactive, 9 daemons.
  Absentee users = 0 background;
  Max background users = 2
  System up since 11/21/79  0908.1
  Last shutdown was at 11/18/79  02304.1
```

To print brief information about the SysDaemon project, type:

```
! hmu -bf .SysDaemon
  .SysDaemon = 3 + 0*
```

To print brief information about the user whose Person_id is Smith, type:

```
! hmu -bf Smith
  Smith = 1 + 1*
```

SYNTAX AS A COMMAND:

```
if expression -then cmdline1 {-else cmdline2}
```

FUNCTION: provides conditional execution of a command line.

ARUGMENTS:

expression
is true or false. Usually, expression is an active string that the command processor evaluates to either true or false.

cmdline1
is a command line to be executed if expression is true. If the command line contains blanks, it must be enclosed in quotes.

cmdline2
is an optional command line to be executed if expression is false. If the command line contains blanks, it must be enclosed in quotes.

NOTES: If -else cmdline2 is omitted, no action is taken if expression is false. If expression is neither true nor false, no action is taken. If cmdline1 is not present and expression is true, no action is taken.

EXAMPLES: The following abbreviation compares two segments. If they are identical, one is deleted; if they are not identical, the differences are placed in a file, which is then printed.

```
.ab check do "if [compare  &1 &2] -then ""delete &2"" -else
   ""fo check; cpa &1 &2; ro; dp -dl check"""
```

SYNTAX AS A COMMAND:

im {destination} {-control_arg}

FUNCTION: restores the immediate printing of messages sent to the user by the send_message command and the "You have mail." notification sent by the send_mail command.

ARGUMENTS:

destination
is of the form Person_id.Project_id to specify a mailbox. The default is the user's default mailbox. If destination contains < or >, it is assumed to be the pathname of a mailbox.

CONTROL ARGUMENTS:

-pathname path, -pn path
specifies a mailbox by pathname. The mbx suffix is assumed. This control argument and the destination argument are mutually exclusive.

NOTES: This command "cancels" the defer_messages command.

For a description of the mailbox, refer to the accept_messages and print_mail commands.

SYNTAX AS A COMMAND:

ind oldpath {newpath} {-control_args}

FUNCTION: improves the readability of a PL/I source segment by indenting it according to a set of standard conventions described below.

ARGUMENTS:

oldpath
is the pathname of the input source segment. Source segments with suffixes for PL/I, create_data_segment, and the reduction_compiler are recognized. If the segment does not have a recognized suffix, indent attempts to use a segment of .pl1, name.cds or name.rd, in that order.

newpath
is the pathname of the output source segment. The output segment must have the same suffix as the input segment. If this argument is omitted, newpath is assumed to be the same as oldpath, and the indented copy of the program replaces the original copy. However, if errors are detected during indentation and newpath is not specified, the original copy is not replaced. Instead, the pathname of the temporary file containing the indented copy is printed in an error message.

CONTROL ARGUMENTS:

-brief, -bf
suppresses warning comments on invalid or non-PL/I characters found outside of a string or comment. (Such characters are never removed.) When this argument is specified, those errors whose warning messages are suppressed do not prevent the original copy from being replaced.

-lmargin STR, -lm STR
sets the left margin (indentation for normal program statements) to STR. If this argument is omitted, the default left margin is 11.

-comment STR, -cm STR
sets the comment column to STR. Comments are lined up in this column unless they occur in the beginning of a line or are preceded by a blank line. If this argument is omitted, the default comment column is 61.

-indent STR, -in STR
sets indentation for each level to STR. Each do, begin, proc, and procedure statement causes an additional STR spaces of indentation until the matching end statement is encountered. If this argument is omitted, the default indentation is 5.

NOTES ON CONVENTIONS: Declaration statements are indented five spaces for dcl declarations and ten for declare declarations. Identifiers appearing on different lines of the same declare statement are lined up under the first identifier on the first line of the statement. Structure declarations are indented according to level number; after level two, each additional level is indented two additional spaces.

An additional level of indentation is also provided for the then clause of an if-statement; else clauses are lined up with the corresponding if. Statements that continue over more than one line have an additional five spaces of indentation for the second and all succeeding lines.

Multiple spaces are replaced by a single space, except inside of strings or for nonleading spaces and tabs in comments. Trailing spaces and tabs are removed from all lines. The indent command inserts spaces before left parentheses, after commas, and around the constructs =, ->, <=, >=, and ^=. Spaces are deleted if they are found after a left parenthesis or before a right parenthesis. Tabs are used wherever possible to conserve storage in the output segment.

The indent command counts parentheses and expects them to balance at every semicolon. If parentheses do not balance at a semicolon, or if the input segment ends in a string or comment, indent prints a warning message. Language keywords (do, begin, end, etc.) are recognized only at parenthesis level zero, and most keywords are recognized only if they appear to begin a statement.

NOTES ON RESTRICTIONS: Lines longer than 350 characters are split, since they overflow indent's buffer size. This is the only case in which indent splits a line.

Labelled end statements do not close multiple open do statements.

The indent command assumes that the identifiers begin, end, procedure, proc, declare, and dcl are reserved words when they appear at the beginning of a statement. If the input contains a statement like:

```
do = do + 1;
```

the indent command interprets it to mean that the statement delimits a do group and does not indent correctly.

Structure level numbers greater than 99 do not indent correctly.

SYNTAX AS A COMMAND:

```
index strA strB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[index strA strB]
```

FUNCTION: returns an integer representing the character position in strA where strB begins. If strB does not occur in strA, 0 is returned.

EXAMPLES:

```
! string [index abcdefhgij ef]
5
! string [index "Now is the time" hte]
0
```

SYNTAX AS A COMMAND:

```
index_set {F1} B1 {I1} ... {Fn} Bn {In}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[index_set {F1} B1 {I1} ... {Fn} Bn {In}]
```

FUNCTION: returns one or more integers, separated from each other by spaces.

ARGUMENTS:

F
is the first number of a set, either a positive or negative integer. This argument is optional (see "Examples").

B
is a bound on the set, either a positive or negative integer.

I
is the increment between the numbers of a set, either a positive or negative integer. If F > B, then I is assumed to be a negative integer. Otherwise, I is assumed to be positive. This argument is optional (see "Examples").

NOTES: If more than one F-B-I triple is specified, F, B, and I must be specified in each triple. If only one F-B-I triple is specified, I or both I and F can be omitted. I and F are assumed to be 1 if omitted.

EXAMPLES: The following interactions illustrate the index_set active function:

```
! string [index_set 6]
  1 2 3 4 5 6
! string [index_set 5 21 3]
  5 8 11 14 17 20
! create file_([picture 99 [index_set 5 21 3]])
```

```
! list file_*

   Segments = 6, Lengths = 0.

   r w     0  file_20
   r w     0  file_17
   r w     0  file_14
   r w     0  file_11
   r w     0  file_08
   r w     0  file_05
```

The following interactions illustrate command usage:

```
! index_set 4 20 5
   4 9 14 19
! index_set 4 20 5 8 30 6
   4 9 14 19 8 14 20 26
! index_set 5
   1 2 3 4 5
! index_set 5 2
   5 4 3 2
```

SYNTAX AS A COMMAND:

in path {ref_names} {-control_args}

FUNCTION: enables users to make segments known directly, i.e., not using the normal search rules.

ARGUMENTS:

path
is the pathname of a segment to be made known. A relative pathname can be used. The star convention is NOT allowed.

ref_names
are optional reference names by which the segment can be known without further initiating. See "Notes" below.

CONTROL ARGUMENTS:

-all, -a
initiates all the names on the segment in addition to any ref_names specified.

-chase
used in conjunction with -all, uses the names on a link's target rather than those on the link. This is the default.

-force, -fc
terminates specified ref_names, or the names of the segments if no ref_names are specified, first if they are already initiated.

-long, -lg
prints the segment number assigned to the segment on the user's terminal.

-no_chase
used in conjunction with -all, uses the names on a link rather than those on the link's target.

NOTES: For a discussion of search rules, see "Search Rules" in the MPM Reference Guide. When this command is used to explicitly make known a segment by some reference name, the first reference to that name accesses the initiated segment instead of searching among the search directories for a segment by that name.

If no ref_names are specified in the command line, the segment is made known by the entryname part of the pathname. If any reference name is present in the command line, the entryname of the segment is not initiated; the specified reference names are. If the pathname is a single element name, the directory assumed is the working directory. The star convention is not supported.

If a reference name cannot be initiated, an error message is printed and the command continues initiating the other names.

To make a segment known, the user must have nonnull access to that segment.

EXAMPLES: The command line:

```
! in >udd>Demo>Jones>gamma x y
```

makes the segment >udd>Demo>Jones>gamma known, initiating the names x and y.

The command line:

```
! in pop
```

makes the segment pop in the working directory known, and initiates it with the reference name pop.

The command line:

```
! in xx u v -long
```

makes the segment xx in the working directory known, initiates the reference names u and v, and prints out the assigned segment number.

SYNTAX AS A COMMAND:

io opname switchname {args}

SYNTAX AS AN ACTIVE FUNCTION:

[io opname switch_name {args}]

FUNCTION: performs a variety of operations on specified I/O switches and returns a result.

ARGUMENTS:

opname
designates the operation to be performed. Opnames permitted, followed by their alternate forms where applicable, are:

attach	move_attach
close	open
control	position
delete_record, delete	print_iocb
detach_iocb, detach	put_chars
destroy_iocb	read_key
find_iocb	read_length
get_chars	read_record, read
get_line	rewrite_record, rewrite
look_iocb	seek_key
modes	write_record, write

Allowed by the io_call active function only:

attached	opened
attach_desc	open_desc
closed	test_mode
detached	valid_mode
io_module	valid_op

See "List of operations" below for a description of each opname, its command syntax line, and specific application. Operations are arranged functionally rather than alphabetically.

switchname
is the name of the I/O switch.

args
can be one or more arguments, depending on the particular operation to be performed.

LIST OF OPERATIONS: Unless otherwise specified, if a control block for the I/O switch does not already exist, an error message is printed on error_output and the operation is not performed. If the requested operation is not supported for the switch's attachment and/or opening, an error message is printed on error_output.

Differences between command and active function invocation are described under the individual operations.

The explanations of the operations cover only the main points of interest and, in general, treat only the cases where the I/O switch is attached to a file or device. For full details see the descriptions of the iox_ subroutine and the I/O modules in the MPM Subroutines "Input and Output Facilities," in the MPM Reference Guide.

Operation: attach

Syntax: io attach switchname modulename {args}

where:

modulename
is the name of the I/O module to be used in the attachment. If modulename contains less-than (<) or greater-than (>) characters, it is assumed to be a pathname, otherwise, it is a reference name.

args
can be one or more arguments, depending on what is permitted by the particular I/O module.

Function: attaches the I/O switch using the designated I/O module. The attach description is the concatenation of modulename and args separated by blanks. The attach description must conform to the requirements of the I/O module. If the I/O modulename is specified by a pathname, it is initiated with a reference name equal to the entryname. If the entryname or reference name does not contain a dollar sign

($), the attachment is made by calling modulename$modulenameattach. If a dollar sign is specified, the entry point specified is called. See "Entry Point Names" in the MPM Reference Guide.

If a control block for the I/O switch does not already exist, one is created.

Operation: detach_iocb, detach

Syntax: io detach switchname

Function: detaches the I/O switch.

Operation: open

Syntax: io open switchname mode

where mode is one of the following opening modes, which can be specified by either its full name or its abbreviation:

stream_input, si	sequential_update, squ
stream_output, so	keyed_sequential_input, ksqi
stream_input_output, sio	keyed_sequential_output, ksqo
sequential_input, sqi	keyed_sequential_update, ksqu
sequential_output, sqo	direct_input, di
sequential_input_output, sqio	direct_output, do
	direct_update, du

Function: opens the I/O switch with the specified opening mode.

Operation: close

Syntax: io close switchname

Function: closes the I/O switch.

Operation: get_line

Syntax: io get_line switchname {N} {-control_args}

io_call (io)

where:

N
is a decimal number greater than zero specifying the maximum number of characters to be read.

control_args
can be any of the following:

-segment path {offset}, -sm path {offset}
specifies that the line read from the I/O switch is to be stored in the segment specified by path, at the character position specified by offset. The default offset is the position after the latest line read, or 1.

-nnl
specifies that the newline character, if present, is deleted from the end of the line.

-nl
specifies that a newline character is added to the end of the line if one is not present.

-lines
specifies that the offset, if specified, is measured in lines rather than characters. This control argument only has meaning if the -segment control argument is also specified.

Function: reads the next line from the file or device to which the I/O switch is attached. If N is given, and the line is longer than N, then only the first N characters are read.

If the -segment control argument is not specified, the line read is written onto the I/O switch user_output, with a newline character appended if one is not present and if -nnl has not been specified.

If the -segment control argument is specified, the line is stored in the segment specified by path. If this segment does not exist, it is created. If offset is specified, the line is stored at that position relative to the start of the segment. This is normally measured in characters, unless -lines has been used. If offset is omitted, the line is appended to the end of the segment. The bit count of the segment is always updated to a point beyond the newly added data.

Syntax: [io get_line switchname {N} {-control_args}]

Function: returns the data read as a quoted string. A trailing newline character is deleted. The same control arguments are accepted as for the command, with the addition of -no_quote (-nq) to return the data unquoted. If the maximum number of characters N is not specified, the maximum segment size is assumed.

Operation: get_chars

Syntax: io get_chars switchname N {-control_args}

where:

N
is a decimal number greater than zero specifying the number of characters to read.

control_args
can be selected from the same list as described under the get_line operation.

Function: reads the next N characters from the file or device to which the I/O switch is attached. The disposition of the characters read is the same as described under the get_line operation; that is, they are written upon user_output if the -segment control argument is not specified, or stored in a segment if the -segment control argument is specified.

Syntax: [io get_chars switchname {N} {-control_args}]

Function: returns the data read as a quoted string. A trailing newline character is deleted. The same control arguments are accepted as for the command, with the addition of -no_quote (-nq) to return the data unquoted. If the maximum number of characters N is not specified, the maximum segment size is assumed.

Operation: put_chars

Syntax: io put_chars switchname {string} {-control_args}

where:

string
can be any character string.

control_args
can be any of the following:

-segment path {length}, -segment path {offset} {length},
-sm path {length}, -sm path {offset} {length}
specifies that the data for the output operation is to be found in the segment specified by pathname. The location and length of the data can optionally be described with offset and length parameters.

-nnl
specifies that a newline character is not to be appended to the end of the output string.

-nl
specifies that a newline character is to be appended to the end of the output line if one is not present.

-lines
specifies that offsets and lengths are measured in lines instead of characters.

The string parameter and the -segment control argument are mutually exclusive. If a string is specified, the contents of the string are output to the I/O switch. If the -segment control argument is specified, the data is taken from the segment specified by path, at the offset and length given. If offset is omitted, the beginning of the segment is assumed. If length is omitted, the entire segment is output.

If the I/O switch is attached to a device, this command transmits the characters from the string or the segment to the device. If the I/O switch is attached to an unstructured file, the data is added to the end of the file. The -nl control argument is the default on a put_chars operation: a newline character is added unless one is already present, or the -nnl control argument is specified.

Operation: read_record, read

Syntax: io read_record switchname N {-control_args}

where:

N
is a decimal integer greater than zero specifying the size of the buffer to use.

control_args
can be selected from the same list as described under the get_line operation.

Function: reads the next record from the file to which the I/O switch is attached into a buffer of length N. If the -segment control argument is not specified, the record (or the part of it that fits into the buffer) is printed on user_output. If the -segment control argument is specified, the record is stored in a segment as explained under the get_chars operation.

Syntax: [io read_record switchname {N} {-control_args}]

Function: returns the data read as a quoted string. A trailing newline character is deleted. The same control arguments are accepted as for the command, with the addition of -no_quote (-nq) to return the data unquoted. If the maximum number of characters N is not specified, the maximum segment size is assumed.

Operation: write_record, write

Syntax: io write_record switchname {string}
{-control_args}

where:

string
is any character string.

control_args
can be selected from the same list as described under the put_chars operation.

Function: adds a record to the file to which the I/O switch is attached. If the string parameter is specified, the record

is equal to the string. If the -segment control argument is specified, the record is extracted from the segment as described under the put_chars operation. In either case, the -nnl control argument is the default: a newline character is added only if the -nl control argument is specified. If the file is a sequential file, the record is added at the end of the file. If the file is an indexed file, the record's key must have been defined by a preceding seek_key operation.

Operation: rewrite_record, rewrite

Syntax: io rewrite_record switchname {string}
{-control_args}

where:

string
is any character string.

control_args
can be selected from the same list as described under the put_chars operation.

Function: replaces the current record in the file to which the I/O switch is attached. The new record is either the string parameter, or is taken from a segment, as described under the write_record operation. The current record must have been defined by a preceding read_record, seek_key, or position operation as follows:

read_record
current record is the last record read.

seek_key
current record is record with the designated key.

position
current record is the record preceding the record to which the file was positioned.

Operation: delete_record, delete

Syntax: io delete_record switchname

Function: deletes the current record in the file to which the I/O switch is attached. The current record is determined as in rewrite_record above.

Operation: position

Syntax: io position switchname type

where type can be one of the following:

bof
sets position to beginning of file.

eof
sets position to end of file.

forward N, fwd N, f N
sets position forward N records or lines (same as reverse -N).

reverse N, rev N, r N
sets position back N records (same as forward -N records).
any other numeric argument or pair of numeric arguments can be specified, but its function depends on the I/O module being used and cannot be implemented for all I/O modules.

Function: positions the file to which the I/O switch is attached. If type is bof, the file is positioned to its beginning, so that the next record is the first record (structured files), or so that the next byte is the first byte (unstructured files). If type is eof, the file is positioned to its end; the next record (or next byte) is at the end-of-file position. If type is forward or reverse the file is positioned forwards or backwards over records (structured files) or lines (unstructured files). The number of records or lines skipped is determined by the absolute value of N.

In the case of unstructured files, the next byte position after the operation is at a byte immediately following a newline character (or at the first byte in the file or at the end of the file); and the number of newline characters moved over is the absolute value of N.

If the I/O switch is attached to a device, only forward skips (where type is forward) are allowed. The effect is to discard the next N lines input from the device.

Syntax: [io position switchname type]

Function: attempts the specified position operation and returns true if it succeeds, false otherwise.

Operation: seek_key

Syntax: io seek_key switchname key

where key is a string of ASCII characters with $0 \leq$ length ≤ 256.

Function: positions the indexed file to which the I/O switch is attached to the record with the given key. The record's length is printed on user_output. Trailing blanks in the key are ignored.

If the file does not contain a record with the specified key, it becomes the key for insertion. A following write_record operation adds a record with this key.

Syntax: [io seek_key switchname key]

Function: returns true if the key exists, false otherwise.

Operation: read_key

Syntax: io read_key switchname

Function: prints, on user_output, the key and record length of the next record in the indexed file to which the I/O switch is attached. The file's position is not changed.

Syntax: [io read_key switchname {-control_arg}]

Function: returns the value of the key, quoted unless -no_quote (-nq) is specified.

Operation: read_length

Syntax: io read_length switchname

Function: prints, on user_output, the length of the next record in the structured file to which the I/O switch is attached. The file's position is not changed.

Syntax: [io read_length switchname]

Function: returns the length of the next record, in bytes.

Operation: control

Syntax: io control switchname order {args}

where:

order
is one of the orders accepted by the I/O module used in the attachment of the I/O switch.

args
are additional arguments dependent upon the order being issued and the I/O module being used.

Function: applies only when the I/O switch is attached via an I/O module that supports the control I/O operation. The exact format of the command line depends on the order being issued and the I/O module being used. For more details, refer to "Control Operations from Command Level" in the appropriate I/O module in the MPM Subroutines. If the I/O module supports the control operation and the paragraph just referenced does not appear, it can be assumed that only control orders that do not require an info_structure can be performed with the io command. This is true because the io_call command/active function uses a null info_ptr. (See the description of the iox_$control entry point and the I/O module's control operation, both in the MPM Subroutines.)

Syntax: [io control switchname order {args}]

Function: returns a value that depends on the I/O module and the order specified.

Operation: modes

Syntax: io modes switchname {string} {-control_arg}

where:

string
is a sequence of modes separated by commas. The string must not contain blanks.

control_arg
can be -brief or -bf.

Function: applies only when the I/O switch is attached via an I/O module that supports modes. The command sets only new modes specified in string, and then prints the old modes on user_output. Printing of the old modes is suppressed if the -brief control argument is used.

If the switch name is user_i/o, the command refers to the modes controlling the user's terminal. See the I/O module tty_ subroutine description in the MPM Subroutines for an explanation of applicable modes.

Syntax: [io modes switchname {string}]

Function: performs the specified modes operation and returns the old modes.

Operation: find_iocb

Syntax: io find_iocb switchname

Function: prints, on user_output, the location of the control block for the I/O switch. If it does not already exist, the control block is created.

Operation: look_iocb

Syntax: io look_iocb switchname

Function: prints, on user_output, the location of the control block for the I/O switch. If the I/O switch does not exist, an error is printed.

Syntax: [io look_iocb switchname]

Function: returns true if the specified iocb exists, false otherwise.

Operation: move_attach

Syntax: io move_attach switchname switchname2

where switchname2 is the name of a second I/O switch.

Function: moves the attachment of the first I/O switch (switchname) to the second I/O switch (switchname2). The original I/O switch is left in a detached state.

Operation: destroy_iocb

Syntax: io destroy_iocb switchname

Function: destroys the I/O switch by deleting its control block. The switch must be in a detached state before this command is used. Any pointers to the I/O switch become invalid.

Operation: print_iocb

Syntax: io print_iocb switchname

Function: prints, on user_output, all of the data in the control block for the I/O switch, including all pointers and entry variables.

Operation: attached

Syntax: [io attached switchname]

Function: returns true if the switch is attached, false otherwise.

Operation: opened

Syntax: [io opened switchname]

Function: returns true if the switch is open, false otherwise.

Operation: closed

Syntax: [io closed switchname]

Function: returns true if the switch is closed, false otherwise.

Operation: detached

Syntax: [io detached switchname]

Function: returns true if the switch is detached, false otherwise.

Operation: attach_desc

Syntax: [io attach_desc switchname {-control_arg}]

Function: returns the attach description of the switch, quoted unless -no_quote (-nq) is specified.

Operation: open_desc

Syntax: [io switchname open_desc {-control_arg}]

Function: returns the current open description (stream_input, etc.), quoted unless -no_quote (-nq) is specified.

Operation: io_module

Syntax: [io io_module switchname]

Function: returns the name of the I/O module through which the switch is attached.

Operation: valid_op

Syntax: [io valid_op switchname operation]

Function: returns true if the operation (put_chars, modes, etc.) is defined on the switch.

Operation: test_mode

Syntax: [io test_mode switchname mode]

Function: performs a modes operation and returns true if mode appears in the mode string, false if ^mode appears.

Operation: valid_mode

Syntax: [io valid_mode switchname mode]

Function: performs a modes operation and returns true if either mode or ^mode appears in the mode string, false otherwise.

Summary of Operations

```
Syntax:  io attach switchname modulename {args}
Syntax:  io detach switchname
Syntax:  io open switchname mode
Syntax:  io close switchname
Syntax:  io get_line switchname {N} {-control_args}
Syntax:  io get_chars switchname N {-control_args}
Syntax:  io put_chars switchname {string} {-control_args}
Syntax:  io read_record switchname N {-control_args}
Syntax:  io write_record switchname {string} {-control_args}
Syntax:  io rewrite_record switchname {string} {-control_args}
Syntax:  io delete_record switchname
Syntax:  io position switchname type
Syntax:  io seek_key switchname key
Syntax:  io read_key switchname
Syntax:  io read_length switchname
Syntax:  io control switchname order {args}
Syntax:  io modes switchname {string} {-control_arg}
Syntax:  io find_iocb switchname
Syntax:  io look_iocb switchname
Syntax:  io move_attach switchname switchname2
Syntax:  io destroy_iocb switchname
Syntax:  io print_iocb switchname
   Active function only:
Syntax:  [io attached switchname]
Syntax:  [io opened switchname]
Syntax:  [io closed switchname]
Syntax:  [io detached switchname]
Syntax:  [io attach_desc switchname {-control_arg}]
Syntax:  [io open_desc switchname {-control_arg}]
Syntax:  [io io_module switchname]
Syntax:  [io valid_op switchname]
Syntax:  [io test_mode switchname]
Syntax:  [io valid_mode switchname]
```

last_message (lm)

SYNTAX AS A COMMAND:

lm {address}

SYNTAX AS AN ACTIVE FUNCTION:

[lm {address}]

FUNCTION: returns the text of the last message received from the send_message command.

ARGUMENTS:
address can be any of the following to specify a mailbox:

-pathname path, -pn path
where path is the pathname of a mailbox. The mbx suffix is assumed.

STR
specifies a mailbox pathname of STR that contains a > or <.

Person.Project
specifies the Person_id and Project_id of a user whose mailbox is indicated.

NOTES: See the description of send_message, accept_message, last_message_sender, and last_message_time in this manual.

last_message_sender (lms)

SYNTAX AS A COMMAND:

lms {address}

SYNTAX AS AN ACTIVE FUNCTION:

[lms {address}]

FUNCTION: returns the sender of the last message received (from the send_message command) in the form "Person_id.Project_id" (e.g., RSJones.Demo).

ARGUMENTS:
address can be any of the following to specify a mailbox:

-pathname path, -pn path
where path is the pathname of a mailbox. The mbx suffix is assumed.

STR
specifies a mailbox pathname of STR that contains a > or <.

Person.Project
specifies the Person_id and Project_id of a user whose mailbox is indicated.

NOTES: The user is cautioned against using this active function when in polite mode. In polite mode, the system holds all messages until the user finishes typing a line (i.e., until the carriage is at the left margin). Therefore, it is possible that while the user is sending a message, the user's process can receive another message from a different user -- a message not yet seen. By using the last_message_sender active function in such a situation, the user can inadvertently attribute a message to the "wrong" person.

See the descriptions of send_message, accept_message, last_message, and last_message_time in this manual.

last_message_sender (lms)

EXAMPLES: Assume that a user just received the following message:

```
From RSJones.Demo 11/19/76 1231.7  mst Fri:  need access to
test xyz
```

A reply can be sent as follows:

```
sm [lms] sorry for the oversight, you have access now.
```

last_message_time (lmt)

SYNTAX AS A COMMAND:

lmt {address}

SYNTAX AS AN ACTIVE FUNCTION:

[lmt {address}]

FUNCTION: returns the time that the last message (from the send_message command) was received.

ARGUMENTS:
address can be any of the following to specify a mailbox:

-pathname path, -pn path
where path is the pathname of a mailbox. The mbx suffix is assumed.

STR
specifies a mailbox pathname of STR that contains a > or <.

Person.Project
specifies the Person_id and Project_id of a user whose mailbox is indicated.

NOTES: See the descriptions of send_message, accept_message, last_message, and last_message_sender in this manual.

SYNTAX AS A COMMAND:

```
ln str
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[ln str]
```

FUNCTION: returns an integer representing the number of characters in str.

ARGUMENTS:

str
is any string of alphanumeric characters. If str contains blanks or other command language characters, it must be enclosed in quotes.

EXAMPLES:

```
! string [ln "A multiple word string"]
22
```

The following example from an exec_com segment tests for a string that is greater than 27 characters.

```
&if [nless [ln &1] 27] &then &goto OK
&print Entry name too long.  &1.info
&quit
&label OK
ec exec_com2 &1.info
```

SYNTAX AS A COMMAND:

```
less strA strB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[less strA strB]
```

FUNCTION: returns true if strA is less than strB according to ASCII collating sequence; otherwise it returns false.

NOTES: The strings are compared character by character according to their ASCII code value (i.e., if the first character in each string has the same ASCII code value, compare the second character; if their values are identical, compare the third character; etc.). See nless and ngreater in this manual for a way to compare numeric strings.

line_length (ll)

SYNTAX AS A COMMAND:

ll maxlength

FUNCTION: allows the user to control the maximum length of a line output to the device that the user's process is connected to through the user_output I/O switch. This device is usually the user's terminal.

ARGUMENTS:

maxlength
is a positive decimal number greater than 4 that specifies the maximum number of characters that can henceforth be printed on a single line using the I/O switch named user_output. In most cases, this is the maximum length of a line of output printed at the user's terminal.

SYNTAX AS A COMMAND:

lk path1_1_ {path2_1_ ... path1_n_ path2_n_}

FUNCTION: causes a storage system link with a specified name to be created in a specified directory pointing to a specified segment, directory, or link.

ARGUMENTS:

path1_i_
specifies the pathname of the storage system entry to which path2_i_ is to point. The star convention is allowed. The pathnames must be specified in pairs.

path2_i_
specifies the pathname of the link to be created. If omitted (in the final argument position of a command line only), a link to path1_i_ is created in the working directory with the entryname portion of path1_i_ as its entryname. The equal convention is allowed.

ACCESS REQUIRED: The user must have append permission for the directory in which the link is to be created.

NOTES: For a discussion of links, see "Directory Contents" in the MPM Reference Guide.

Entrynames must be unique within the directory. If the creation of a specified link would introduce a duplication of names within the directory, and if the old entry has only one name, the user is interrogated whether to delete the the entry bearing the old instance of the name. If the answer is "no", the link is not created. If the old entry has multiple names, the conflicting name is removed and a message to that effect is issued to the user. In either case, since the directory in which the link is to be created is being changed, the user must also have modify permission for that directory.

EXAMPLE: The command line:

```
! lk >my_dir>beta alpha >dictionary>grammar
```

creates two links in the working directory, named alpha and grammar; the first points to the segment beta in the directory >my_dir and the second points to the segment grammar in the directory >dictionary.

SYNTAX AS A COMMAND:

```
links star_names {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[links star_names {-control_arg}]
```

FUNCTION: returns the entrynames or absolute pathnames of links that match one or more star names.

ARGUMENTS:

star_names
is a star name to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per link is returned; i.e., if a link has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by links is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

EXAMPLES:

```
! pwd
  >udd>Apple>Jones
! ls -lk

  Links = 3

  prog.temp2                    >udd>Apple>Jones>temp_seg_2
  prog.temp1                    >udd>Apple>Jones>temp_seg_1
  junk                          >udd>Apple>Jones>empty_seg

! string [links **]
  junk prog.temp1 prog.temp2
! string [links *]
  junk
! string [links **.temp*]
  prog.temp1 prog.temp2

! links **
  junk prog.temp1 prog.temp2
! links *.temp* -absp
  >udd>Apple>Jones>temp_seg_1 prog.temp1 >udd>Apple>Jones>
  temp_seg_1>prog.temp1
```

SYNTAX AS A COMMAND:

ls {entrynames} {-control_args}

FUNCTION: prints information about entries contained in a single directory. A large selection of control arguments enable the user to specify the directory to be listed, which entries are to be listed, the amount and kind of information to be printed for each entry, and the order in which the entries are to be listed.

ARGUMENTS:

entrynames
are the names of entries to be listed. If entrynames are specified, only entries having at least one name matching an entryname argument are listed. The star convention is allowed. If no entryname argument is given, all entries (of the types given by control arguments) in the directory are listed. A pathname can be specified instead of an entryname. In this case, entries matching the entryname portion of the pathname, in the directory specified by the directory portion of the pathname, are listed. See the description of the -pathname control argument for restrictions on the use of this feature.

Except where otherwise noted in the descriptions of the control arguments ("Control Arguments for the list Command" below), the entrynames and control_args arguments can appear anywhere on the command line.

Control Arguments for the list Command

The control arguments for the list command are described in detail on the following pages. For convenience, these arguments have been arranged in categories according to the function they perform. The categories and their respective control arguments are listed below (detailed descriptions follow the list, in the same order):

directory
-pathname path, -pn path

entry type
-segment, -sm

```
   -multisegment_file, -msf
   -file, -f
   -directory, -dr
   -branch, -br
   -link, -lk
   -all, -a

columns
   -mode, -md
   -length, -ln
   -record, -rec
   -name, -nm
   -date_time_used, -dtu
   -date_time_entry_modified, -dtem
   -date_time_contents_modified, -dtcm
   -count, -ct
   -link_path, -lp

totals/header line
   -total, -tt
   -no_header, -nhe

multiple-name entries
   -primary, -pri
   -match

entry order
   -sort XX, -sr XX
   -reverse, -rv

entry exclusion
   -exclude entryname, -ex entryname
   -first N, -ft N
   -from D, -fm D
   -to D

output format
   -brief, -bf
   -short, -sh
```

NOTES: If the list command is invoked without any arguments, it lists all segments and multisegment files in the working directory, printing the name(s), access mode, and length of each. Segments and multisegment files are listed separately (segments first), each preceded by a line giving the total entries of that type and the sum of their lengths. (This line is referred to as the totals information or the header.) Within each entry type, entries are listed in the order in which they are found in the directory.

EXAMPLES: The following example shows the results of invoking the list command without any arguments (the line typed by the user is preceded by an exclamation mark):

```
! list

   Segments = 8, Lengths = 41.

   r w   10  new_code_info.runout
   rew    9  new_code_info.runoff
   r w    3  work.pl1
   r w    7  work.list
   re     2  work
   r w    1  print.ec
   r w    1  output_file
   r      8  data_base

   Multisegment-files = 1, Lengths = 334.

   r w  334  info_segs
```

Notice that the information about the entries is arranged in columns without column headings. The set of columns printed by the list command depends on the control arguments specified by the user and the type of entry being listed.

There are four entry types: segments, multisegment files, directories, and links. Segments and multisegment files are referred to collectively as files; segments, multisegment files, and directories are referred to collectively as branches. The set of possible columns is different for branches and links. For branches, the set of possible columns and their order (from left to right) is: modification date, date and time used, access mode, size, names, and number of names; for links: date and time entry modified, names, number of names, and link pathname. The modification-date column contains either the date and time the entry was modified or the date and time the contents were modified, and the size column contains either records used or length (in records) computed from the bit count, as specified by control arguments. Unless otherwise specified by control arguments, the items printed for branches are: access modes, length, and names; for links: names and link pathname.

The list command offers the user precise control over the command output. The various control arguments specify exactly

what is to be printed. Most users will find that the following subset of list command control arguments allows them to adequately define the desired information.

-file, -f
lists information about files. This is the default.

-directory, -dr
lists information about directories.

-link, -lk
lists information about links.

-name, -nm
prints the names column, giving primary and any additional names of each entry.

-date_time_entry_modified, -dtem
prints the date and time the entry was last modified (e.g., by the changing of attributes such as names, ACL, or bit count).

-primary, -pri
prints only the primary name (in the names column) of each entry.

-sort XX, -sr XX
sorts the entries, within each entry type, according to the column name specified by XX. (The column names and their sorting order are described under "Entry Order" below.)

-total, -tt
prints only the heading line (totals information) for each entry type specified; this line gives the total number of entries and the sum of their sizes.

Detailed information on each of the above control arguments is given in "Control Arguments for the list Command" below.

DIRECTORY

If no directory is specified, the working directory is assumed. The list command can list only one directory at a time, and it is an error to specify more than one directory to be listed.

-pathname path, -pn path
causes entries in the directory specified by path to be listed.

The directory to be listed can also be specified by giving a pathname instead of an entryname, as described earlier. The difference between the two methods of specifying a directory is that the entire pathname after the -pathname control argument is taken to be that of a directory whose entries are to be listed, while a pathname not preceded by the -pathname control argument is separated into its directory and entryname portions, and the former specifies the directory while the latter specifies the entries within it that are to be listed.

ENTRY TYPE

If no control arguments from this category are specified, the -file control argument is assumed. More than one of the following control arguments can be specified.

-segment, -sm
lists information about segments.

-multisegment_file, -msf
lists information about multisegment files.

-file, -f
lists information about files (i.e., segments and multisegment files, in that order).

-directory, -dr
lists information about directories.

-branch, -br
lists information about branches (i.e., segments, multisegment files, and directories, in that order).

-link, -lk
lists information about links.

-all, -a
lists information about all entry types in the following order: segments, multisegment files, directories, and links.

COLUMNS

If no control arguments from this category are specified, the access-mode, length, and names columns (in that order) are printed for branches and the names and link-path columns (in that order) are printed for links. More than one of the control arguments listed below can be given in a single invocation of the list command. When the -brief, -mode, -record, -length, or -name control arguments are specified, only the names column plus those columns explicitly selected by control arguments are printed.

The user is given a choice as to what can be printed in two of the columns for branches (size and modification date). For size, the user can choose between length computed from the bit count and a count of records used. For modification date, the user can choose between the date and time the entry was modified (e.g., by the changing of attributes such as names, ACL, or bit count) and the date and time the contents of the segment or directory were modified.

If sorting by a size or modification date is specified, the above choices also apply to sorting, and the specifications of what to sort on and what to print must be consistent. For example, it is not possible to print length computed from bit count while sorting on records used.

Because of the way the information is maintained by the storage system, the records-used, date-time-contents-modified, and date-time-used values are more expensive to obtain than the other items printed by the list command. It is recommended that these values not be used for printing or sorting except when absolutely necessary. Less expensive alternatives are provided that should be suitable in most cases (e.g., length computed from bit count, and date and time the entry was modified).

The names column is printed in every invocation of the list command except when the user explicitly requests only totals information (see "Totals/Header Line" below).

-mode, -md
prints the access-mode column.

-length, -ln
prints current length computed from the bit count. This control argument is inconsistent with the -record control argument. The -length argument, which is the less expensive of the two, is the default.

-record, -rec
prints the records used. This argument is inconsistent with the -length control argument. The -record control argument is the more expensive of the two.

-name, -nm
prints the names column, giving the primary name and any additional names of each entry.

-date_time_used, -dtu
prints the date and time the entry was last used.

-date_time_entry_modified, -dtem
prints the date and time the entry was last modified. (e.g., by the changing of attributes such as names, ACL, or bit count). This argument is inconsistent with the -date_time_contents_modified control argument. This argument is the less expensive of the two.

-date_time_contents_modified, -dtcm
prints the date and time the contents of the segment or directory were last modified. This argument is inconsistent with the -date_time_entry_modified control argument; only one of the two may be given. This argument is the more expensive of the two.

-count, -ct
prints the count column, which gives the total number of names for entries that have more than one name.

-link_path, -lp
prints the link-path column.

TOTALS/HEADER LINE

If no control arguments from this category are specified, both totals and detailed information are printed.

-total, -tt
prints only the heading line (totals information) for each entry type specified; this line gives the total number of entries and the sum of their sizes.

-no_header, -nhe
omits all heading lines.

MULTIPLE-NAME ENTRIES

The control arguments in this category are applicable only to entries that have more than one name. If no control arguments from this category are specified, all of the names of the specified entries are printed in the names column.

-primary, -pri
prints, in the names column, only the primary name of each entry. This control argument does not suppress the printing of any other columns; it merely suppresses the printing of secondary names.

-match
prints, in the names column, only those names that match one of the given entrynames.

ENTRY ORDER

If no control arguments from this category are specified, entries are printed in the order in which they are found in the directory.

-sort XX, -sr XX
sorts entries, within each entry type, according to the sort column XX where XX can be one of the following:

-name, -nm
sort entries by primary name, according to the standard ASCII collating sequence.

-length, -ln
sort entries by length computed from the bit count, largest first. This argument is inconsistent with the -record control argument.

-record, -rec
sort entries by records used, largest first. This argument is inconsistent with the -length control argument. If this argument is specified, and the size column is being printed, the value printed in that column is records used, rather than length.

-mode, -md
sort entries by access mode in the following order: null, r (or s), rw (or sm), re, rew (or sma). (This order is the result of sorting by the internal representation of the mode.)

-date_time_entry_modified, -dtem
sort entries by the date and time the entry was last modified, most recent first. This argument is inconsistent with the -dtcm control argument. If the -dtem control argument is specified and no sort key follows the -sort control argument, this argument is implied as the default sort key.

-date_time_contents_modified, -dtcm
sort entries by the date and time the contents of the entry were last modified, most recent first. This argument is inconsistent with the -dtem control argument. If the -dtcm control argument is specified and no sort key follows the -sort control argument, this argument is implied as the default sort key.

-date_time_used, -dtu
sort entries by the date and time used, most recent first.

-count, -ct
sort entries by number of names, most names first.

It is not necessary for a column to be printed in order to sort on it, but note the restrictions described earlier regarding sorting on and printing the modification-date and size columns.

If the sort column XX is omitted, the default sorting column is determined as follows: if no date column is being printed, sort by primary name; if only one of the date columns is being printed, sort by that date; if both the modification-date and date-time-used columns are being printed, sort by the modification-date column.

Links can only be sorted by the name, modification-date, or count columns. If sorting by any other column is specified, links are printed in the order in which they are found in the directory, while branches (if also being listed) are sorted by the specified column. (See "Notes" below.)

-reverse, -rv
prints entries in the reverse of the order in which they are

found in the directory. If the -sort control argument is also specified, the specified sort is reversed.

ENTRY EXCLUSION

The following control arguments limit the amount of output produced by excluding entries according to either name or date or by setting an upper limit on the number of entries listed.

-exclude entryname, -ex entryname
does not list any entries that have a name that matches the specified entryname. The star convention is allowed in entryname.

To exclude more than one entryname, the user must give an -exclude control argument for each one of them. The entrynames specified in all -exclude control arguments and any names specified in the entryname arguments (explained on the first page of the list command description) operate together to limit the entries that are listed. All entries that have at least one name that matches any one of the entrynames specified in the -exclude control arguments are excluded from the listing. From the entries that remain, those matching any of the entryname arguments are listed; if no entryname arguments are specified, all the remaining entries are listed. (See "Examples" below.)

-first N, -ft N
lists only the first N entries (after sorting, if specified) of each entry type being listed. The heading lines contain the totals figures for all entries that would have been listed if the -first control argument had not been specified. This control argument is useful to avoid tying up a terminal by listing a large directory, when only the first few entries are of interest.

The following two arguments exclude entries on the basis of date. The date used in this comparison is the modification-date value in all cases except when the only date column being printed or sorted on is the date-time-used column. If no date column is being printed, the date-time-entry-modified value is used.

-from D, -fm D
does not list any entries that have a date value (selected as described above) before the one specified by D.

-to D
does not list any entries that have a date value (selected as described above) after the one specified by D.

The D value after the -from or -to control arguments must be a string acceptable to the convert_date_to_binary_ subroutine, described in the MPM Subroutines. If the date-time string contains spaces, the string must be enclosed in quotes. The D value should specify both a date and a time; if only a date is specified, the convert_date_to_binary_ subroutine uses the current time of day as the default time.

If both the -from and -to control arguments are specified, the -from D value must be earlier than the -to D value.

OUTPUT FORMAT

If no control argument from this category are specified, the output format of the list command is not changed.

-brief, -bf
if just totals information is being printed, this control argument causes the totals information for all selected entry types to be abbreviated and printed on a single line. Otherwise, it suppresses the printing of the default columns when they are not explicitly named in control arguments. For example, typing:

```
! ls -dtu -brief
```

causes the names and date-time-used columns, but not the access-mode and length columns, to be printed.

-short, -sh
prints link pathnames starting two spaces after their entrynames, instead of aligning them in column position 35.

NOTES: The obsolete name for a modification date (date_time_modified, dtm) is accepted, in both the control argument and sort key form, as a synonym for the date-time-entry-modified value.

Links do not have a date-time-contents-modified value. If links are being listed and either modification-date value is specified for printing, sorting, or entry exclusion (using the -from and -to control arguments), the date-time-entry-modified value of links is used.

Examples

The command line:

```
! ls -pri -ct
```

lists all files in the working directory (the default directory); the names column contains only the primary names of all entries; the total number of names (for those entries having more than one name) is printed after the primary name. In addition to the names column, the access-mode and length columns are printed.

The command line:

```
! ls -ex *.*
```

lists all the files in the working directory having other than two-component names, printing the three default columns (access mode, length, and names).

The command line:

```
! ls -sm *.* -ex *.pl1
```

lists all the segments in the working directory having two-component names whose second component is not pl1, printing the three default columns.

The command line:

```
! ls -dtem -sr
```

lists all files in the working directory, sorted by the date-time-entry-modified column (the default sort key since the user specifically requested that date column). The date-time-entry-modified column is printed in addition to the three default columns.

The command line:

```
! ls -nm -sr dtm
```

lists all files in the working directory, sorted by the date-time-entry-modified value. Only the names column is printed. Note the use of dtm as a synonym for dtem.

The command line:

```
! ls -sm -nm -pri -nhe
```

lists only the primary name of each segment in the working directory without printing the heading line or any blank lines. This combination of arguments, together with the file_output command, is useful for generating a file that contains the primary names of a selected set of entries.

The command line:

```
! ls -md -pri
```

lists the access mode and primary name of each file in the working directory.

The command line:

```
! ls -tt -to "7/1/75 000.0" -dtu -rec
```

prints the totals (number of entries and total records used) for all files that have not been used since the end of June 1975. Notice that the -dtu control argument is used to specify that the -to date refers to the date and time used.

list_abs_requests (lar)

SYNTAX AS A COMMAND:

lar {path} {-control_args}

FUNCTION: lists requests in the absentee queues.

ARGUMENTS:

path
is the pathname of a request to be listed. The star convention is allowed. Only requests matching this pathname are selected. If the path argument is not specified, all pathnames are selected. Also see the -entry control argument below.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
prints the full pathname of each selected request, rather than just the entryname.

-admin {User_id}, -am {User_id}
selects the requests of all users, or of the user specified by User_id. If the -admin control argument is not specified, only the user's own requests are selected. See "Notes" below.

-all, -a
searches all queues and prints the totals for each non-empty queue whether or not any requests are selected from it. If the -all control argument is not specified, nothing is printed for queues from which no requests are selected. This control argument is incompatible with the -queue control argument.

-brief, -bf
prevents the printing of the state and the comment of the request. If the -brief control argument is not specified, these items are printed. This control argument is incompatible with the -long and -total control arguments.

-deferred_indefinitely, -dfi
selects only requests that are deferred indefinitely. Such requests are not run until the operator releases them.

-entry STR, -et STR
selects only requests whose entrynames match STR. The star convention is allowed. Directory portions of request pathnames are ignored when selecting requests. This control

argument is incompatible with the path argument.

-foreground, -fg
searches only the foreground queue, and prints the totals for this queue, whether or not any requests are selected from it. Also, see the -queue control argument.

-id ID
selects only requests whose identifier matches the specified ID.

-immediate, -im
selects only requests that can be run immediately upon reaching the heads of their respective queues. This does not include requests deferred indefinitely, requests deferred until a specific time, or requests that have reached the head of the queue and have been deferred by the system because their CPU time limits are higher than the maximum for the current shift. It does include requests deferred because of load control or resource unavailability, because those conditions could change at any time. Also, see the -position control argument.

-long, -lg
prints all of the information pertaining to an absentee request including the long request identifier and the full pathname. If this control argument is omitted, only the short request identifier, entryname, state and comment, if present, are printed. The -long, -brief, and -total control arguments are incompatible.

-long_id, -lgid
prints the long form of the request identifier. If this or the -long control argument is not specified, the short form of the request identifier is printed.

-position, -psn
prints the position within its queue of each selected request. When used with the -total control argument, it prints a list of all the positions of the selected requests. When used with the -immediate control argument, it considers only immediate requests when computing positions. See "Notes" below.

-queue N, -q N
searches only queue N, and prints the totals for that queue, whether or not any requests are selected from it. If the -queue control argument is not specified, all queues are searched but nothing is printed for queues from which no requests are selected. For convenience in writing exec_coms and abbreviations, the word "foreground" or "fg" following the

-queue control argument performs the same function as the -foreground control argument. This control argument is incompatible with the -all control argument.

-resource {STR}, -rsc {STR}
selects only requests having a resource requirement. If STR is specified, only requests whose resource descriptions contain that string are selected. This control argument also causes the resource descriptions of the selected requests to be printed, even when the -long control argument is not specified. See the reserve_resource command in this manual for a description of resource description specification. If this control argument is not specified, the request type "printer" is assumed.

-sender STR
specifies that only requests from sender STR should be listed. One or more request identifiers must also be specified. In most cases, the sender is an RJE station identifier.

-total, -tt
prints only the total number of selected requests and the total number of requests in the queue plus a list of positions if the -position control argument is also specified. If the queue is empty, it is not listed. This control argument is incompatible with the -long and -brief control arguments.

-user User_id
selects only requests entered by the specified user. See "Notes" below.

ACCESS REQUIRED: The user must have o access to the queue(s) to invoke lar. The user must have r extended access to the queue(s), in order to use the -admin, -position, or -user control arguments, since it is necessary to read all requests in the queue(s) in order to select those entered by a specified user or to compute the positions of the selected requests.

NOTES: All queues are searched for the user's requests; the request identification, entryname, state, and comment, if present, of each request is printed. If no arguments are specified, only the user's own requests are selected for listing. Nothing is printed for queues from which no requests are selected. See also the enter_abs_request command in this manual.

When a user name is specified, with either the -admin or -user control arguments, then proxy requests are selected if either the user who entered the request, or the proxy user on whose behalf it was entered, matches the specified user name.

The entry name specified after the -entry control argument, the entry portion of the pathname argument, and the RJE station name specified after the -sender control argument, may each be starnames.

The User_id arguments specified after the -admin or -user may have any of the following forms:

Person_id.Project_id	matches that user only
Person_id.*	matches that person on any project
Person_id	same as Person_id.*
*.Project_id	matches any user on that project
*.Project_id	same as *.Project_id
.	same as -admin with no User_id following it

EXAMPLES: To find out what absentee requests he has in all queues, the user types:

```
! lar

Queue 1:  1 request. 3 total requests.
211401 translate.absin

Queue 3:  2 requests. 6 total requests.
211421 tasks.absin
211463 bindings.absin
```

To get all information about his absentee requests in queue 1, he types:

```
! lar -q 1 -lg
Queue 1:  1 request. 6 total requests.

Request ID:             781211002253.583765
Time queued:            12/10/78  1922.8 est Sun
Input segment:          >udd>sys>Jones>monthly_status.absin
Restartable:            yes
Deferred time:          01/07/79  1922.8 est Sun
Output file:            >udd>sys>cp>Jones>monthly_status.
                        absout
```

list_abs_requests (lar)

To find out the total number of absentee requests he has in all queues, he types:

```
! lar -tt -a

   Queue 1:  2 requests. 15 total requests.

   Queue 3:  0 requests. 39 total requests.
```

Note that queue 2 being empty, does not have a total line listed.

list_accessible (lac)

SYNTAX AS A COMMAND:

lac {path} {User_id} {-control_args}

FUNCTION: scans a directory and lists segments, multisegments, files, and directories with a specified access for a specified User_id.

ARGUMENTS:

path
is the pathname of the directory to be scanned. If path is omitted or -wd is specified, the working directory is scanned.

User_id
is an access name. It can have null components. The star convention for access names is allowed. See the description of set_acl in this manual. If User_id is omitted, the User_id of the calling process with a star tag is assumed.

CONTROL ARGUMENTS:
If no control arguments are specified, all the segments and directories to which the named user(s) has nonnull access are listed.

-dir_mode STR
lists directories to which the named user(s) has any of the modes specified in STR, where STR can be any or all of the letters sma.

-seg_mode STR
lists segments to which the named user(s) has any of the modes specified in STR, where STR can be any or all of the letters rew.

ACCESS REQUIRED: The user must have status (s) permission on the directory.

NOTES: If there can be more than one User_id (i.e., the specified User_id has null components), the modes for each matched User_id and the matched User_id are listed on a per entry basis.

list_accessible (lac)

EXAMPLES:

```
! lac >udd>work>Smith

   r        Smith.profile
   rew      Smith.con_msgs
   s        index
   r        mpm_cont_pull.ec
   r        start_up.ec
   re       naming.runoff
   sma      work
   r        sr.runoff


! lac >udd>work>Smith -dir_mode m

   r        Smith.profile
   rew      Smith.con_msgs
   r        mpm_cont_pull.ec
   r        start_up.ec
   r        naming.runoff
   sma      work
   r        sr.runoff

! lac -wd .Group.

   r        Smith.profile                  Smith.Group.*
   rew      Smith.con_msgs                 Smith.Group.*
   r        Smith.con_msgs                 Jones.Group.*
   s        index                          *.Group.*
   sma      work                           Smith.Group.*
   s        work                           Jones.Group.*
   s        work                           *.Group.*
```

SYNTAX AS A COMMAND:

la {path} {User_ids} {-control_args}

SYNTAX AS AN ACTIVE FUNCTION:

[la {path} {User_id} {-control_args}]

FUNCTION: lists the access control lists (ACLs) of segments, multisegment files, and directories.

ARGUMENTS:

path
is the pathname of a segment, multisegment file, or directory. If it is -wd, -working_dir, or omitted, the working directory is assumed. If it is omitted, no User_ids can be specified. The star convention can be used.

User_ids
are access control names that must be of the form Person_id.Project_id.tag. All ACL entries with matching names are listed. If User_id is omitted, the entire ACL is listed.

CONTROL ARGUMENTS:

-ring_brackets, -rb
lists the ring brackets. This control argument is not valid is list_acl is invoked as an active function.

-brief, -bf
suppresses the message "User name not on ACL of path." If list_acl is invoked as an active function, and User_id is not on the ACL, the null string is returned regardless of the -brief control argument.

-directory, -dr
lists the ACLs of directories only. The default is segments, multisegment files, and directories. (See "Notes" below.)

-segment, -sm
lists the ACLs of segments and multisegment files only.

ACCESS REQUIRED: The user must have status (s) permission on the directory.

NOTES: The -directory and -segment control arguments are used to resolve an ambiguous choice that can occur when path is a star name.

If the list_acl command is invoked with no arguments, it lists the entire ACL of the working directory.

For a description of ACLs, see "Access Control" in the MPM Reference Guide. Ring brackets are discussed under "Intraprocess Access Control" in the MPM Reference Guide. For a description of the matching strategy, refer to set_acl in this manual.

EXAMPLES: The command line:

```
! la notice.runoff .Faculty. Doe
```

lists, from the ACL of notice.runoff, all entries with Project_id Faculty and the entry for Doe.*.*.

The command line:

```
! la *.pl1 -rb
```

lists the whole ACL and the ring brackets of every segment in the working directory that has a two-component name with a second component of pl1.

The command line:

```
! la -wd -rb .Faculty. *.*.*
```

lists access modes and ring brackets for all entries on the working directory's ACL whose middle component is Faculty and for the *.*.* entry.

list_daemon_requests (ldr)

SYNTAX AS A COMMAND:

ldr {path} {-control_args}

FUNCTION: lists requests in the I/O daemon queues. The request identifier and entryname of each request is printed.

ARGUMENTS:

path
is the pathname of a request to be listed. The star convention is allowed. Only requests matching this pathname are selected. If the path argument is not specified, all pathnames are selected. This argument is incompatible with the -entry control argument.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
prints the full pathname of each selected request, rather than just the entryname.

-admin {User_id}, -am {User_id}
selects the requests of all users, or of the user specified by User_id. This control argument is incompatible with -user. If the -admin control argument is not specified, only the user's own requests are selected. See "Notes" and "Access Required" below.

-all, -a
searches all queues and prints the totals for each non-empty queue whether or not any requests are selected from it. If the -all control argument is not specified, the default queue is searched. This control argument is incompatible with the -queue control argument.

-brief, -bf
suppresses the printing of the state of the request. This control argument is incompatible with the -long and -total control arguments.

-entry STR, -et STR
selects only requests whose entrynames match STR. The star convention is allowed. Directory portions of request pathnames are ignored when selecting requests. This control argument is incompatible with the path argument.

-id ID
selects only requests whose identifier matches the specified ID.

-immediate, -im
selects only requests that can be run immediately and skips requests deferred by the I/O daemon.

-long, -lg
prints all of the information about each selected request including the long request identifier and the full pathname. If this control argument is not specified, only the short request identifier, entryname, and state are printed. This control argument is incompatible with the -brief and -total control arguments.

-long_id, -lgid
prints the long form of the request identifier. If this or the -long control argument is not specified, the short form of the request identifier is printed.

-position, -psn
prints the position within its queue of each selected request. When used with the -total control argument, it prints a list of all the positions of the selected requests. See "Notes" and "Access Required" below.

-queue N, -q N
searches queue N, and prints the totals for that queue, whether or not any requests are selected from it. If the -queue control argument is not specified, the default queue for the request type is searched. This control argument is incompatible with the -all control argument.

-request_type STR, -rqt STR
specifies that the request moved is found in the queue for the request type identified by STR. If this control argument is not specified, the default request type is "printer". Request types can be listed by the print_request_types command.

-total, -tt
prints only the total number of selected requests and the total number of requests in the queue plus a list of positions, if the -position control argument is specified. If the queue is empty, it is not listed. This control argument is incompatible with the -long and -brief control arguments.

list_daemon_requests (ldr)

-user User_id
selects only requests entered by the specified user. This control argument is incompatible with -admin. See "Notes" and "Access Required" below.

ACCESS REQUIRED: The user must have o access to the queue(s) to invoke ldr. The user must have r extended access to the queue(s), in order to use the -admin, -position, or -user control arguments, since it is necessary to read all requests in the queue(s) in order to select those entered by a specified user or to compute the positions of the selected requests.

NOTES: The User_id arguments specified after -admin or -user can have any of the following forms:

Person_id.Project_id	matches that user only
Person_id.*	matches that person on any project
Person_id	same as Person_id.*
*.Project_id	matches any user on that project
.Project_id	same as *.Project_id
.	same as -admin with no User_id following it

The state is printed only if it is deferred and the -brief argument is not specified.

EXAMPLES: To find out what dprint requests are in the default queue for the default request type, the user types:

```
! ldr

   Queue 3:  3 requests. 6 total requests.

   211401 translate.list
   211421 ldr.runoff
   211463 Jones.profile
```

list_daemon_requests (ldr)

To get all information about dprint requests in queue 1 of the default request type, the user types:

```
! ldr -lg -q 1

   Queue 1:     2 total requests (1 deferred).

   Request ID:              781120185551.598368
   Time queued:             11/20/78  1155.8 mst Mon
   Pathname:                >udd>Tech>Jones>f.symbol
   State:                   deferred
   Copies:                  1
   Delete:                  no
   Heading:                 Jones
   Destination:             Tech

   Request ID:              781121173121.663091
   Time queued:             11/21/78  1031.3 mst Tue
   Pathname:                >udd>Tech>Jones>f.symbol
   Copies:                  1
   Delete:                  no
   Heading:                 Jones
   Destination:             Tech
```

To find out the total number of requests in each queue of the request type "punch", the user types:

```
! ldr -tt -a -rqt punch

   Queue 1:  0 requests. 15 total requests.
   Queue 3:  2 requests. 39 total requests.
```

Note that queue 2, being empty, does not have a total line listed.

SYNTAX AS A COMMAND:

```
lh topic1 topic2 ... topicN {-control_args}
```

FUNCTION: displays the names of all info segments pertaining to a given topic. Topics are specified by arguments to the list_help command. An info segment is considered to pertain to a given topic if the topic name appears in (i.e., is a substring of) the info segment name.

ARGUMENTS:

topic1 through topicN
are topics to be searched for.

CONTROL ARGUMENTS:

-brief, -bf
does not display the alternate names on the info segments. The default is to display them.

-all, -a
displays the names of all info segments. The default is to display the names of only those info segments whose names match the topics specified.

-pathname path, -pn path
specifies the pathname of a directory to search for applicable segments. The default is to search the directories in the info_segments search list. See "Notes on Search List" below.

NOTES ON SEARCH LIST The check_info_segs command uses the "info_segments" search list that has the synonyms "info_segs" and "info". The default "info_segments" search list is:

```
>doc>iml_info
>doc>info
```

These directories contain info segments provided by the site and those supplied with the system. Type "print_search_paths info_segments" to see what the current "info_segments" search list is. For more information about search lists, see the search facility commands, and in particular, the add_search_paths command description in this manual.

list_help (lh)

EXAMPLES:

```
!  list_help fortran pl1

   fortran_io.changes
   fortran_io.status
   pl1.changes (pl1_changes)
   pl1.status (pl1_status)
   display_pl1io_error (dpe)
   fortran_abs (fa)
   fortran_common.gi
   fortran.gi (ft.gi)
   fortran (ft)
   pl1
   pl1_abs (pa)
   pl1_new_features (pl1_new)
   set_fortran_common (sfc)
   fort_options.gi (fortran_options.gi)
   fortran_optimizer.gi
   fortran_io.gi
```

list_iacl_dir (lid)

SYNTAX AS A COMMAND:

lid {path} {User_ids} {-control_args}

SYNTAX AS AN ACTIVE FUNCTION:

[lid {path} {User_ids} {-control_args}]

FUNCTION: lists some or all of the entries on a directory initial access control list (directory initial ACL) of a specified directory.

ARGUMENTS:

path
specifies the directory in which the directory initial ACL should be listed. The star convention is allowed. If path is -wd, -working_dir, or omitted, then the working directory is assumed. If path is omitted, no User_ids can be specified.

User_ids
are access control names of the form Person_id.Project_id.tag. All access names that match the given User_ids are listed. If no User_id is specified, the entire initial ACL is listed.

CONTROL ARGUMENTS:

-ring N, -rg N
identifies the ring number whose directory initial ACL should be listed. If present, it must be followed by N (where $0 \leq N \leq 7$). This argument can appear anywhere on the line and affects the whole line. If this argument is not specified, the directory initial ACL of the user's current ring is listed.

-brief, -bf
suppresses the message "User name not on ACL of path." If lid is invoked as an active function, and the supplied User_id is not on the initial ACL, the null string is returned regardless of the -bf control argument.

ACCESS REQUIRED: The user must have status (s) permission on the containing directory.

NOTES: If the list_iacl_dir command is invoked without any arguments, the entire initial ACL for the working directory is listed.

A directory initial ACL contains the ACL entries to be placed on directories created in the specified directory. For a discussion of initial ACLs, see "Access Control" in the MPM Reference Guide.

For a description of the matching strategy for User_ids, refer to set_acl in this manual.

EXAMPLES:

The command line:

```
! lid all_runoff .Faculty Fred..
```

lists, from the directory initial ACL of the all_runoff directory, all entries ending in Faculty.* and all entries with the Person_id Fred.

The command line:

```
! lid -wd -rg 5
```

lists entries in the ring 5 directory initial ACL of the working directory.

list_iacl_seg (lis)

SYNTAX AS A COMMAND:

lis {path} {User_ids} {-control_args}

SYNTAX AS AN ACTIVE FUNCTION:

[lis {path} {User_ids} {-control_args}]

FUNCTION: lists some or all of the entries on a segment initial access control list (segment initial ACL) in a specified directory.

ARGUMENTS:

path
specifies the directory in which the segment initial ACL should be listed. The star convention is allowed. If path is -wd, -working_dir, or omitted, the working directory is assumed. If path is omitted, no User_ids can be specified.

User_ids
are access control names of the form Person_id.Project_id.tag. All access names that match the given components are listed. If no User_id is specified, the entire segment initial ACL is listed.

CONTROL ARGUMENTS:

-ring N, -rg N
identifies the ring number whose segment initial ACL should be listed. If present, it must be followed by N (where $0 \leq N \leq 7$). This argument can appear anywhere on the line and affects the whole line. If this argument is not specified, the segment initial ACL of the user's current ring is listed.

-brief, -bf
suppresses the message "User name not on ACL of path." If lis is invoked as an active function, and the supplied User_id is not on the initial ACL, the null string is returned regardless of the -bf control argument.

ACCESS REQUIRED: The user must have status (s) permission on the containing directory.

NOTES: If list_iacl_seg (lis) is invoked without any arguments, the entire segment initial ACL for the working directory is listed.

A segment initial ACL contains the ACL entries to be placed on segments created in the specified directory. For a discussion of initial ACLs, see "Access Control" in the MPM Reference Guide.

For a description of the matching strategy for User_ids, refer to the set_acl command.

EXAMPLES:

The command line:

```
! lis all_runoff .Faculty. Fred
```

lists, from the segment initial ACL of the all_runoff directory, all entries with the Project_id Faculty and the entry for Fred.*.*.

The command line:

```
! lis -wd -rg 5
```

lists entries in the ring 5 segment initial ACL of the working directory.

list_not_accessible (lnac)

SYNTAX AS A COMMAND:

lnac {path} {User_id} {-control_args}

FUNCTION: scans a directory and lists segments and directories to which a given User_id does not have a given access condition.

ARGUMENTS:

path
is the pathname of the directory to be scanned. If path is omitted or -wd is specified, the working directory is scanned.

User_id
is an access name. It can have null components. The star convention for access names is allowed. See the description of set_acl in this manual. If User_id is omitted, the User_id of the user's process is assumed.

CONTROL ARGUMENTS:
If no control arguments are specified, all segments and directories to which the named user(s) has null access are listed.

-dir_mode STR
lists directories to which the named user(s) does not have any of the modes specified in STR, where STR can be any or all of the letters sma.

-seg_mode STR
lists segments to which the named user(s) does not have any of the modes specified in STR, where STR can be any or all of the letters rew.

ACCESS REQUIRED: The user must have status (s) permission on the directory.

list_not_accessible (lnac)

EXAMPLES:

```
! lnac >udd>work>Smith
  null  Smith.mbx

! lnac >udd>work>Smith -dir_mode m
  s     index
  null  Smith.mbx
  s     newindex
  s     gary1
  s     stuff
```

list_ref_names (lrn)

SYNTAX AS A COMMAND:

lrn paths {-control_args}

FUNCTION: lists the reference names associated with a specified segment; it accepts both segment numbers and pathnames as segment specifications.

ARGUMENTS:

paths
can be segment numbers or pathnames of segments known to the user's process. If path is a segment number, the pathname and reference names of the segment are printed. If path is a pathname, the segment number (in octal) and the reference names of the segment are printed. If a pathname looks like a control argument (i.e., if it is preceded by a minus sign) or a number, then path should be preceded by -name or -nm.

CONTROL ARGUMENTS:

-all, -a
prints the pathnames and reference names of all known segments, as well as the reference names of ring 0 segments. The -all control argument is equivalent to -from 0.

-brief, -bf
suppresses printing of the reference names for the entire execution of the command.

-from N, -fm N
allows the user to specify a range of segment numbers. This control argument can be used with the -to control argument. The pathnames and reference names of the segments in this range are printed. If -to is not specified, the highest used segment number is assumed.

-to
allows the user to specify a range of segment numbers. This control argument can be used with the -from control argument. The pathnames and reference names of the segments in this range are printed. If -from is not specified, the segment number of the first segment not in ring 0 is assumed, unless -all is used.

list_ref_names (lrn)

NOTES: All of the above arguments (segment specifiers and control arguments) can be mixed. For example, in the command line:

```
! lrn 156 -from 230 path_one
```

the pathname and reference names of segment 156 and of all segments from 230 on are printed. The segment number (in octal) and the reference names of path_one are printed.

In the default condition, when called with no arguments, list_ref_names prints information on all segments that are not in ring 0.

When a pathname is specified, the segment number by which it is known is printed. When a segment number is specified, lrn also prints the pathname of the segment.

list_resource_types (lrt)

SYNTAX AS A COMMAND:

lrt {type1 ... typen} {-control_args}

FUNCTION: prints a list of all resource types described in a resource type description table (RTDT).

ARGUMENTS:

typei
is the resource type defined in the RTDT for which information is to be listed. If no type is specified, all known resource types are listed.

CONTROL ARGUMENTS:

-no_header, -nhe
omits the column headers.

-pathname path, -pn path
lists resource types defined in the RTDT specified by path. If this control argument is not specified, the RTDT residing in >system_control_1 is used.

-long, -lg
lists the defined attributes for each resource type.

NOTES: For more information on RTDT, see MAM System, Order No. AK50.

list_resources (lr)

SYNTAX AS A COMMAND:

lr {-control_args}

FUNCTION: lists groups of resources managed by the Resource Control Package (RCP), selected according to criteria specified by the user.

CONTROL ARGUMENTS:

-acquisitions, -acq
lists resources acquired by the user specified by the -user control argument. If this control argument is used, -type must also be specified.

-assignments, -asm
lists resource assignments.

-awaiting_clear
lists those resources that are awaiting manual clearing.

-device STR, -dv STR
lists device resources with the name STR. No other resources are listed.

-logical_volume, -lv
lists logical volumes that are currently attached.

-long, -lg
prints all the information known about each resource listed. If this control argument is not supplied, only the name is printed for each resource listed.

-mounts, -mts
lists resources currently mounted by the process.

-reservations, -resv
lists only device and volume reservations.

-type STR, -tp STR
lists resources of the type STR. See list_resource_types for information on obtaining the names of resource types.

-user User_id
selects a particular user or group of users for whom resource information is to be printed. This control argument can be used only in conjunction with -acquisitions. The User_id can be any of the following forms:

list_resources (lr)

Person.Project
specifies a particular Person_id and Project_id combination.
*.Project
specifies all users on a specified project.
.
specifies all users (i.e., all acquired resources are listed).
free
specifies all resources in the free pool.
system
specifies all resources in the system pool.
**
specifies all users plus the free and system pools (i.e., all registered resources will be listed).

If this control argument is not specified, the User_id of the user invoking list_resources is assumed. See "Notes on Access Restrictions" below.

NOTES ON ACCESS RESTRICTIONS: Access to rcp_admin_ is required to obtain information on other users. Access to read the PDT of a project is required to obtain information for the project, using *.Project.

NOTES: If this command is invoked without any arguments, all resources assigned and devices attached to the calling process are listed.

EXAMPLES: In the example below, the user issues the list_resources command with no control arguments. The system responds with the name of the assigned devices.

```
! lr

   Device Assignments
        Device tape_05
        Device tape_02
```

list_resources (lr)

In the next example, the user issues the list_resources command with the -long control argument. The system responds with all the information known about each resource listed.

```
! lr -lg

   Device Assignments
   2 devices assigned
        Device tape_05
          State     =  assigned
          Time      =  04/30/76  1316.2 edt Fri
          Disp      =  retain
          Level     =  4
          Model     =  500
          Tracks    =  9
          Densities =  200 556 800 1600
          Speed     =  125
        Device tape_02
          State     =  assigned
          Time      =  04/30/76  1314.7 edt Fri
          Disp      =  retain
          Level     =  4
          Model     =  500
          Tracks    =  9
          Densities =  200 556 800 1600
          Speed     =  125
```

In the following example list_resources is invoked to obtain a list of tape volumes for which the user is the accounting owner.

```
! lr -tp tape_vol -acq
   4 resources of type tape_vol acquired by Dahl.GNP
   at 05/10/79 2025.5 mst Thu:

   a-153
   a-022
   u-405
   a-558
```

list_retrieval_requests (lrr)

SYNTAX AS A COMMAND:

lrr {path} {-control_args}

FUNCTION: lists retrieval requests in the retrieval daemon queues. The request identifier and entryname of each request are printed.

ARGUMENTS:

path
is the pathname of a request to be listed. The star convention is allowed. Only requests matching this pathname are selected. If the path argument is not specified, all pathnames are selected. Also see the -entry control argument below.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
prints the full pathname of each selected request, rather than just the entryname.

-admin {User_id}, -am {User_id}
selects the requests of all users, or of the user specified by User_id. If the -admin control argument is not specified, only the user's own requests are selected. See "Access Required" below.

-all, -a
searches all queues and prints the totals for each non-empty queue whether or not any requests are selected from it. This control argument is incompatible with the -queue control argument.

-brief, -bf
suppresses the printing of the state of the request. This control argument is incompatible with the -long and -total control arguments.

-entry STR, -et STR
selects only requests whose entrynames match STR. The star convention is allowed. Directory portions of request pathnames are ignored when selecting requests. This control argument is incompatible with the path argument.

-id ID
selects only requests whose identifiers match the specified ID.

-long, -lg
prints all the information pertaining to a retrieval request. If this control argument is omitted, only the full pathname of the object or subtree to be retrieved is printed.

-long_id, -lgid
prints the long form of the request identifier. If this or the -long control argument is not specified, the short form of the request identifier is printed.

-position, -psn
prints the position within its queue of each selected request. When used with the -total control argument, it prints a list of all the positions of the selected requests. See "Access Required" below.

-queue N, -q N
searches only queue N. If the -queue control argument is not specified, only queue 3 is searched. This control argument is incompatible with the -all control argument.

-total, -tt
prints only the total number of selected requests and the total number of requests in the queue plus a list of positions, if the -position control argument is specified. If the queue is empty, it is not listed. This control argument is not compatible with the -long and -brief control arguments.

-user User_id
selects only requests entered by the user specified by User_id. See "Access Required" below.

ACCESS REQUIRED: The user must have o access to the queue(s) to invoke lrr. The user must have r extended access to the queue(s), in order to use the -admin, -position, or -user control arguments, since it is necessary to read all requests in the queue(s) in order to select those entered by a specified user.

list_retrieval_requests (lrr)

NOTES: The -total, -brief, and -long control arguments are incompatible and cannot be used in the same list_retrieval_requests command line.

The default condition is to list only pathnames for the default queue.

The STR after the -admin or -user control arguments can have any of the following forms:

Person_id.Project_id	matches that user only
Person_id.*	matches that person on any project
Person_id	same as Person_id.*
*.Project_id	matches any user on that project
.Project_Id	same as *.Project_id
.	same as -admin with no User_id following

If no arguments are specified, only the user's own requests are selected for listing. Also see the enter_retrieval_requests command in this manual.

EXAMPLES: To get information about retrieval requests in the default queue, user Jones types the command line:

```
! lrr

   Queue 3:  3 requests. 6 total requests.

   211401 mydir
   211421 myseg
   211463 subtree
```

To get all information about retrieval requests in queue 1, the user types:

```
! lrr -lg -q 1

   Queue 1:  1 request. 27 total requests.

   Pathname:                    >udd>Demo>Jones>dump>mydir
   From time:                   01/16/77  2300.0 edt Thu
   Mode:                        notify previous
```

list_retrieval_requests (lrr)

To get the total number of retrieval requests in all queues, the user types:

```
! lrr -tt -a

   Queue 1:  2 requests. 15 total requests.

   Queue 3:  0 requests. 39 total requests.
```

Note that queue 2, being empty, does not have a total line listed.

SYNTAX AS A COMMAND:

logout {-control_args}

FUNCTION: terminates a user session and ends communication with the Multics system. It signals the finish condition for the process; and, after the default on unit for the finish condition returns, it closes all open files and destroys the process.

CONTROL ARGUMENTS:

-hold, -hd
the user's session is terminated. However, communication with the Multics system is not terminated, and a user can immediately log in without redialing.

-brief, -bf
no logout message is printed, and if the -hold control argument has been specified, no login message is printed either.

NOTES: See "How to Access the Multics System" in the New Programmer's Intro, Order No. AL40.

SYNTAX AS A COMMAND:

```
long_date {dt}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[long_date {dt}]
```

FUNCTION: returns a month name, a day number, and a year as a single string in the form "month day, year" (e.g., November 2, 1976).

ARGUMENTS:

dt
is any string acceptable to convert_date_to_binary_. The default is the current date.

NOTES: See the MPM Subroutines for a complete description of convert_date_to_binary_.

SYNTAX AS AN ACTIVE FUNCTION:

[low N]

FUNCTION: returns a specified number N of copies of the first (lowest) character in the ASCII character set, the NUL character or 000 octal.

NOTES: See the description of high in this manual.

SYNTAX AS A COMMAND:

lower_case strings

SYNTAX AS AN ACTIVE FUNCTION:

[lower_case strings]

FUNCTION: returns strings with all uppercase alphabetic characters translated to lowercase.

ARGUMENTS:

strings
is one or more character strings.

NOTES: Returned strings are separated from each other by a space. See the description of upper_case in this manual.

EXAMPLES:

The following interactions illustrate use of lower_case as an active function.

```
! ioa_ [lower_case "The time zone is MST."]
the time zone is mst.
```

The following interactions illustrate use of lower_case as a command.

```
! lower_case The time zone is MST.  My home is Phoenix.
the time zone is mst. my home is phoenix.
```

SYNTAX AS A COMMAND:

```
ltrim strA {strB}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[ltrim strA {strB}]
```

FUNCTION: returns a character string trimmed of specified characters on the left.

NOTES: The ltrim command, or active function, finds the first character of strA not in strB, trims the characters from strA preceding this character, and returns the trimmed result. Space characters are trimmed if strB is omitted.

EXAMPLES:

```
! string [ltrim 000305.000 0]
305.000
! string [ltrim "   This is it.   "]
This is it.
```

SYNTAX AS A COMMAND:

```
lv_attached lv_name
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[lv_attached lv_name]
```

FUNCTION: returns true if the volume specified (i.e., the lv_name argument) is attached to the user's process or if the volume is a public logical volume; otherwise it returns false.

ARGUMENTS:

lv_name
is the name of the logical volume.

SYNTAX AS A COMMAND:

mvp key args {-control_args}

SYNTAX AS AN ACTIVE FUNCTION:

[mvp key args {-control_args}]

FUNCTION: allows a user or a group of users to regulate the use of a predefined set of volumes (tape reels, etc.).

ARGUMENTS:

args
are one or more unique volume identifiers.

LIST OF KEYWORDS:

add, a
adds the specified volumes to the user's data base.

allocate, al
taking the arguments in pairs, marks volume argi as allocated and associates with it argument arg(i+1) as a comment about the volume. If argi is an asterisk (*), the oldest free volume is withdrawn and a message identifying the volume is printed, or returned in the active function case.

change, c
taking the arguments in pairs, changes the comment of volume argi to arg(i+1). Volumes must be in the allocated state.

comment, cm
prints the comment associated with each specified volume.

delete, d
deletes the specified volumes from the user's data base. Volumes must not be in the allocated state.

free, f
frees the specified volumes in the user's data base.

list, l
lists information about the specified volumes, or all volumes known to the user if no arguments are supplied. Three control arguments can also be supplied as part of the list request.

They are:

-free
list only those volumes that are free.

-match STR
list only those volumes whose comment contains STR as a substring.

-no_header, -nhe
suppress printing of the header.

print, p
prints the pathname of the current volume segment.

reserve, r
reserves the specified volumes.

test, t
tests whether the specified volumes are free. If argi is an asterisk (*), the oldest free volume is printed, or returned in the active function case.

In the active function case, test returns "true" if any of the specified volumes are free.

use {path}, u {path}
specifies the pathname of the manage_volume_pool segment to be used by future invocations of mvp in this process. The volumes suffix is assumed. If the pathname is omitted, the user's default volume segment is used.

NOTES: Normally, a tape reel or disk pack is a volume but any other set of objects, such as library books or portable terminals could just as easily be regulated. The data base describing the pool is named Person_id.volumes and exists in the user's home directory. This default can be reset via the 'use' key described above. Objects can be added to or deleted from the pool. Associated with each object in the pool is a state, a state change date, and a comment. An object can have one of three states: free, reserved, or allocated. The comment field can be any ASCII string up to 64 characters. The comment is intended to describe the contents of the volume but can as easily describe the attach description that created the volume.

master_directories (mdirs)

SYNTAX AS A COMMAND:

```
mdirs star_names {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[mdirs star_names {-control_arg}]
```

FUNCTION: returns the entrynames or absolute pathnames of master directories that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per directory is returned; i.e., if a master directory has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by master_directories is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

EXAMPLES: The following interaction illustrates the use of the master_directories active function.

```
! string [mdirs >udd>**]
Multics SysMaint
```

SYNTAX AS A COMMAND:

```
max num_args
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[max num_args]
```

FUNCTION: returns the maximum of the numeric arguments passed to it.

EXAMPLES:

```
! string [max 3.6 1e-3]
  3.6
```

SYNTAX AS A COMMAND:

memo {-control_arg} {optional_args} {memo_text}

FUNCTION: creates and manages an interactive notebook and reminder list.

CONTROL ARGUMENTS:
Only one control argument can appear on the command line, and it must be the first argument. If no control argument appears, the rest of the line is used to set a memo. If no arguments are specified, mature memos are printed or executed and alarms are enabled.

-pathname path, -pn path
uses the memo segment specified by the pathname path. The memo suffix is assumed. If the segment specified by path does not exist, memo attempts to create it. This control argument must not be followed by any optional arguments.

-list, -ls
lists memos selected by the optional arguments in full detail, including their maturity times, text, memo_numbers and information about the optional arguments (optional_args) used when the memos were set. No memos are executed.

-print, -pr
prints the text of all memos selected by the optional arguments. No memos are executed.

-delete, -dl
deletes all memos selected by the optional arguments.

-off
suppresses all memo alarms, until the next memo -on, memo, or memo -brief command. This control argument must not be followed by any optional arguments.

-on
enables memo alarms without printing or executing any nonalarm memos. This control argument must not be followed by any optional arguments.

-brief, -bf
suppresses the message "No memos" if none are found. Mature memos are printed or executed and alarms are enabled. This control argument must not be followed by any optional arguments.

LIST OF OPTIONAL ARGUMENTS:
can be selected from the following optional arguments. Some of the arguments can be used for setting memos; some for selecting memos to be printed, listed, or deleted; and others for both setting and selecting memos.

memo_number
specifies which memos are to be selected. The value of memo_number for each memo is printed when the user types memo, memo -brief, or memo -list.

-date DT, -dt DT
identifies a time (DT) in a form suitable for input to the subroutine convert_date_to_binary_ (see the description for this subroutine in the MPM Subroutines). The DT is truncated to midnight preceding the date in which DT falls. If used while setting a memo, then the truncated DT becomes the maturity time of the new memo. If memos are being selected, only those memos with maturity times prior to or equal to the truncated DT are selected.

-time DT, -tm DT
identifies a time (DT) in a form suitable for input to the subroutine convert_date_to_binary_. This optional argument is used in the same manner as the -date optional argument above except that DT is not truncated.

-alarm, -al
if a memo is being set, this specifies that the memo is to be an alarm. When mature, it is to be printed or executed immediately (or as soon as alarms are enabled) and then deleted. If memos are being selected, this argument selects any memos that are alarms.

-repeat DT, -rp DT
identifies the interval (where DT is a relative time >= 1 minute and acceptable to convert_date_to_binary) at which this memo is to appear. This optional argument is used when setting a memo. When the memo is mature, an identical memo is set with a maturity time that is DT amount of time in the future.

-invisible, -iv
specifies that the memo is never to be mature and will never be printed during a normal memo print. (Used only when setting a memo.)

-call
if a memo is being set, this argument specifies that the memo is to be passed to the command processor as a command. If

memos are being selected, this argument selects any memos that are such calls.

-match STR
selects memos containing substrings matching all of the strings (where each STR is a character string). The remainder of the command line is interpreted as the set of the strings to be matched. The maximum number of strings that can be specified is 32, and the maximum length of any one string is 32 characters.

MEMO TEXT ARGUMENT:

memo_text
is the text of the memo being set. The text portion can contain up to 132 characters.

NOTES: The memo command makes it possible to use Multics as an interactive notebook and reminder list containing memos. The user can specify a maturity time for each memo (a time before which the memo will not appear). By use of the alarm feature, the user can specify the exact time the memo is to be printed on the terminal. Memos can also be set that are passed directly to the command processor and executed as normal Multics command lines. Using these features jointly, the user can set a memo that actually performs a specified action at a specified time by itself rather than merely reminding the user to perform the action. Finally, the user can specify that the memo is to be repeated at regular intervals.

In the default case, memo maintains its information in the segment Person_id.memo in the user's home directory. If memo is invoked and such a segment does not exist, memo attempts to create and initialize it. Optionally, a different memo segment can be specified and used, for example in the user's start_up.ec. Each memo in the memo segment consists of a text portion containing up to 132 characters, a maturity date, a sequence number (memo_number) assigned by the memo command, and additional information telling whether the memo is to be repeated or not and whether it is to be printed or executed.

For the user's convenience, control arguments allow the printing, listing, and deletion of memos selected by subsequent optional arguments. Memos can be selected by number, type, maturity time, and content. Other control arguments enable or disable memo alarms.

If a date, time, repeat interval, or match string contains embedded blanks, that string must be enclosed in quotes so that the command processor passes it to memo as a single argument.

If the -pathname control argument is specified, the argument that follows must be the pathname of the memo segment that is to be used. If a memo segment is specified by this means, it continues to be used for the duration of the user's process, unless changed again by the -pathname control argument. If a segment with the specified pathname does not exist, memo attempts to create it.

To set a memo, no control arguments are given. Any of the optional arguments except -match and memo_number can be used to specify the type of memo being set and the time it is to mature. If no maturity time or date is specified, the maturity time is assumed to be the current time.

If memo is invoked with no arguments or with only the -brief control argument, all mature memos are printed or passed to the command processor. Alarms are enabled, and any alarms pending are printed or executed.

If either the -print or -list control argument is specified, all memos selected by the optional arguments are printed. The contents of the memo segment do not change in any way, and memos that would ordinarily be passed to the command processor are printed instead. If no optional arguments are used to select which memos are to be printed or listed, all memos are printed or listed. If the -date or -time optional arguments are specified, only those memos that mature before the specified date or time are printed. If the -call argument is specified, any such memo is printed. If the -alarm argument is specified, any alarm memos are printed.

If the -delete control argument is used, memos selected by the optional arguments are deleted. If no optional arguments have been used to specify which memos are to be deleted, none are deleted.

The -off control argument is useful for times when the user does not wish any extraneous output, such as when using the Multics runoff command. The command line memo -on can be

given to reenable alarms after they have been turned off, or it can be used at login or new_proc time to enable alarms without printing or executing other mature memos. Memo alarms are enabled by memo, memo -brief and memo -on commands, only.

EXAMPLES: In the following sequence of memo examples, input typed by the user is preceded by an exclamation mark (!). Ready messages from the system are omitted. First, the user's memo segment is initialized and is demonstrated to have no mature memos. Four memos are set and then listed, first in their entirety, then alarm memos, then only mature memos, then all memos maturing before a specified date. Finally, the only mature memo is deleted, and its successful deletion is demonstrated. The time of the example is 5/15/73 1729.

```
! memo
  memo:  Creating >udd>Demo>Jones>Jones.memo.


! memo
  No memos.


! memo get bookshelves
! memo -al -dt 5/23/73 -rp 2weeks Staff meeting at two.
! memo -call -al -dt 6/1/73 -rp 1month list -dtem -rv
! memo -tm "Thursday 9am" -rp 1week Weekly report due Friday.


! memo -ls
  1) Tue 05/15/73 1729 get bookshelves
  2) Wed 05/23/73 0000 Staff meeting at two. (alarm, "2weeks")
  3) Fri 06/01/73 0000 list -dtem -rv (call, alarm, "1month")
  4) Thu 05/17/73 0900 Weekly report due Friday. ("1week")


! memo -ls -al
  2) Wed 05/23/73 0000 Staff meeting at two. (alarm, "2weeks")
  3) Fri 06/01/73 0000 list -dtem -rv (call, alarm, "1month")


! memo
  1) get bookshelves
```

```
! memo -pr -dt 5/30/73
  get bookshelves
  Staff meeting at two.
  Weekly report due Friday.

! memo -dl -match book

! memo
  No memos.
```

FUNCTION: provides a generalized file merging capability, which is specialized for execution by user-supplied parameters.

NOTES: The basic function of the merge is to read one or more input files of records, which are ordered according to the values of one or more key fields (i.e., the files have been sorted using the sort command), merge (collate) those records according to the values of those key fields, and write a single file of ordered (or "ranked") records.

For a detailed description of both the sort and merge commands, refer to the Multics Sort/Merge Reference Manual, Order No. AW32.

SYNTAX AS A COMMAND:

ma paths {-control_args}

FUNCTION: merges two or more related ASCII text segments.

ARGUMENTS:

paths
are pathnames of segments to be merged as automatically as possible. Up to six segments can be merged, including those preceded by the -edit control_argument.

CONTROL ARGUMENTS:

-original path, -orig path
identifies path as the pathname of a segment containing the original version of the text. The proper original is the most recent common ancestor of the texts being merged. Overlapping changes, even if identical, cause edit mode to be entered.

-old_original path, -old_orig path
identifies path as the pathname of a segment antecedent to the most recent common ancestor of the texts being merged and allows the automatic picking up of identical changes present in all the texts being merged.

-output_file path, -of path
put the merged output text in the segment named path.

-edit path
merges the segment named path in a nonautomatic manner. Edit mode is entered each time a modification is found in the specified segment.

-minlines N
specifies the minimum number of lines that must be identical for merge_ascii to assume blocks of text in different segments are identical. The default value of of minlines is 2.

-minchars N
specifies the minimum number of characters that must be identical for merge_ascii to assume blocks of text in different segments are identical. The default value of minchars is 25.

NOTES: The merge_ascii program is typically used to merge texts that have been independently modified by several users. If an original version of the text is available, and if the user desires, merge_ascii performs the merge automatically, requiring user intervention only when overlapping modifications are detected. When user intervention is required, merge_ascii displays line-numbered blocks of text and then enters edit mode allowing the user to choose lines from any text or insert new lines.

When blocks of text are displayed, each line is preceded by a text identifier and a line number. The text identifier A is reserved for the original, whether supplied or not. The identifiers B-G are assigned to the texts being merged in the order in which their pathnames are encountered on the command line. The identifier M is used for the merged output, if printed while in edit mode.

The equal convention is allowed; equal processing is based on the first path argument in the command invocation.

Either the -original or -old_original (but not both) control argument may be used to enable automatic merging. If neither is supplied, edit mode is entered each time differences are found in the segments being merged. The -old_original control argument should be used judiciously, only if appropriate, and the user fully understands the relationships between the texts being merged.

EXAMPLES:

The command line:

```
! ma -orig pathA pathB pathC -of pathM
```

automatically merges the contents of pathB and pathC into pathM. Because an original version, pathA, is supplied, all nonoverlapping changes in pathB and pathC are placed in pathM. Only overlapping changes are displayed on the user's terminal and cause edit mode to be entered. (See "Notes on Edit Requests" below.)

The command line:

! ma pathB pathC -of pathM

performs a nonautomatic merge on the contents of pathB and pathC. All differences are displayed and cause edit mode to be entered. This type of merging is typically used when there is no "original" segment.

The command line:

! ma -orig pathA -edit pathB -edit pathC -of pathM

also performs a nonautomatic merge, but provides information to the user about the contents of the original text. In this case, although an original segment exists, the user wants complete control over what goes into the output segment.

The command line:

! ma -original pathA pathB -edit pathC -of pathM

performs a merge in which changes found in pathC cause edit mode to be entered. Nonoverlapping changes in pathB are picked up and automatically placed in the output segment. This combination of control arguments is useful when the user is familiar with the changes present in pathB but wishes to review changes present in pathC before picking them up.

The command line:

! ma -old_orig pathA pathB pathC -of pathM

merges pathB and pathC automatically under the assumption that pathA is an earlier version of the text than the most recent common ancestor of pathB and pathC. Changes present in both pathB and pathC are picked up automatically. The -old_original control argument can also be used to obtain an automatic merge if pathA is a true original but some changes have been applied to both pathB and pathC. If the -old_original control argument is used and pathA contains changes not present in both pathB and pathC, then the resulting output segment is nearly always useless to the user.

NOTES ON EDIT REQUESTS: In any invocation of edit mode the current block in each text is just the block of lines

previously displayed. The current block in text M is initially empty, and is grown as the user selects or inputs lines.

The print (p) and copy (k) requests may address any lines in any text (A to M) known to merge_ascii. The delete (d) request can only be applied to the current block in text M, and has the effect of undoing all edit requests made since changes were last displayed.

The edit requests are described below. In the syntax of the edit requests, <text_id> is the lowercase letter corresponding to the text identifier used by merge_ascii; <line_no> is a line number in the text segment. Line numbers can be specified as "<" to address the first line or as ">" to specify the last line of a current block.

<text_id>k
copy current block from specified text (e.g., bk copies current block from text B).

<text_id><line_no>k
copy specified line from specified text (e.g., b5k copies line 5 from text B).

<text_id><line_no>,<line_no>k
copy specified lines from specified text (e.g., b4,7k copies lines 4 through 7 from text B).

<text_id>p
print current block from specified text (e.g., bp prints current block from text B).

<text_id><line_no>p
print specified line from specified text (e.g., b6p prints line 6 from text B).

<text_id><line_no>,<line_no>p
print specified line from specified text (e.g., b12,16p prints lines 12 through 16 from text B).

<text_id>d
delete the current block in specified text (e.g., md deletes the current block in text M).

input
enter input mode.

.
return from input mode to edit mode.

go
exit editor and continue comparison.

quit
abort merge and return to command level. If this request is given during a merging procedure, all work is lost. Work is not saved unless merging is done from the beginning to the end of the segments.

e
execute rest of line as a Multics command line.

x
display identifiers, current line numbers, and pathnames of each text.

help
print a list of the edit requests and a brief explanation of each one.

NOTES: Multiple edit requests, delimited by blanks, can be given on a single request line. However, the quit, go, input, and e requests must not be followed by other requests.

SYNTAX AS A COMMAND:

```
min num_args
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[min num_args]
```

FUNCTION: returns the minimum of the numeric arguments passed to it.

EXAMPLES:

```
! string [min 3 -4]
-4
```

SYNTAX AS A COMMAND:

```
minus numA numB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[minus numA numB]
```

FUNCTION: returns the result of numA minus numB.

EXAMPLES:

```
! string [minus 3.5 3]
  0.5
```

SYNTAX AS A COMMAND:

```
minute {dt}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[minute {dt}]
```

FUNCTION: returns the one- or two-digit number of a minute of the hour, from 0 to 59.

ARGUMENTS:

dt
is a date-time in a form acceptable to convert_date_to_binary_. If no argument is specified, the current time is used.

NOTES: See the descriptions of time, hour, date_time, and date in this manual.

EXAMPLES:

```
! string [minute]
13
```

SYNTAX AS A COMMAND:

```
mod numA numB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[mod numA numB]
```

FUNCTION: returns the remainder of numA divided by numB (numA modulo numB).

EXAMPLES:

```
! string [mod 4. 3]
1
! string [mod 4.5 3.5]
1
```

SYNTAX AS A COMMAND:

```
month {dt}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[month {dt}]
```

FUNCTION: returns the one- or two-digit number of a month of the year, from 1 to 12.

ARGUMENTS:

dt
is a date-time in a form acceptable to convert_date_to_binary_. If no argument is specified, the current month is used.

EXAMPLES: The following example enters an absentee request for deferred execution to start at the beginning of the next month.

```
! ear abs_seg -time [date [month 1 month]/1]
```

The arguments to the month active function indicate that "1 month" should be added to the current date to get the date from which the month is to be calculated. The "/1" (when concatenated with the calculated month) forms a date string, e.g., "2/1".

SYNTAX AS A COMMAND:

```
month_name {dt}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[month_name {dt}]
```

FUNCTION: returns the full name of a month of the year.

ARGUMENTS:

dt
is a date-time in a form acceptable to covert_date_to_binary_. If no argument is specified, the current month is used.

EXAMPLES:

```
! string [month_name 08/09/79]
August
```

SYNTAX AS A COMMAND:

move path1_1 {path2_1...path1_n path2_n} {-control_arg}

FUNCTION: causes a designated segment or multisegment file (along with its access control list (ACL) and all names) to be moved to a new position in the storage system hierarchy.

ARGUMENTS:

path1_i
is the pathname of a segment or multisegment file to be moved. The star convention is allowed.

path2_i
is the pathname to which path1_i is to be moved. The equal convention is allowed. If the last path2 segment is not specified, path1_n is moved to the working directory and given the entryname path1_n.

CONTROL ARGUMENTS:

-acl
copies the ACL.

-all, -a
copies multiple names and ACLs.

-brief, -bf
suppresses the messages "Bit count inconsistent with current length..." and "Current length is not the same as records used...."

-chase
copies the targets of links that match path1. (See "Notes".)

-long
prints warning messages as necessary. This is the default.

-name, -nm
copies multiple names.

-no_acl
does not copy the ACL. This is the default.

-no_chase
does not copy the targets of links that match path1. (See "Notes".)

-no_name, -nnm
does not copy the multiple names. This is the default.

ACCESS REQUIRED: Read access is required for path1_i. Status and modify permission are required for the directory containing path1_i. Status, modify, and append permission are required for the directory containing path2_i.

NOTES: When an entry is moved, it is given all of the names that the path1_i argument already has plus the entryname specified in the path2_i argument.

The default for chasing links depends on path1. If path1 is a star name, links are not chased by default, if path1 is not a star name, links are chased.

Since two entries in a directory cannot have the same entryname, special action is taken by this command if the creation of a segment or multisegment file introduces a duplication of names within the directory. If an entry with the entryname path2_i already exists in the target directory and this entry has an alternate name, the conflicting name is removed and the user is informed of this action; the move then takes place. If the entry having the entryname path2_i has only one name, the entry must be deleted in order to remove the name. The user is asked if the deletion should be done; if the user answers "no", the move does not take place.

If path_i is protected by the safety switch, the user is asked whether path_i is to be deleted after it has been copied.

The initial ACL of the target directory has no effect on the ACL of the segment or multisegment file after it has been moved. The ACL remains exactly as it was in the original directory.

move (mv)

EXAMPLES: The command line:

```
! move alpha >Doe>= >Doe>beta b
```

moves alpha from the current working directory to the directory >Doe, keeping the name alpha, and moves beta from the directory >Doe to the current working directory with the names b and beta.

move_abs_request (mar)

SYNTAX AS A COMMAND:

mar request_ids {-control_args}

FUNCTION: moves a request from one absentee queue to another. The request is always placed at the end of the target queue.

ARGUMENTS: request_ids must be one or more of the following--

path
is the full or relative pathname of an absin or absentee input segment. The star convention is allowed.

-entry STR, -et STR
identifies the request to be moved by the entryname portion of the absin segment pathname. The star convention is allowed.

-id ID
identifies the request to be moved by the given ID. See the MPM Reference Guide for a description of request identifiers.

CONTROL ARGUMENTS:

-all, -a
searches all queues except the target queue. This control argument is not compatible with -foreground and -queue control arguments.

-brief, -bf
suppresses messages telling that a particular request_id was not found or which requests were moved when using star names or -all.

-foreground, -fg
specifies that the foreground queue contains the requests to be moved. This control argument is not compatible with -all or -queue control arguments.

-queue N, -q N
specifies that absentee queue N contains the requests to be moved. If not specified, all queues are searched. This control argument is not compatible with -all or -foreground control arguments.

-sender STR
specifies that only requests from sender STR should be moved. One or more request identifiers must be given.

move_abs_request (mar)

-to_queue N, -tq N
specifies which queue to move the request to. (Required)

-user User_id
is a character string specifying the name of the submitter of the request to be moved, if not equal to the group id of the process. User_id can be of the form Person_id.Project_id, Person_id, or .Project_id. This control argument is primarily for the operator and administrators. Both r and d extended access to the queue are required. This control argument causes the command to use privileged message segment primitives that preserve the original identity of the submitter. The AIM ring_1 privilege is needed to preserve the original AIM attributes. If ring_1 privilege is not present the AIM attributes of the user executing mar are used. The default is that only requests entered by the user executing mar are moved.

ACCESS REQUIRED: The user must have "o" access to move a user owned absentee request, and "adros" access to move another user's absentee request.

NOTES: When star names are not used and a single request_id matches more than one request in the queue(s) searched, none of the requests are moved. However, a message is printed telling how many matching requests there are.

A complete description of User_id and AIM attributes can be found in the MPM Reference Guide.

move_daemon_request (mdr)

SYNTAX AS A COMMAND:

mdr request_identifiers {-control_args}

FUNCTION: moves a request from one I/O daemon queue to another. The move can be within the same request type or from one request type to another. The request is always placed at the end of the target queue.

REQUEST IDENTIFIERS:

path
is the full or relative pathname of the request to be moved. The star convention is allowed to match the entrynames of segments.

-entry STR, -et STR
identifies the request to be moved by STR, the entryname portion of the input segment pathname. The star convention is allowed.

-id ID
identifies the request to be moved specified by its request identifier.

CONTROL ARGUMENTS:

-request_type STR, -rqt STR
specifies that the request moved is found in the queue(s) for the request type identified by STR. If this control argument is not specified, the default request type is "printer". Request types can be listed by the print_request_types command.

-queue N, -q N
specifies that queue N for the specified request type contains the request to be moved, where N is an integer specifying the number for the queue. If this control argument is omitted, only the default queue for the request type is searched. This control argument is incompatible with the -all control argument.

-all, -a
searches all queues for the requests to be moved. This control argument is incompatible with the -queue control argument. The target queue is not searched by the -all control argument.

move_daemon_request (mdr)

-to_request_type STR, -to_rqt STR
specifies that the request should be moved to request type STR. If this control argument is not specified, the original request type is used. The target request types must be of the same generic type as the original request type.

-to_queue N, -to_q N
is a required control argument specifying which queue to move the request to. The default queue is 3.

-brief, -bf
suppresses messages telling the user that a particular request identifier was not found or that requests were moved when using star names or the -all control argument.

-user User_id
specifies the name of the submitter of the requests to be moved. The default is to move only requests entered by the user executing the command. The User_id can be Person_id.Project_id, Person_id, or .Project_id. This control argument is primarily for the operator and administrators. Both r and d extended access to the queue are required. This control argument causes the command to use privileged message segment primitives that preserve the original identity of the submitter. If the process has access isolation mechanism (AIM) ring one privilege, the AIM attributes of the original submitter are preserved. Otherwise, the AIM attributes of the current process are used.

ACCESS REQUIRED: The user must have o extended access to the queue from which the request is being taken, and a access to the queue to which the request is being moved. The user must have r and d extended access to move a request owned by another user (see the description of the -user control argument above).

NOTES: When star names are not used and a single request identifier matches more than one request in the queue(s) searched, none of the requests are moved. However, a message is printed telling how many matching requests are found.

See the MPM Reference Guide for a description of request identifiers.

move_daemon_request (mdr)

EXAMPLES: To move from every queue, to queue 1, in the default request_type all requests where the last component of the pathname matches "list", the user types:

```
! mdr -et *.list -to_q 1 -all

   Daemon request mydir.list moved from queue 2 to queue 1.
   Daemon request myseg.list moved from queue 3 to queue 1.
```

SYNTAX AS A COMMAND:

mvd source_dir {target_dir} {-entry_type_keys} {-control_args}

FUNCTION: moves a directory and its subtree, including all of the associated attributes, to another point in the hierarchy.

ARGUMENTS:

source_dir
is the pathname of the directory to be moved.

target_dir
is the new pathname for source_dir. If the entryname is different from one already on source_dir, it is added to the existing names. If target_dir is not specified, source_dir is moved to the working directory and given the same entryname.

entry_type_keys
control what type of storage system entry is moved. If no entry_type_key is specified, all entries are moved. If any entry_type_key is specified, only those entry types specified are moved. The keys are:

-branch, -br
-directory, -dr
-file, -f
-link, -lk
-multisegment_file, -msf
-non_null_link, -nnlk
-segment, -sm

If one or more entry_type_keys are specified, but not the -directory key, the subtree of source_dir will not be followed.

CONTROL ARGUMENTS:

-brief, -bf
suppresses the printing of warning messages such as "Bit count inconsistent with current length" and "Current length is not the same as records used".

-force
continues execution when target_dir already exists, without asking the user. If the -force control argument is not specified, the user is queried.

-replace, -rp
deletes the contents of target_dir existing before the copying begins. If target_dir is non-existent or empty, this control argument has no effect. The default is to append the contents of the source directory to the target directory if it already exists.

ACCESS REQUIRED: Status and modify permission are required for source_dir and all of the directories in its tree, and its containing directory. If target_dir does not exist, append permission is required for its containing directory. If it does exist, modify and append permission for target_dir are required. This command does not force access.

NOTES ON ACCESS PROVISION: The access control list associated with source_dir is moved to target_dir.

NOTES ON EXISTENCE OF target_dir: If target_dir already exists, the user is so informed and asked if processing should continue unless the -force control argument is specified. If target_dir is contained in or contains source_dir, an appropriate error message is printed and control is returned to command level. Otherwise, the contents of source_dir either are appended to or replace the contents of target_dir. (See the -replace control argument.)

NOTES ON USE OF STAR AND EQUAL CONVENTIONS: The star and equal conventions can be used for source_dir and target_dir respectively. The star convention matches only directory names and moves them. Matching names associated with other storage types are ignored.

NOTES ON NAME DUPLICATIONS: Since two entries in a directory cannot have the same entryname, this command takes special action if the entryname of the entry being copied already exists in the directory specified by target_dir. If the entry is a directory, it is handled in the same fashion as duplication between source_dir and target_dir is handled, unless the existing entry in target_dir is not also a directory. In this case the entryname duplication is treated the same as non-directory entries. The procedure for non-directory entries is the standard system technique. If the -replace control argument is specified or target_dir does not exist, name duplication will not occur. See the copy command in this manual.

NOTES ON LINK TRANSLATION: Links are translated; that is, if there are references to a source directory in a link pathname, the link pathname is changed to refer to the target directory. See also the copy, move, and copy_dir commands in this manual.

If part of the tree is not moved, problems with link translation may occur. If the link in the source_dir tree was in the part of the tree not moved, there may be no corresponding entry in the target_dir tree. Hence, translation of the link (presumably originally non-null) will cause the link to become null.

EXAMPLES: If the working directory is >udd>Project>Smith, the command line:

```
mvd source_dir new>target_dir -rp
```

moves the directory named >udd>Project>Smith>source_dir and its subtree to >udd>Project>Smith>new>target_dir replacing its contents with the contents of source_dir.

move_quota (mq)

SYNTAX AS A COMMAND:

mq path1 quota_change1 ... {pathn quota_changen}

FUNCTION: allows a user to move records of quota between two directories, one immediately inferior to (contained in) the other.

ARGUMENTS:

pathi
is the pathname of a directory. The quota change takes place between this branch and its containing directory. A pathi of -wd or -working_directory specifies the working directory. The star convention is not allowed.

quota_changei
is the number of records to be moved between the immediately superior (containing) directory quota and the pathi quota. The quota_change argument can be either a positive or negative number. If it is positive, the quota is moved from the containing directory to pathi; if it is negative, the move is from pathi to the containing directory.

NOTES: The user must have modify permission on both the directory specified by pathi and its containing directory.

After the change, the quota on pathi must be greater than or equal to the number of records used in pathi unless the change makes the quota zero.

If the change makes the quota on pathi zero, there must be no immediately inferior directory with nonzero quota, and the records used and the record-time product for pathi are reflected up to the superior directory.

If pathi is an upgraded directory (its access class is greater than the access class of its containing directory), quota_changei must be positive. Quota can only be moved back to the containing directory of an upgraded directory by deleting the upgraded directory.

Quota cannot be moved between a master directory and its containing directory. It can, however, be moved between a master directory and an inferior directory as described above. See the set_mdir_quota command in MAM Project, Order No. AK51, for information on changing the quota on a master directory.

EXAMPLES:

The command line:

```
! mq >udd>m>Smith>sub1_dir 1000
```

adds 1000 records to the quota on >udd>m>Smith>sub1_dir and subtracts 1000 records from the quota on >udd>m>Smith.

The command line:

```
! mq >udd>m>Smith>sub1_dir>sub2_dir -50
```

subtracts 50 records from the quota on >udd>m>Smith>sub1_dir>sub2_dir and adds 50 records to the quota on >udd>m>Smith>sub1_dir.

SYNTAX AS A COMMAND:

```
msfs star_names {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[msfs star_names {-control_arg}]
```

FUNCTION: returns the entrynames or absolute pathnames of multisegment files that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per multisegment file is returned; i.e., if a multisegment file has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by msfs is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

EXAMPLES: The following interaction illustrates the use of the msfs active function.

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w    0  empty_seg
  re     1  test
  r w    1  test.list
  r w    1  test.pl1
  re     1  prog
  r w    1  prog.list
  r w    1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w  513  prog.output
  r w  257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                    >udd>Apple>Jones>temp_seg_2
  prog.temp1                    >udd>Apple>Jones>temp_seg_1
  junk                          >udd>Apple>Jones>empty_seg

! string [msfs **]
  prog.data prog.output
! string [msfs *.output]
  prog.output
```

SYNTAX AS A COMMAND:

```
nequal numA numB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[nequal numA numB]
```

FUNCTION: returns true if numA is numerically equal to numB; otherwise it returns false.

NOTES: See the description for other commands and active functions pertaining to logical operations and comparing numeric data in this manual.

EXAMPLES:

```
! string [nequal 5 5.0]
  true
! string [nequal 001 1]
  true
! string [nequal one 1]
  Error
```

SYNTAX AS A COMMAND:

new_fortran path {-control_args}

FUNCTION: invokes the FORTRAN compiler to translate a segment containing the text of a FORTRAN source program into a Multics object segment.

ARGUMENTS:

path
is the pathname of a FORTRAN source segment. The fortran suffix is assumed if not present. However, the fortran suffix must be the last component of the name of the source segment.

CONTROL ARGUMENTS:

-brief, -bf
shortens all error messages written to the user_output I/O switch and combines error messages as described under "Notes on Error Diagnostics" below. The same error messages appear in the listing segment in the long form.

-brief_table, -bftb
generates a partial symbol table consisting only of a statement table that gives the correspondence between source line numbers and object locations. The table appears in the symbol section of the object segment produced for the compilation. This control argument does not significantly increase the size of the object program.

-card
specifies that the source segment is in card-image format and that all uppercase letters not occurring within character-string constants are mapped to their lowercase form. See Multics FORTRAN, Order No. AT58 for a description of card-image format.

-check, -ck
is used for syntactic checking of a FORTRAN program. The code generation phase of the compiler is not executed and no object segment is created. Any requested listing segment will be created, but it contains only the source and the error messages from the compiler. No symbol table and no assembly listing are produced.

-fold
maps all uppercase letters not occurring within character-string constants to their lowercase form. This control argument is useful for segments that are in free format but in uppercase.

-line_numbers, -ln
ignores line numbers on each input source line. If this control argument is not specified, the line numbers for each source line are printed. Note that the line numbers are printed only if they are input with the source. See Multics FORTRAN, Order No. AT58 for information on line numbers.

-list, -ls
produces a complete source program listing including an assembly-like listing of the compiled program. Use of the -list control argument significantly increases compilation time and should be avoided whenever possible by using the -map control argument. The -map control argument produces sufficient information to allow the user to debug most problems online. See "Notes on Listing" below.

-map
produces a partial source program listing of the compiled program. See "Notes on Listing" below.

-non_relocatable, -nrlc
inhibits the generation of relocation information. If this control argument is specified, the resulting object segment cannot be bound.

-optimize, -ot
performs the following global optimizations: removal of common subexpressions, removal of invariant expressions from loops, strength reduction, test replacement, constant propagation, and removal of assignments made dead by other optimizations. Various local optimizations are also performed. For further explanation of the above global optimizations, see the Multics FORTRAN Users' Guide, Order No. CC70. These optimizations tend to reduce the execution time and size of the object program. Use of this control argument can significantly increase compilation time.

-profile, -pf
generates additional code to meter the execution of individual statements. Each statement in the object program contains an additional instruction to increment an internal counter associated with that statement. After a program has been executed, the profile command can be used to print the execution counts.

-relocatable, -rlc
generates relocation information and includes it in the object segment. This is the default.

-safe_optimize, -safe_ot
inhibits some code movement out of loops by the optimizer. All other optimizations implied by the -optimize control argument are performed. Removal of invariant operations from portions of a loop that are not always executed when the loop is entered is inhibited if these operations could possibly cause the fixedoverflow, underflow, or overflow conditions to be signalled. Assignments and operations that could cause the zerodivide or error conditions to be signalled are never removed from the abovementioned portions of a loop whether or not the -safe_optimize control argument is specified. For most programs, the -optimize control argument always gives correct results. The -safe_optimize control argument is necessary only if the -optimize control argument causes a valid program to signal the fixedoverflow, underflow, or overflow conditions that were not signalled when unoptimized. For further information, see the Multics FORTRAN Users' Guide, Order No. CC70.

-severityN, -svN
does not print error messages whose severity is less than N (where N is 1, 2, 3, or 4) although all errors are written to the listing. If this control argument is not specified, a severity level of 1 is assumed. For a description of severity levels, see "Notes on Error Diagnostics" below.

-subscriptrange, -subrg
produces extra code for all subscripted array references to check for subscript values exceeding the declared dimension bounds. Such an error causes the subscriptrange condition to be signalled. This control argument must not be used with -optimize.

-table, -tb
generates a full symbol table for use by symbolic debuggers. The symbol table is part of the symbol section of the object program and consists of two parts: a statement table that gives the correspondence between source line numbers and object locations, and a name table that contains information about names actually referenced by the source program. This control argument usually causes the object segment to become significantly longer. It may not produce the desired results if -optimize is also specified.

-time, -tm
prints a table after compilation giving the time (in seconds),

the number of page faults, and the size of the temporary area for each phase of the compiler.

-time_ot
prints a table after optimization giving the time in seconds and the number of page faults for each phase of the optimizer.

NOTES: The only result of invoking the new_fortran command without control arguments is to generate an object segment.

A successful compilation produces an object segment and leaves it in the user's working directory. If an entry with that name already exists in the directory, its access control list (ACL) is saved and given to the new copy of the object segment. Otherwise, the user is given re access to the segment with ring brackets v,v,v where v is the validation level of the process.

If the user specifies the -map or -list control arguments, the new_fortran command creates a listing segment in the working directory and gives it a name consisting of the entryname portion of the source segment with a suffix of list rather than fortran (e.g., a source segment named test.fortran has a listing segment named test.list). The ACL is as described for the object segment except that the user is given rw access to it when newly created. Previous copies of the object segment and the listing segment are replaced by the new segments created by the compilation.

For information on Multics FORTRAN, refer to the Multics FORTRAN, Order No. AT58 and to the Multics FORTRAN Users' Guide, Order No. CC70. For information on using the FAST subsystem to compile FORTRAN source code, refer to Multics FAST Subsystem Users' Guide, Order No. AU25.

NOTES ON ERROR DIAGNOSTICS: The new FORTRAN compiler can diagnose and issue messages for about 200 different errors. These messages are graded in severity as follows:

1 Warning only. Compilation continues without ill effect.

2 Correctable error. The compiler remedies the situation and continues, probably without ill effect. For example, a missing end statement can be and is corrected by simulating the appending of an end statement to the source to complete

the program. This does not guarantee correct results, however.

3 An uncorrectable but recoverable error. That is, the program is definitely in error and cannot be corrected but the compiler can and does continue executing up to the point just before code is generated. Thus, any further errors are diagnosed. If the error is detected during code generation, code generation is completed although the code generated is not correct. After the compilation, a message is printed to the error_output I/O switch to inform the user that an error of severity 3 has occurred.

4 An unrecoverable error. The compiler cannot continue beyond this error. The message is printed and then control is returned to the new_fortran command. The command writes an abort message to the error_output I/O switch and returns to its caller.

Error messages are written to the user_output I/O switch as they occur. Thus, a user can quit the compilation immediately when an error occurs. As indicated above, the user can set the severity level so as not to be bothered by minor error messages. The user can also specify the -brief control argument so that messages are shorter. An example of an error message in its long form is:

```
ERROR 42, severity 3 on line 107
A do statement cannot be the second part of a logical if
statement.
```

If the -brief control argument is specified, the user sees instead:

```
ERROR 42, severity 3 on line 107
do
```

The -severity 4 control argument suppresses this message entirely.

Once a given error message is printed on the user's terminal in the long form, all further instances of that error message are printed in the short form.

If a listing is being produced, all error messages, regardless of severity, appear in the listing segment in their long form.

NOTES ON SEVERITY: The new_fortran command associates the following severity values to be used by the severity active function when the "fortran" keyword is used:

Value	Meaning
0	No compilation yet or no error.
1	Warning.
2	Correctable error.
3	Fatal error.
4	Unrecoverable error.
5	Bad control arguments or could not find source.

NOTES ON LISTING: The listing created by the new_fortran command begins with header lines specifying the complete pathname of the source segment, the version of the FORTRAN compiler used, the date and time the compilation occurred, and the control arguments specified by the user.

After the header lines, the following information is provided for each program unit in the compilation:

- a line-numbered, printable ASCII listing of the program unit. The compiler provides line numbers if the user does not.
- an alphabetized table of all names, except statement labels, used in the program unit. Each name appears with its attributes, such as mode, storage class, location, and type of name, and a list of lines on which it is used. If the code generator is not invoked, all names, except statement labels, appear in this table.
- an alphabetized table of all names, except statement labels, declared in the program unit but not used. If the name is not a member of a common block, that name is not allocated storage. Each name appears with its attributes and a list of lines on which it is declared.
- a table, in ascending numeric order, of all statement labels in the program unit. Each label appears with the type of statement with which it is associated, its location

if it is associated with an executable statement, the line on which it is declared, and a list of lines on which it is used.

- a table associating each executable source line with an object location. The table is arranged by ascending source line number. This table is only available if the code generator is invoked.

- a list of error messages for the program unit. All error messages appear in their long form.

- an assembly-like listing of the object segment. Each executable statement appears followed by the executable instructions generated for that statement. Each instruction appears on a line with the octal representation of the instruction word, and an assembly-like representation of the word including operation code, pointer-register, and modifier mnemonics. All offsets in the assembly representation are decimal numbers. If the address field of the instruction uses the IC (self-relative) modifier, the absolute text location corresponding to the relative address is printed in the remarks field of the line. If the reference is to a constant or name declared by the user, the name is printed in the remarks field of the line.

  The assembly-like listing is only produced if the code generator is invoked and the user specifies the -list control argument. Producing an assembly list significantly increases compilation time and the size of the listing segment.

After the above information is put in the listing segment, the following general information concerning the object segment is appended if the code generator is invoked:

- an assembly-like listing of all constants allocated in the object segment. This is provided only if the user specifies the -list control argument.

- a table showing the storage requirements for the object segment. This table gives the size and offset for each section of the object segment. The size of the stack frame of the object segment is also given.

- an alphabetized list of all entry point names defined in this compilation. Each entry point name appears with the

object segment offset for its external entry point, the program unit in which it appears if it is not a major entry point, the line on which it appears, and all external references in other program units that are resolved by it.

- an alphabetized list of all external references made that are not resolved within this compilation, including the lines on which they are referenced.

- an alphabetized list of all common blocks, the lines on which each is declared, and the declared length for each declaration.

- a list of all source segments used in the compilation. This includes the source segment specified on the command line as well as any include files used.

SYNTAX AS A COMMAND:

```
new_proc {-control_arg}
```

FUNCTION: destroys the user's current process and creates a new one, using the control arguments given initially with the login command, and the optional argument to the new_proc command itself. Just before the old process is destroyed, the "finish" condition is signalled. After the default on unit returns, all open files are closed. The search rules, I/O attachments, and working directory for the new process are as if the user had just logged in.

CONTROL ARGUMENTS:

-authorization STR, -auth STR
to create the new process at authorization STR, where STR is any authorization acceptable to the convert_authorization_ subroutine. (See the convert_authorization_ subroutine in the MPM Subroutines.) The authorization must be less than or equal to both the maximum authorization of the process and the access class of the terminal. The default is to create the new process at the same authorization.

NOTES: If the user's initial working directory contains a segment named start_up.ec, and the user did not log in with the -no_start_up control argument, new_proc causes the command line:

```
! exec_com start_up new_proc interactive
```

to be automatically issued in the new process. This feature can be used to initialize per-process static variables.

SYNTAX AS A COMMAND:

```
ngreater numA numB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[ngreater numA numB]
```

FUNCTION: returns true if numA is numerically greater than numB; otherwise it returns false.

EXAMPLES:

```
! string [ngreater 5 8]
  false
! string [ngreater 9 4]
  true
```

SYNTAX AS A COMMAND:

```
nless numA numB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[nless numA numB]
```

FUNCTION: returns true if numA is numerically less than numB; otherwise it returns false.

EXAMPLES:

```
! string [nless 8 4]
  false
! string [nless 4 8]
  true
! string [nless -5 -3]
  true
```

SYNTAX AS A COMMAND:

nmdirs star_names {-control_arg}

SYNTAX AS AN ACTIVE FUNCTION:

[nmdirs star_names {-control_arg}]

FUNCTION: returns the entrynames or absolute pathnames of directories that are not master directories that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per directory is returned; i.e., if a directory has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by nonmaster_directories is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

nonmaster_directories (nmdirs)

EXAMPLES: The following interaction illustrates the use of the nonmaster_directories active function.

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w    0  empty_seg
  re     1  test
  r w    1  test.list
  r w    1  test.pl1
  re     1  prog
  r w    1  prog.list
  r w    1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w  513  prog.output
  r w  257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                  >udd>Apple>Jones>temp_seg_2
  prog.temp1                  >udd>Apple>Jones>temp_seg_1
  junk                        >udd>Apple>Jones>empty_seg

! string [nmdirs *]
  documents prog_stuff
```

no_save_on_disconnect

SYNTAX AS A COMMAND:

no_save_on_disconnect

FUNCTION: disables process preservation across hangups in the user's process, causing the process to log itself out automatically if its terminal channel hangs up.

NOTES: This command is only meaningful if process preservation was in effect for the process at login time, either by default or because the -save_on_disconnect control argument was specified on the login command line.

nondirectories (nondirs)

SYNTAX AS A COMMAND:

nondirs star_names {-control_arg}

SYNTAX AS AN ACTIVE FUNCTION:

[nondirs star_names {-control_arg}]

FUNCTION: returns the entrynames or absolute pathnames of segments, multisegment files, and links that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per entry is returned; i.e., if a segment, multisegment file, or link has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by nondirectories is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

nondirectories (nondirs)

EXAMPLES: The following interaction illustrates the use of the nondirectories active function.

```
! pwd
>udd>Apple>Jones
! ls -a

   Segments = 7, Lengths = 6.

   r w     0  empty_seg
   re      1  test
   r w     1  test.list
   r w     1  test.pl1
   re      1  prog
   r w     1  prog.list
   r w     1  prog.pl1

   Multisegment-files = 2, Lengths = 770.

   r w   513  prog.output
   r w   257  prog.data

   Directories = 2.

   sma  prog_stuff
   sma  documents

   Links = 3.

   prog.temp2                  >udd>Apple>Jones>temp_seg_2
   prog.temp1                  >udd>Apple>Jones>temp_seg_1
   junk                        >udd>Apple>Jones>empty_seg

! string [nondirs prog.*]
   prog.pl1 prog.list prog.data prog.output prog.temp1
     prog.temp2
! string [nondirs *]
   prog test empty_seg junk
```

SYNTAX AS A COMMAND:

nonfiles star_names {-control_arg}

SYNTAX AS AN ACTIVE FUNCTION:

[nonfiles star_names {-control_arg}]

FUNCTION: returns the entrynames or absolute pathnames of directories and links that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per entry is returned; i.e., if a directory or link has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by nonfiles is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

EXAMPLES: The following interaction illustrates the use of the nonfiles active function.

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w    0  empty_seg
  re     1  test
  r w    1  test.list
  r w    1  test.pl1
  re     1  prog
  r w    1  prog.list
  r w    1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w  513  prog.output
  r w  257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                >udd>Apple>Jones>temp_seg_2
  prog.temp1                >udd>Apple>Jones>temp_seg_1
  junk                      >udd>Apple>Jones>empty_seg

! string [nonfiles *]
  documents prog_stuff junk
```

SYNTAX AS A COMMAND:

nonmsfs star_names {-control_arg}

SYNTAX AS AN ACTIVE FUNCTION:

[nonmsfs star_names {-control_arg}]

FUNCTION: returns the entrynames or absolute pathnames of segments, directories, and links that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per entry is returned; i.e., if a segment, directory, or link has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by nonmsfs is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

EXAMPLES: The following interaction illustrates the use of the nonmsfs active function.

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w    0  empty_seg
  re     1  test
  r w    1  test.list
  r w    1  test.pl1
  re     1  prog
  r w    1  prog.list
  r w    1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w  513  prog.output
  r w  257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                  >udd>Apple>Jones>temp_seg_2
  prog.temp1                  >udd>Apple>Jones>temp_seg_1
  junk                        >udd>Apple>Jones>empty_seg

! string [nonmsfs prog.*]
  prog.pl1 prog.list prog.temp1 prog.temp2
! string [nonmsfs *]
  prog test empty_seg documents prog_stuff junk
```

nonnull_links (nnlinks)

SYNTAX AS A COMMAND:

nnlinks star_names {-control_arg}

SYNTAX AS AN ACTIVE FUNCTION:

[nnlinks star_names {-control_arg}]

FUNCTION: returns the entrynames or absolute pathnames of links for which the target entry exists that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per link is returned; i.e., if a link has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by nonnull_links is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

nonnull_links (nnlinks)

EXAMPLES: The following interaction illustrates the use of the nonnull_links active function.

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w    0  empty_seg
  re     1  test
  r w    1  test.list
  r w    1  test.pl1
  re     1  prog
  r w    1  prog.list
  r w    1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w  513  prog.output
  r w  257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                  >udd>Apple>Jones>temp_seg_2
  prog.temp1                  >udd>Apple>Jones>temp_seg_1
  junk                        >udd>Apple>Jones>empty_seg

! string [nnlinks **]
  junk
```

SYNTAX AS A COMMAND:

nonsegs star_names {-control_arg}

SYNTAX AS AN ACTIVE FUNCTION:

[nonsegs star_names {-control_arg}]

FUNCTION: returns the entrynames or absolute pathnames of directories, multisegment files, or links that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per entry is returned; i.e., if a directory, multisegment file, or link has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by nonsegments is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

EXAMPLES: The following interaction illustrates the use of the nonsegments active function.

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w    0  empty_seg
  re     1  test
  r w    1  test.list
  r w    1  test.pl1
  re     1  prog
  r w    1  prog.list
  r w    1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w  513  prog.output
  r w  257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                  >udd>Apple>Jones>temp_seg_2
  prog.temp1                  >udd>Apple>Jones>temp_seg_1
  junk                        >udd>Apple>Jones>empty_seg

! string [nonsegs prog*.**]
  prog.data prog.output prog_stuff prog.temp1 prog.temp2
```

nonzero_files (nzfiles)

SYNTAX AS A COMMAND:

```
nzfiles star_names {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[nzfiles star_names {-control_arg}]
```

FUNCTION: returns the entrynames or absolute pathnames of files (segments and multisegment files) with a nonzero bit count that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENT:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per file is returned; i.e., if a file has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by nonzero_files is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

nonzero_files (nzfiles)

EXAMPLES: The following interaction illustrates the use of the nonzero_files active function.

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w     0  empty_seg
  re      1  test
  r w     1  test.list
  r w     1  test.pl1
  re      1  prog
  r w     1  prog.list
  r w     1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w   513  prog.output
  r w   257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                  >udd>Apple>Jones>temp_seg_2
  prog.temp1                  >udd>Apple>Jones>temp_seg_1
  junk                        >udd>Apple>Jones>empty_seg

! string [nzfiles *]
  prog test
! string [nzfiles prog.*]
  prog.pl1 prog.list prog.data prog.output
```

nonzero_msfs (nzmsfs)

SYNTAX AS A COMMAND:

nzmsfs star_names {-control_arg}

SYNTAX AS AN ACTIVE FUNCTION:

[nzmsfs star_names {-control_arg}]

FUNCTION: returns the entrynames or absolute pathnames of multisegment files with a nonzero bit count that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per msf is returned; i.e., if a multisegment file has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by nonzero_msfs is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

nonzero_msfs (nzmsfs)

EXAMPLES: The following interaction illustrates the use of the nonzero_msfs active function.

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w    0  empty_seg
  re     1  test
  r w    1  test.list
  r w    1  test.pl1
  re     1  prog
  r w    1  prog.list
  r w    1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w  513  prog.output
  r w  257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                  >udd>Apple>Jones>temp_seg_2
  prog.temp1                  >udd>Apple>Jones>temp_seg_1
  junk                        >udd>Apple>Jones>empty_seg

! string [nzmsfs **]
  prog.data prog.output
```

nonzero_segments (nzsegs)

SYNTAX AS A COMMAND:

nzsegs star_names {-control_arg}

SYNTAX AS AN ACTIVE FUNCTION:

[nzsegs star_names {-control_arg}]

FUNCTION: returns the entrynames or absolute pathnames of segments with a nonzero bit count that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per segment is returned; i.e., if a segment has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by nonzero_segments is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

nonzero_segments (nzsegs)

EXAMPLES: The following interaction illustrates the use of the nonzero_segments active function.

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w    0  empty_seg
  re     1  test
  r w    1  test.list
  r w    1  test.pl1
  re     1  prog
  r w    1  prog.list
  r w    1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w  513  prog.output
  r w  257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                >udd>Apple>Jones>temp_seg_2
  prog.temp1                >udd>Apple>Jones>temp_seg_1
  junk                      >udd>Apple>Jones>empty_seg

! string [nzsegs *]
  prog test
```

SYNTAX AS A COMMAND:

not str

SYNTAX AS AN ACTIVE FUNCTION:

[not str]

FUNCTION: returns false if str is equal to true; true if str is equal to false; otherwise prints an error message.

SYNTAX AS A COMMAND:

```
null_links star_names {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[null_links star_names {-control_arg}]
```

FUNCTION: returns the entrynames or absolute pathnames of links for which the target does not exist that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per link is returned; i.e., if a link has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by null_links is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

EXAMPLES: The following interaction illustrates the use of the null_links active function.

```
! pwd
>udd>Apple>Jones
! ls -a

   Segments = 7, Lengths = 6.

   r w    0  empty_seg
   re     1  test
   r w    1  test.list
   r w    1  test.pl1
   re     1  prog
   r w    1  prog.list
   r w    1  prog.pl1

   Multisegment-files = 2, Lengths = 770.

   r w  513  prog.output
   r w  257  prog.data

   Directories = 2.

   sma  prog_stuff
   sma  documents

   Links = 3.

   prog.temp2                >udd>Apple>Jones>temp_seg_2
   prog.temp1                >udd>Apple>Jones>temp_seg_1
   junk                      >udd>Apple>Jones>empty_seg

! string [null_links prog.temp*]
prog.temp1 prog.temp2
```

SYNTAX AS A COMMAND:

```
oct values
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[oct values]
```

FUNCTION: returns one or more values in octal.

ARGUMENTS:

value
is a value to be processed. The last character of the value indicates its type. Acceptable types are binary (b), quartenary (q), octal (o), hexadecimal (x), or unspecified (u). Any valid PL/I real value is allowed. The absence of any specifier means decimal. The unspecified value is limited to 8 characters.

EXAMPLES:

```
! string [octal 1024]
2000
```

SYNTAX AS A COMMAND:

```
on conditions handler_com_line {-control_args}
     subject_com_line
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[on conditions handler_com_line {-control_args}
     subject_com_line]
```

FUNCTION: establishes a handler for a specified set of conditions, executes an imbedded command line with this handler in effect, and then reverts the handler. The handler is another imbedded command line to be executed if the condition is signalled.

The active function returns true if any of the specified conditions are signalled during the execution of subject_com_line, false otherwise.

ARGUMENTS:

conditions
is a list of condition names separated by commas to be trapped by the on command.

handler_com_line
is the command line to be executed when one of the conditions contained in the list of condition names is raised. If handler_com_line contains spaces or other command language characters, it must be enclosed in quotes. If no command is to be executed when a condition is raised, handler_com_line must be given as "".

subject_com_line
is the command line to be executed under the control of on. The subject_com_line consists of the remaining arguments.

CONTROL ARGUMENTS:
must appear before subject_com_line.

-brief, -bf
suppresses the comment printed when a condition occurs.

-long, -lg
prints a detailed message describing the condition raised, if one is available.

-restart, -rt
continues execution of the subject_com_line after execution of handler_com_line, or if -cl is also specified, after the start command is executed.

-cl
establishes a new command level after the execution of handler_com_line. The state of subject_com_line is preserved. This control argument is not allowed for the on active function.

-exclude STR, -ex STR
prevents on from trapping the conditions given in STR. If more than one condition is listed, condition names are separated by commas. This control argument is useful when handling the any_other condition.

NOTES: The handler_com_line is a single argument, and must be quoted if it contains spaces or other command language characters, The subject_com_line consists of the remaining arguments on the line and should be quoted if it contains parenthesis, brackets, or semicolon. The handler is only in effect while subject_com_line is being executed. After executing handler_com_line, the on command returns nonlocally, thus aborting the execution of subject_com_line, unless -restart has been specified. The standard default error handler is in effect during the execution of handler_com_line.

The message produced by the -long control argument is the same as the message printed by the reprint_error command. The -brief and -long control arguments are mutually exclusive.

See the MPM Reference Guide for a list of standard system conditions.

EXAMPLES:

The command line:

```
! on command_error "pwd; ls" -bf ws node la
```

does a walk_subtree starting at the node directory, listing the access of the working directory. When the list_acl (la) command fails, for example, because of insufficient access, the pathname and contents of the working directory are printed and the user returns to command level since -restart is not specified.

The command line:

```
! on any_other -ex quit,program_interrupt,mme2 "ec dump" -lg
myprog
```

executes the myprog command. If any condition except the quit, program_interrupt, and mme2 condition is raised, on executes the "ec dump" command, after printing a detailed explanation of the condition raised.

The command line:

```
! on quit,mme2 db -bf -rt testcom
```

executes the testcom command but responds to quits and breaks set in testcom by invoking debug. The control arguments -rt, causing execution of testcom to continue after the user quits out of debug, and -bf, suppressing a warning message when one of the specified conditions is signalled, apply to the on command.

In an exec_com, the command line:

```
! on linkage_error "ec linkerr" ec recurse
```

calls a recursive entry point in the exec_com to continue execution, but with a linkage_error handler in effect. When linkage_error is signalled during the course of running recurse.ec, that exec_com is aborted and linkerr.ec is run.

The exec_com &if control line:

```
&if [on command_error "" -bf -rt command_name]
&then &quit
```

executes the command command_name. If the command_error condition is raised, the exec_com being executed is terminated after completing the execution of the command. The on command does not print any message in this example; restarting the command_error condition will print a message.

SYNTAX AS A COMMAND:

or tf_args

SYNTAX AS AN ACTIVE FUNCTION:

[or tf_args]

FUNCTION: returns true if any tf_arg is equal to true; otherwise it returns false. If any tf_arg does not have the value true or false, an error message is printed.

SYNTAX AS A COMMAND:

ov paths {-control_args}

FUNCTION: reads several ASCII segments and writes on the user_output I/O switch output that is the result of superimposing print positions from each segment.

ARGUMENTS:

paths
are the pathnames of input segments.

CONTROL ARGUMENTS:

-page_length N, -pl N
sets the page length of the output. If this control argument is not given, a page length of 60 is used.

-indent N, -in N
indents the print positions of an input segment N columns on output. This control argument only affects the path immediately preceding it. If this control argument is not specified, an indent of 0 is used.

NOTES: Because the overlay command uses the printer conversion programs, control characters are removed from input files except for newline (NL), backspace (BS), vertical tab (VT), and formfeed (FF).

If identical print positions containing the same characters are superimposed, a boldface type results. By following input segments with the -indent control argument, the user creates output containing columns of text.

page_trace (pgt)

SYNTAX AS A COMMAND:

pgt {N} {-control_args}

FUNCTION: prints a recent history of page faults and other system events within the calling process.

ARGUMENTS:

N
prints the last N system events (mostly page faults) recorded for the calling process. If N is not specified, all the entries in the system trace list for the calling process are printed. Currently, there is room for approximately 300 entries in the system trace array.

CONTROL ARGUMENTS:

-from STR, -fm STR
searches the trace array for a user marker matching STR. If one is found, printing begins with it; otherwise, printing begins with the first element in the array.

-long, -lg
prints full pathnames where appropriate. The default is to print only entrynames.

-no_header, -nhe
suppresses the header that names each column. The default is to print the header.

-output_switch swname, -osw swname
writes all output on the I/O switch named swname, which must already be attached and open for stream_output. The default is to write all output on the user_output I/O switch.

-to STR
stops printing if a user marker matching STR is found. The default is to print until the end of the array. If both -from and -to are specified, the from marker is assumed to occur before the to marker.

page_trace (pgt)

NOTES ON OUTPUT FORMAT: The first column of output describes the type of trace entry. An empty column indicates that the entry is for a page fault. The second column of output is the real time, in milliseconds, since the previous entry's event occurred. The third column (printed for page faults only) is the ring number in which the page fault occurred. The fourth column of output contains the page number for entries, where appropriate. The fifth column gives the segment number for entries, where appropriate. The last column is the entryname (or pathname) of the segment for entries, where appropriate.

NOTES: Since it is possible for segment numbers to be reused within a process, and since only segment numbers (not entrynames or pathnames) are kept in the trace array, the entrynames and pathnames associated with a trace entry may be for previous uses of the segment numbers, not the latest ones. In fact, the entry and pathnames printed are the current ones appropriate for the given segment number.

For completeness, events occurring while inside the supervisor are also listed in the trace. The interpretation of these events sometimes requires detailed knowledge of the system structure; in particular, they may depend on activities of other users. For many purposes, the user will find it appropriate to identify the points at which the supervisor was entered and exited and ignore the events in between.

Typically, any single invocation of a program does not induce a page fault on every page touched by the program, since some pages may still be in primary memory from previous uses or use by another process. It may be necessary to obtain several traces to fully identify the extent of pages used.

A count value (N) and either the -from or -to control argument cannot be specified in the same invocation of the page_trace command.

SYNTAX AS A COMMAND:

```
path path
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[path path]
```

FUNCTION: returns the absolute pathname represented by the path argument.

EXAMPLES: Assume that the user's working directory is >udd>Demo>JRSmith.

```
! string [path no_seg]
   >udd>Demo>JRSmith>no_seg
! string [path <no_seg]
   >udd>Demo>no_seg
! string [path >no_seg]
   >no_seg
```

picture (pic)

SYNTAX AS A COMMAND:

pic pic_string values {-control_arg}

SYNTAX AS AN ACTIVE FUNCTION:

[pic pic_string values {-control_arg}]

FUNCTION: returns one or more values processed through a specified PL/I picture.

ARGUMENTS:

pic_string
is a valid PL/I picture as defined in the PL/I Reference Manual and the PL/I Language Specification.

values
are strings having data appropriate for editing into the picture. Each value must be convertible to the type implied by the picture specified. If multiple values are presented, the results are separated by single spaces. Any resulting value that contains a space is quoted.

CONTROL ARGUMENTS:

-strip
removes leading spaces from edited picture values; removes trailing zeros following a decimal point; removes a decimal point if it would have been the last character of a returned value.

NOTES: For more information on PL/I picture and picture strings, see the PL/I Reference Manual, Order No. AM83 or the PL/I Language Specification, Order No. AG94.

EXAMPLES:

```
! create file_([pic 999 [index_set 8 14]])
! list file_*

   Segments = 7, Lengths = 0

   r w  0  file_014
   r w  0  file_013
   r w  0  file_012
   r w  0  file_011
   r w  0  file_010
   r w  0  file_009
   r w  0  file_008

! string [pic zzzzz9v.9999 000305.000]
   305.0000
! string [pic zzzzz9v.9999 000305.000 -strip]
   305
```

SYNTAX AS A COMMAND:

pl1 path {-control_args}

FUNCTION: invokes the PL/I compiler to translate a segment containing the text of a PL/I source program into a Multics object segment.

ARGUMENTS:

path
is the pathname of a PL/I source segment that is to be translated by the PL/I compiler. If path does not have a suffix of pl1, one is assumed. However, the suffix pl1 must be the last component of the name of the source segment.

CONTROL ARGUMENTS:

-brief, -bf
causes error messages written into the user_output I/O switch to contain only an error number, statement identification, and (when appropriate) the identifier or constant in error. In the normal, nonbrief mode, an explanatory message of one or more sentences is also written, followed (in most cases) by the text of the erroneous statement.

-brief_table, -bftb
generates a partial symbol table consisting of only a statement table that gives the correspondence between source line numbers and object locations for use by symbolic debuggers. The table appears in the symbol section of the object segment produced for the compilation. This control argument does not significantly increase the size of the object program.

-check, -ck
is used for syntactic and semantic checking of a PL/I program. Only the first three phases of the compiler are executed. Code generation is skipped, as is the manipulation of the working segments used by the code generator.

-check_ansi
generates an error message of severity 1 for each construct the compiler detects that is allowed by Multics PL/I but not by the ANSI standard X3.53-1976.

-list, -ls
produces a source program listing with symbols, followed by an assembly-like listing of the compiled object program. Use of the -list control argument significantly increases compilation time and should be avoided whenever possible by using the -map control argument.

-long_profile, -lpf
generates additional code that records the virtual CPU time and number of page faults for each source statement. It is incompatible with the -profile control argument. The new profile command can handle both regular and long profiles. Use of this feature adds considerable CPU overhead to heavily executed code. The extra CPU time is subtracted out, so that it does not appear in the report generated by the profile command.

-map
produces a source program listing with symbols, followed by a map of the object code generated by the compilation. The -map control argument produces sufficient information to allow the user to debug most problems online.

-optimize, -ot
invokes an extra compiler phase just before code generation to perform certain optimizations, such as the removal of common subexpressions, which reduces the size and execution time of the object segment. Use of this control argument adds 10% to 20% to the compilation time.

-profile, -pf
generates additional code to meter the execution of individual statements. Each statement in the object program contains an additional instruction to increment an internal counter associated with that statement. After a program has been executed, the profile command can be used to print the execution counts. See the profile command in this document.

-separate_static, -ss
causes the compiler to generate separate sections in the object segment created for the linkage information and the internal static variables. The default is to place internal static variables in the linkage section since both types of data are perprocess and writable. The -separate_static control argument is useful primarily for programs that are prelinked and can therefore share the linkage section with other users.

-severityN, -svN
causes error messages whose severity is less than N (where N

is 1, 2, 3, or 4) to not be written into the user_output switch although all errors are written into the listing. If this control argument is not specified, a severity level of 1 is assumed. For a description of severity levels, see "Notes on Error Diagnostics" below.

-single_symbol_list, -ssl
reformats the symbol table produced by the -map or -list control argument to be one, single, alphabetized list. If this control argument is not given, the default is to separate the symbols into 4 lists, arranged by declaration type. This control argument has been added at the request of users who find it easier to look up names in one list rather than 4.

-table, -tb
generates a full symbol table for use by symbolic debuggers. The symbol table is part of the symbol section of the object program and consists of two parts: a statement table that gives the correspondence between source line numbers and object locations, and an identifier table containing information about every identifier actually referenced by the source program. This control argument usually causes the object segment to become significantly longer.

LIST OF ADDITIONAL CONTROL ARGUMENTS:
The following control arguments, while available, are probably not of interest to every user.

-debug, -db
leaves the list-structured internal representation of the source programs intact after a compilation. This control argument is used for debugging the compiler. The command pl1$clean_up can be used to discard the list structure.

-time, -tm
prints a table after compilation, a table giving the time (in seconds), the number of page faults, and the amount of free storage used by each of the phases of the compiler. This information is also available from the command pl1$times invoked immediately after a compilation.

NOTES: The only result of invoking the pl1 command without control arguments is to generate an object segment.

A successful compilation produces an object segment and leaves it in the user's working directory. If an entry with that name already exists in the directory, its access control list

(ACL) is saved and given to the new copy. Otherwise, the user is given re access to the segment with ring brackets v,v,v where v is the validation level of the user's process.

If the user specifies the -map or -list control arguments, the pl1 command creates a listing segment in the working directory and gives it a name consisting of the entryname portion of the source segment with a suffix of list rather than pl1 (e.g., a source segment named valid.pl1 has a listing segment named valid.list). The ACL is as described for the object segment except that the user is given rw access to the newly created segment. Previous copies of the object segment and the listing segment are replaced by the new segments created by the compilation.

Include files contained in the PL/I source segment to be translated by the PL/I compiler are located by use of the translator search list. For more information on the search list, see the "Search List" section below.

A listing segment is optionally produced. These results are placed in the user's working directory. This command cannot be called recursively.

For information on PL/I, refer to the Multics PL/I Language Specification manual, Order No. AG94 and the Multics PL/I Reference Manual, Order No. AM83.

NOTES ON SEARCH LIST: The PL/I compiler uses the translator search list which has the synonym trans. For more information on search lists, see the search facility commands and, in particular, the add_search_paths command description in this manual.

NOTES ON ERROR DIAGNOSTICS: The PL/I compiler can diagnose and issue messages for about 350 different errors. These messages are graded in severity as follows:

1 Warning only. Compilation continues without ill effect.

2 Correctable error. The compiler remedies the situation and continues, probably without ill effect. For example, a missing end statement can be corrected by appending the

string ";end;" to the source. This action does not, however, guarantee the correct results.

3 An uncorrectable but recoverable error. That is, the program is definitely in error and cannot be corrected but the compiler can and does continue executing up to the point just before code is generated. Thus, any further errors are diagnosed. If the error is detected during code generation, code generation is completed although the code generated is not correct. After the compilation, a message is printed to the error_output I/O switch to inform the user that an error of severity 3 has occurred.

4 An unrecoverable error. The compiler cannot continue beyond this error. The message is printed and then control is returned to the pl1 command unwinding the compiler. The command writes an abort message into the error_output I/O switch and returns to its caller.

Error messages are written into the user_output I/O switch as they occur. Thus, a terminal user can quit the compilation immediately when an error message is printed. As indicated above, the user can set the severity level so as not to be bothered by minor error messages. The user can also specify the -brief control argument so that the messages are shorter. An example of an error message in its long form is:

```
ERROR 158, SEVERITY 2 ON LINE 30
A constant immediately follows the identifier "zilch".
SOURCE:   a = zilch 4;
```

If the -brief control argument is specified, the user sees instead:

```
ERROR 158, SEVERITY 2 ON LINE 30
  "zilch"
```

The -severity 3 control argument suppresses this message entirely.

Once a given error message is printed on the user's terminal in the long form, all further instances of that error message are printed in the short form.

If a listing is being produced, the error messages are also written into the listing segment. They appear, sorted by line number, after the listing of the source program. No more than 100 messages are printed in the listing.

NOTES ON SEVERITY VALUES: The pl1 command associates the following severity values to be used by the severity active function:

Value	Meaning
0	No compilation yet or no error.
1	Warning.
2	Correctable error.
3	Fatal error.
4	Unrecoverable error.
5	Could not find source.

NOTES ON LISTING: The listing created by the pl1 command begins with a line-numbered image of the source segment. This is followed by a table of all of the names declared within the program. The names are categorized by declaration type as follows:

1. declare statement
2. explicit context (labels, entries, and parameters)
3. implicit context

Within these categories, the symbols are sorted alphabetically and then listed with their location; storage class; data type; size or precision; level; attributes such as initial, array, internal, external, aligned, and unaligned; and a cross-reference list. Next is a table of the program's storage requirements and the reasons why a block is nonquick. Next is a listing of internal static variables, if any exist, sorted by offset, and a listing of automatic variables, if any, sorted by block and offset. Next is a listing of external operators used, external entries called, and external variables referenced by the program. The symbol listing is followed by the error messages, if any.

The object code map follows the list of error messages. This table gives the starting location in the text segment of the

instructions generated for statements starting on a given line. The table is sorted by ascending storage locations.

Finally, the listing contains the assembly-like listing of the object segment produced (if -list is specified). The executable instructions are grouped under an identifying header that contains the source statement that produced the instruction. Operation code, base-register, and modifier mnemonics are printed beside the octal instruction. If the address field of the instruction uses the IC (self-relative) modifier, the absolute text location corresponding to the relative address is printed on the remarks field of the line. If the reference is to a constant, the octal value of the first word of the constant is also printed. If the address field of the instruction references a symbol declared by the user, its name appears in the remarks field of the line.

SYNTAX AS A COMMAND:

pa paths {pl1_args} {dp_args} {-control_args}

FUNCTION: submits an absentee request to perform PL/I compilations.

ARGUMENTS:

paths
are the pathnames of segments to be compiled.

pl1_args
are one or more control arguments accepted by the pl1 command.

dp_args
are one or more control arguments (except -delete, -dl) accepted by the dprint command.

CONTROL ARGUMENTS:

-queue N, -q N
specifies in which priority queue the request is to be placed ($N \leq 3$). The default queue is 3; the listing segment is also dprinted in queue N.

-hold, -hd
specifies that pl1_abs should not dprint or delete the listing segment.

-limit N, -li N
specifies a time limit in seconds for the absentee job. The default value is defined by installation site.

-output_file path, -of path
specifies that absentee output is to go to the segment whose pathname is path.

NOTES: The absentee process for which pl1_abs submits a request compiles the segments named and dprints and deletes the listing segments. If the -output_file control argument is not specified, an output segment, path.absout, is created in the user's working directory (if more than one path is specified, only the first is used). If none of the segments to be compiled can be found, no absentee request is submitted.

pl1_abs (pa)

Control arguments and segment pathnames can be mixed freely and can appear anywhere on the command line after the command. All control arguments apply to all segment pathnames. If an unrecognizable control argument is given, the absentee request is not submitted.

Unpredictable results can occur if two absentee requests are submitted that could simultaneously attempt to compile the same segment or write into the same absout segment.

When doing several compilations, it is more efficient to give several segment pathnames in one command rather than several commands. With one command, only one process is set up. Thus the dynamic intersegment links that need to be snapped when setting up a process and when invoking the compiler need be snapped only once.

SYNTAX AS A COMMAND:

plus num_args

SYNTAX AS AN ACTIVE FUNCTION:

[plus num_args]

FUNCTION: returns the sum of num_args. If no num_args are specified, 0 (the additive identity) is returned.

SYNTAX AS A COMMAND:

pr path {optional_args}

FUNCTION: prints a specified ASCII segment on the user's terminal.

ARGUMENTS:

path
is the pathname of a segment to be printed. The star convention is NOT allowed.

LIST OF OPTIONAL ARGUMENTS:

begin
is an optional line number that identifies where printing begins. If it is not specified, printing starts on the first line of the segment (after an identifying header, see "Notes" below). If begin is not specified, end cannot be specified either.

end
is a line number that identifies where printing ends. If not specified, printing ends with the last line of the segment followed by two blank lines and a ready message. If the line number specified is greater than the number of lines in the segment, the rest of the segment is printed.

ACCESS REQUIRED: The user must have read access to the segment to be printed.

NOTES: Unless the user specifies a range of line numbers, the command prints the entire segment. Multisegment files cannot be printed by invoking the print command.

If the begin argument is not supplied, a short identifying header followed by two blank lines is printed preceding the printing of the segment. See "Examples" below.

The command assumes that newline characters are appropriately embedded in the text. Output is written through the I/O

switch, user_output, that is usually directed to the user's terminal.

EXAMPLES:

The command line:

```
! pr alpha
```

prints the segment alpha in the user's working directory in its entirety.

The command line:

```
! pr alpha 1
```

has the same effect, but omits the identifying header.

The command line:

```
! pr alpha 10 20
```

prints lines 10 through 20 of the segment.

The command line:

```
! pr alpha 10
```

prints lines 10 through the end of the segment.

The command line:

```
! pr alpha 1 10
```

prints the first ten lines of the segment.

print_attach_table (pat)

SYNTAX AS A COMMAND:

pat {switch_names} {-control_args}

SYNTAX AS AN ACTIVE FUNCTION:

[pat {switch_names} {-control_args}]

FUNCTION: prints information on the user's terminal about the I/O switch name associations created by attach calls in the user's current ring.

When invoked as an active function, pat returns the switch names selected by the switch_names and control arguments.

ARGUMENTS:

switch_names
are the names of I/O switches. The star convention is allowed. The -name control argument can be used to inhibit star name processing or to supply a switch name that appears to be a control argument. Information about only the specified switches is printed. If a specified switch is not currently attached, a message to that effect is printed on the user's terminal.

CONTROL ARGUMENTS:

-brief, -bf
suppresses the printing of information for the four standard switch_names (user_i/o, user_input, user_output, and error_output) even if they match a starname.

-attached, -att
prints the state of those switches that match the starnames only if they are currently attached. This is the default. Only one of -attached, -all, or -open can be specified.

-all, -a
prints the state of all the selected switches that match the starnames, whether they are attached or not. This control argument cannot be used with -attached or -open.

-open
prints the state of those switches that match the starnames

print_attach_table (pat)

only if they are attached and open. This control argument cannot be used with -attached or -all.

-name switch_name, -nm switch_name
forces the switch_name to be interpreted as a literal switch_name, even if it looks like a starname or a control argument.

NOTES: The attach and open descriptions associated with the indicated switch names are printed if pat is invoked as a command.

If no arguments are specified, the information for all switches currently attached is printed.

For further information, refer to the description of io_call.

print_auth_names (pan)

SYNTAX AS A COMMAND:

pan {-control_args}

FUNCTION: prints the names of the sensitivity levels and access categories defined for the installation.

CONTROL ARGUMENTS:

-level
lists only the sensitivity levels.

-category, -cat
lists only the access categories.

-brief, -bf
suppresses the title and headings.

-all, -a
lists all possible names (above system high).

NOTES: Only the names that can be used to describe an access class or access authorization between system low and system high are printed, unless the -all control argument is specified.

This command lists the names that are acceptable to the convert_authorization_ subroutine (described in the MPM Subroutines) to define an access class or access authorization. (All commands and system interfaces that use a character string to describe an access class use this subroutine.) Both the long and short names are printed.

print_default_wdir (pdwd)

SYNTAX AS A COMMAND:

pdwd

FUNCTION: prints out the pathname of the current default working directory on the user's terminal.

NOTES: See also the descriptions of change_wdir, default_wdir, and change_default_wdir in this manual.

SYNTAX AS A COMMAND:

prm {address} {-control_args}

FUNCTION: prints all the messages in a mailbox, querying the user whether to delete each one.

ARGUMENTS:

address
specifies the address of a mailbox. See "List of Addresses" below. If no address is specified, the user's default mailbox is assumed.

CONTROL ARGUMENTS:

-brief, -bf
suppresses the printing of the informative messages.

-interactive_messages, -im
operates on interactive messages from send_message as well as mail messages from send_mail. This is the default.

-list, -ls
prints a summary of the messages in the mailbox before entering the request loop.

-no_interactive_messages, -nim
operates on send_mail messages, not on interactive messages sent by send_message.

LIST OF ADDRESSES:

-pathname path, -pn path
where path is the pathname of a mailbox. The mbx suffix is assumed.

-user Person_id.Project_id
specifies the Person_id and Project_id of a user whose mailbox is to be read.

STR
is any argument not beginning with a minus (-) sign, and is interpreted as -pathname STR if it contains > or < characters. Otherwise, it is interpreted as -user STR.

NOTES ON QUERY RESPONSES: After printing each message, print_mail asks the question:

print_mail: Delete message N?

where N is the number of the message just printed.

Five responses are allowed:

- If the user answers "yes", the message is deleted and the next one is printed.
- If the user answers "no", the message is not deleted and the next one is printed.
- If the user answers "reprint", the message just printed is printed again, and the question is asked again.
- If the user answers "quit" or "q", print_mail returns the user to command level after deleting the messages specified.
- If the user answers "abort", print_mail returns to command level, without deleting any messages.

NOTES ON CREATING A MAILBOX: A default mailbox is created automatically the first time a user issues print_mail, read_mail, accept_messages, or print_messages. The default mailbox is:

>user_dir_dir>Project_id>Person_id>Person_id.mbx

NOTES ON EXTENDED ACCESS: Access on a newly created mailbox is automatically set to adrosw for the user who created it, aow for *.SysDaemon.*, and aow for *.*.*. The types of extended access for mailboxes are:

add	a	add a message.
delete	d	delete any message.
read	r	read any message.
own	o	read or delete only your own messages.
status	s	find out how many messages are in the mailbox.
wakeup	w	send a wakeup when adding a message.

The modes "n", "null", and "" specify null access.

print_mail (prm)

NOTES ON RELATED COMMANDS: Special commands exist to create additional mailboxes and to change the attributes of mailboxes. These commands, described in the MPM Subsystem Writers' Guide, are:

mbx_create	create a mailbox.
mbx_delete	delete a mailbox.
mbx_add_name	add a name to a mailbox.
mbx_delete_name	delete a name from a mailbox.
mbx_rename	rename a mailbox.
mbx_list_acl	list the access control list of a mailbox.
mbx_set_acl	change or add entries to the ACL of a mailbox.
mbx_delete_acl	delete entries from the ACL of a mailbox.
mbx_set_max_length	set the maximum length of a mailbox.
mbx_safety_switch_on	turn on the safety switch of a mailbox.
mbx_safety_switch_off	turn off the safety switch of a mailbox.

See also the read_mail and send_mail command in this manual for a complete description of the mail facility.

EXAMPLES: In the following example, the user's input is preceded by a ! character.

```
! prm

   #1  (5 lines)   08/29/78 20:23  Mailed by:  Smith.Pubs
   Subject:  Manual submissions

   <5 lines of text>

   print_mail: Delete message #1?  ! no

   #2  (12 lines)  08/29/78 22:47  Mailed by:  Jones.Maint
   Subject:  Hardware Shipment

   <12 lines of text>

   print_mail: Delete message #2?  ! yes

   #3  (21 lines)  09/27/78 10:45  Mailed by:  Wilson.SHARDS
   Subject:  Management questionnaire

   <21 lines of text>

   print_mail: Delete message #3?  ! q

   r 1210 0.091 82
```

SYNTAX AS A COMMAND:

pm {destination} {-control_args}

FUNCTION: prints any interprocess messages that were received (and saved in the user's mailbox) while the user was not accepting messages.

ARGUMENTS:

destination
can be of the form Person_id.Project_id to specify a mailbox. If destination contains > or <, it is the pathname of a mailbox. If no destination is specified, the user's default mailbox is assumed.

CONTROL ARGUMENTS:

-all, -a
prints all messages, including those held by -hold mode (see accept_messages). This is the default.

-call cmdline
for each message, instead of printing calls the command processor with the line:
cmdline number sender time message {path}
For more details, see the accept_messages command.

-last, -lt
reprints only the latest message received.

-long, -lg
prints the sender and date-time of every message, even when the same for two consecutive messages.

-new
when accept_messages -hold mode is in effect, prints only those messages that have not been printed before. The default is to print all held messages.

-pathname path, -pn path
specifies a mailbox by pathname. The mbx suffix is assumed. This control argument and the destination argument are mutually exclusive.

print_messages (pm)

-short, -sh
prints messages as with accept_messages -short, omitting redundant sender names in favor of the prefix "=:". This is the default.

NOTES: Messages are deleted after they are printed unless the -hold argument was given to the accept_messages command. The "last" message remains available for the life of the process or until redefined by a new message.

If messages are deferred, it is a good practice to print out pending messages periodically.

For a description of the mailbox, refer to the accept_messages and print_mail commands. See also the active functions last_message, last_message_sender, and last_message_time.

print_motd (pmotd)

SYNTAX AS A COMMAND:

pmotd

FUNCTION: prints out changes to the message of the day since the last time the command was called. The print_motd command is intended to be used within a start_up.ec segment.

NOTES: When pmotd is invoked, the current message of the day is first compared with the segment Person_id.motd in the user's home directory. Next, all lines that have been appended or modified since the last time pmotd was invoked are printed on the terminal. Then, the Person_id.motd segment is updated for use the next time print_motd is invoked.

If the segment Person_id.motd does not exist, print_motd attempts to create it, prints the current message of the day, and initializes Person_id.motd.

print_proc_auth (ppa)

SYNTAX AS A COMMAND:

ppa {-control_args}

FUNCTION: prints the access authorization of the current process and current system privileges (if any).

LIST OF OPTIONAL ARGUMENTS:

-long, -lg
prints the site-defined long names (up to 32 characters) for the sensitivity levels and categories.

-all, -a
prints the maximum access authorization of this process.

NOTES: If the -long control argument is not specified, the access authorization printed is composed of the site-defined short names (eight characters or less) for sensitivity levels and categories.

The maximum authorization printed for the -all control argument is the maximum authorization that this process could have been given at login, and corresponds to the maximum access class of upgraded directories that can be created by this process.

print_request_types (prt)

SYNTAX AS A COMMAND:

prt {-control_args}

FUNCTION: prints a list of all request types handled by the I/O daemon.

CONTROL ARGUMENTS:

-brief, -bf
suppresses printing of a heading line.

-access_name STR, -an STR
prints only those request types having an access name of STR of the form Person_id.Project_id.

-gen_type STR, -gt STR
prints request types of generic type STR.

NOTES: For each request type, two items of information are printed: the access name of the I/O daemon driver process that performs requests of that type, and the generic type to which the request type belongs. An asterisk (*) immediately preceding a request type indicates that the request type is the default for its generic type.

The access name IO.SysDaemon indicates a standard request type available to all users. Any other access name indicates a nonstandard request type that is generally not available to all users.

The generic type of a request type determines which commands can be used to submit requests of that request type. For example, the dprint command uses only request types of generic type "printer". The dpunch command uses only request types of generic type "punch".

print_search_paths (psp)

SYNTAX AS A COMMAND:

psp {search_lists} {-control_arg}

FUNCTION: prints the search paths in the specified search lists.

ARGUMENTS:

search_list
is the name of a search list. If no search lists are specified, all search lists referenced in this process are printed.

CONTROL ARGUMENTS:

-expanded, -exp
specifies that all keyword search paths except -referencing_dir, and all unexpanded search paths, are printed as absolute pathnames.

NOTES: All synonyms of a search list name are printed if no search lists are specified.

For a complete list of the search facility commands, see the add_search_paths command description in this manual.

print_search_rules (psr)

SYNTAX AS A COMMAND:

psr

FUNCTION: prints the object segment search rules currently in use.

NOTES: See also the descriptions of the set_search_rules, add_search_rules, and delete_search_rules commands. The standard search rules are described under "Search Rules" in the MPM Reference Guide.

print_wdir (pwd)

SYNTAX AS A COMMAND:

pwd

FUNCTION: prints the pathname of the current working directory.

NOTES: A working directory is a directory in which the user's activity is centered. Its pathname is remembered by the system so that the user need not type the absolute pathname of segments inferior to that directory.

See the descriptions of the change_wdir, change_default_wdir, and working_dir commands. See also "Search Rules" and "Pathnames" in the MPM Reference Guide.

SYNTAX AS A COMMAND:

pb {procedure_name}

FUNCTION: provides symbolic, interactive debugging facilities for programs compiled with PL/I, FORTRAN, or COBOL. Its features permit a user to interrupt a running program at a particular statement, examine and modify program variables in their initial state or during execution, examine the stack of block invocations, and list portions of the source program. External subroutines and functions can be invoked, with arguments as required, for execution under probe control. The probe command can be called recursively.

ARGUMENTS:

procedure_name
is an optional argument that gives the pathname or reference name of an entry to the procedure or subroutine that is to be examined with probe.

OVERVIEW OF PROCESSING: When probe has been invoked, it accepts requests from the user. A probe request consists of a keyword (or its abbreviation) that specifies the desired function and any arguments required by the particular request. Requests are separated from each other by newlines or semicolons.

A series of requests can be given in the form of a request list. This is used in breakpoint request lists and conditional execution lists. Here, each request is separated by semicolons. An example:

```
value a; v b; continue
```

Probe at all times has a "current language". It communicates with the user in terms appropriate to the language of the procedure being examined. The syntax of an expression and the form of probes output vary from language to language.

To use probe to the fullest, a program must be compiled so that the object segment produced has both a symbol table and a statement map (these terms, and others, are defined below in "Notes on Terminology"). A symbol table and statement map are produced for the languages supported if the -table control

argument is given to the compiler. A program can also be compiled with the -brief_table control argument, which produces only a statement map. In this case the user can retrieve information about source statements and where the program was interrupted, and can set breakpoints, but can do little else.

PROBE POINTERS: Two internal "pointers" are used by probe to keep track of the program's state. They are the "source" pointer and the "control" pointer.

The source pointer identifies a line, a block, and a frame. A line is a source program line number. The language of the source line is the language probe will use with the user. The meaning of a block depends on the language. For a PL/I program, it specifies the smallest begin block or procedure that contains the source line. For a FORTRAN program it specifies the program or subprogram the statement occurs in. For a COBOL program it indicates the program-id of the containing program. The frame specifies a stack frame associated with the block. When there are several invocations of the same block on the stack, the frame distinguishes between them. If there is no activation of the block, the frame portion of the source pointer is null. In this case, certain types of storage (i.e. PL/I automatic) are not defined. The initial value of the source pointer is determined by the initial value of the control pointer.

The control pointer indicates the last location executed before probe was invoked. The initial value depends on the manner probe was invoked.

1) If probe is invoked from a breakpoint, the control pointer is set to the line where the break occurred.

2) If probe is invoked from the command line, then if a procedure_name is specified, then if the procedure is active, the control pointer is set to the last line executed in the most recent invocation of that procedure.

3) If the procedure in the command line is not active, the control pointer is set to the entry statement for the procedure.

4) If no procedure_name is specified, then if there is a QUIT signal or condition frame on the stack, the control pointer

is set to the location being executed when the condition was signalled.

5) If no procedure_name is specified, and there are no condition frames on the stack, the last line executed in the most recent frame is used. (This is usually the command processor).

Information about programs being debugged is stored by probe in a segment in the user's home directory called Person_id.probe, where Person_id is the user's log-in name. This segment is created automatically when needed. This segment should not be deleted, or probe will be unable to reset any breaks it has set.

RESTRICTIONS ON INPUT LINES: A probe input line cannot contain unbalanced parenthesis or unbalanced quotes. This means that a request or request list as typed in must fit on one line. It cannot contain a newline character. If a long line must be typed, the Multics escape convention of placing a backslash before the newline can be used. If the newline character is needed (in a character string constant, for example), the escape sequence \012 can be entered instead.

PROBE REQUESTS: The following pages present the format and function of each probe request, giving first the name of the request, then its abbreviated form, if any, and its arguments, required and optional. The syntax of the arguments is described in the following way:

Curly braces enclose optional material

Where the user can select only one of several options, square brackets enclose the list of choices, and the choices are separated by a vertical bar.

Upper case names represent items whose syntax is elsewhere defined (for example, EXPRESSION or PLACE).

Each request that takes arguments is shown with examples of its use. Examples may be in the syntax used for PL/I, FORTRAN, or COBOL. If an example does not make sense to you, it may be in another language.

SAMPLE REQUEST WRITE-UP:

● plugh,pl
plugh {fault} {-all}
plugh {N, {M}} {-all}

This request accepts either the word "fault", or a number (N), or two numbers separated by a comma (N, M); in any of these cases it can also be supplied with the control argument "-all".

The syntax can be written more briefly as:

```
plugh {[N {,M} | fault ]} {-all}
```

or more verbosely, as:

```
plugh  {-all}
plugh N {-all}
plugh N,M {-all}
plugh fault {-all}
```

Legal uses of the "plugh" request include:

```
plugh 4
plugh 4,712 -all
plugh fault -a
```

Illegal uses include:

```
plugh ,275        M was used without N
plugh 3 fault     fault and N can't be together
plugh 17+42,4     17+42 is not a number
plugh N           N is not a number
```

The following items are used throughout the requests section:

N, M
are positive, unsigned integers.

OBJECT
is a path name or reference name of an entry point into some object segment.

REQUEST
is any probe request (or list of requests).

PATH
is a Multics pathname.

LINE
is a line of program text in a source segment and/or the set of instructions in the object segment corresponding to that text. It is defined below under "Syntax of a Line".

EXPRESSION
is an expression, defined below under "Syntax of an Expression".

STRING
is a quoted string. Its beginning and end are delimited by quotes (""). If a quote character is to be included in the string, two quotes should be used. (i.e. "this is a quote: "" character"). When the current language is FORTRAN, the quote character and the apostrophe (') are considered equivalent. Either can delimit a string. For example, "preceded by itself, yields falsehood' is a FORTRAN string.

LIST OF BASIC REQUESTS:

● .

This request causes probe to identify itself by printing "probe" and the current version number on the terminal. It may be used, for example, to determine if a called routine has returned. If the current invocation of probe is not the first invocation of probe on the stack, the recursive depth is also printed. The version number is useful for determining whether the version of probe being used has certain features or bug-fixes. It should always be included in any trouble report about probe.

● help
help
help *
help TOPIC

If the help request is invoked with no argument it prints some general information about probe. If it is invoked with an asterisk (*), it prints a list of all topics; otherwise it prints information about TOPIC, if there is any.

Examples:

help
help expressions

- list_requests, lr

  This request prints a list of all the probe requests.

- list_help, lh
  list_help

  This request prints the names of all topics for which there is information. It is exactly the same as "help *"

- ..
  .. COMMAND LINE

  This request passes the remainder of the line (COMMAND LINE) directly to the Multics command processor. It can never be used in a break request list or a conditional execution list. When used, it must be the first request on the line.

  Example:
  ..wd; ls *.pl1

- value, v
  value [EXPRESSION | CROSS-SECTION]

  The value of the given EXPRESSION or the array elements specified by CROSS-SECTION are displayed.

  A CROSS-SECTION is specified by giving the upper and lower bounds of one or more subscripts. An asterisk can be used, which is equivalent to a cross-section from the lowest to highest subscript of an array.

  Examples:
  value arr (1:5, 3:7)
  value p -> a.b(j)
  value a of b of lrec
  value ijptr(*,3)

  The value request can be used with PL/I structures or COBOL records, in which case the value of every component is displayed as well.

  The value request cannot be used with PL/I areas.

  External functions can be called with the value request. The argument list can involve arbitrary expressions, and the arguments are converted to the proper type, if the called function specifies what type of arguments are expected.

The value request can print identifier names in short or long form, under user control. See the modes request.

- let, l
  let [VARIABLE ¦ CROSS-SECTION] = EXPRESSION

  This request sets the specified variable or array elements to the value of the expression. If the variable and expression are of different types, conversion is performed according to the rules of PL/I. Array cross-sections are expressed as in the value request. One array cross-section cannot be assigned to another, nor can structures be assigned to as a whole. Certain PL/I data types can only be assigned to identical types. For example, areas can only be assigned to areas, and files can only be assigned to files.

  Note that because of unpredictable compiler optimization, the change cannot take effect immediately, even though the value request shows that the variable has been altered.

- quit, q
  quit

  This request causes the current level of probe to return. If there is more than one invocation of probe on the stack, the user may still be in probe. If there is only one, this request causes a return to command level.

SOURCE REQUESTS: The source pointer is used to indicate a block in a program (to resolve variable name conflicts) and a stack frame (to resolve separate invokations of a block), and a statement (to be printed). Its current value can be displayed with the "where" request, its value can be changed by the "position" or "use" request. The source line pointed to can be printed via the "source" request.

- where
  where
  where source
  where control

  The where request displays the values of the probe pointers. If it is invoked with no argument it displays the values of both, otherwise it displays the value of the pointer named. You can abbreviate "source" as "sc" and "control" as "ctl".

- use, position, ps
  use
  use LINE
  use {+|-}N
  use level N
  use OBJECT
  use STRING

  The use request selects the block, and the position request selects the line number, to be used for subsequent probe requests. If invoked as position, the line positioned to is displayed. If invoked as use, there is no display.

  If no argument is supplied to use, the source pointer is reset to its initial value, which is the value of the control pointer.

  The new value of the source pointer can be given in a variety of ways:

  - absolutely, by giving the LINE within the current procedure.
  - relatively, by giving +N or -N. The new source pointer value is the statement N statements after or before the current statement.
  - a new stack frame can be specified by level N, where N is the number of the stack frame of interest. If too high a number is given, the highest numbered frame is used. "level" can be abbreviated as "lv".
  - a new procedure can be specified by giving its path or reference name (OBJECT).
  - by searching: the user can request that probe search through a source segment for an executable line containing STRING. Null string ("") causes probe to use the last search string. It is an error to use an empty quoted string expression if there has not been a previous quoted string. If the searching fails, the source pointer is not changed.

  Examples:

  use level 4
  specify level number - last line executed at level 4.

  ps -4
  statement four statements before current one.

ps 2-34
include file 2, line 34, current procedure.

ps label
set to line whose label is "label".

ps "label:"
search for line containing the word "label" followed by a colon. In effect, the same as the previous example.

Note that probe deals with executable statements, not source lines. The source pointer cannot be set to a source line for which no instructions are executed. This includes blank lines, comment lines, declarations, and COBOL declarative procedures. It is not possible to search for non-executable lines either.

Note also that in PL/I, names, including labels, are not known (available for probe) when the source pointer is outside the block in which they are declared. This means that probe cannot find a label in an internal procedure or begin block unless that block is currently the "current block". The label can still be found by a search for a character string that appears in the labelled line, for example, "label:".

● source, sc
source {N}
sourch path PATH

The first form of the source request prints source lines of the current procedure, beginning with the current source line. If N is supplied, N lines are printed, otherwise the current line is printed. The source pointer remains unchanged.

A statement can take up many lines, and there may be blank lines (or non-executable source lines, such as comments or declarations) between statements. Although the source pointer can only be set to a line for which code is generated, these lines can be displayed along with the statements. If N statements are displayed, any non-executable lines between the first and the last are also displayed.

The second form of this request is used to supply the pathname of the source segment for the current procedure. Multics object segments contain within them the absolute pathnames of the source segments used to compile them. Sometimes these

segments have been moved by the time the object segment is being debugged, and when probe attempts to locate them, it will fail. When it does, it informs the user that the source cannot be located, and the user can supply probe with the path of the source by giving it after the path argument.

BREAK REQUESTS: The next requests deal with setting, printing information about, and resetting breakpoints. A breakpoint is a list of one or more probe requests associated with a source statement. When the statement of a breakpoint is executed, the user program is suspended, and probe executes the requests associated with the breakpoint. When the requests of the breakpoint have executed, the program resumes (unless one of the requests was a "quit" or a "goto"). A breakpoint is said to be "set" either "before" or "after" a given line number. A break before a statement is executed before the statement itself is executed. A break after a line occurs after the line has been executed. When a transfer is made to statement x, a break before statement x is effective, but a break set after statement x-1 is not effective. A break set after a statement that transfers control (such as a goto or return) may or may not be executed.

No breaks can be set after any COBOL statement, due to restrictions of the compiler.

- after, af, a
- before, be, b

The syntax to set either kind of break is the same:

```
before {LINE} {: REQUEST }
after  {LINE} {: REQUEST }
```

LINE indicates some statement where the break is to be inserted. If none is supplied, the statement identified by the source pointer is used. A set of probe requests can be associated with the breakpoint by placing the requests after a colon. If no requests are given, "halt" is used.

Examples:

```
b 29
after foo,2: if a > 7:halt
b 52: (v f; v j; v c; continue)
```

The next two requests are used to print information about

breakpoints (status) and to reset breaks (reset). They have a very similar argument syntax. In both of them LOC is used to distinguish a break set before a LINE from one set after a given LINE. LOC can be one of: "before", "be", "b", "after", "af", "a". It is only needed when there is a break set both before and after a statement. If it is not supplied, both breaks set before and breaks set after the LINE are affected.

- status, st
  status
  status {{LOC} LINE} {-long}
  status OBJECT {-long}
  status -all {-long}
  status *

  The status request lists the breaks set by probe in various procedures. The first form selects all breaks in the current procedure, the second selects the break at LINE in the current procedure. The third form lists all breaks in the procedure specified. The fourth form lists all breaks in all segments. The last form lists the names of all segments that have have breaks set in them.

  If the control argument "-long" appears, the break request list associated with the breakpoint is printed. This is the default for the second form.

  Examples:

status >udd>Dog20>Crecho>zlotny	lists all breaks in zlotny
status 35 -lg	lists breaks before or after line 35
st b 7	lists only the break before line 7
st -all	lists all breaks in the world

- reset, r
  reset {{LOC} LINE} {-brief}
  resent OBJECT {-brief}
  reset -all {-brief}
  reset *

  This request resets breakpoints set by probe at selected lines in selected procedures. The first form resets the break at LINE in the current procedure. If LINE is not given, then if the current invocation of probe was caused by a break, that break is reset. Otherwise the break at the current line is

reset. The second form resets all breaks in the procedure specified. The third and fourth forms reset all breaks that probe has set.

As breaks are reset, the source line number and pathname of the containing segment are printed, unless -brief is given, in which case nothing is printed.

Examples:

r -all	Reset every break probe can find
r a $259,1	Reset the break after the first statement after the line labelled "259"
r -bf	Reset all breaks in the current procedure
r <Weir>estimate_prophet	Reset all breaks in procedure named

REQUESTS USEFUL IN BREAKPOINT REQUEST LISTS:

- halt, h
  halt

This request causes probe to stop processing the request list and read requests from the terminal. A new invocation of probe is created, with the control and source pointers set to the line of the breakpoint. After a subsequent continuation, probe resumes interpreting the break request list that contains the halt. When the list is empty, the user's program is resumed. This request has no effect when issued from the terminal.

Example:

b12:if K>0:halt	this breakpoint stops if K>0.
(v a; halt; v a)	this list displays the value of a before and after stopping.

- pause, pa
  pause

This request is equivalent to "halt;reset" in a break request list. It causes the procedure to execute a breakpoint once, and then reset it when execution is resumed. It has no effect if not executed in a breakpoint request list. If the user does not eventually continue the breakpoint, the break is not reset.

FLOW OF CONTROL REQUESTS: Requests are provided for selected execution of program statements. The user can resume execution after a break, call external procedures, or perform explicit "goto"s.

- continue, c
  continue

  This request restarts a program that has been suspended by a probe breakpoint. If this request is used in any other context, probe returns to its caller, which is usually command level.

- step, s
  step

  This request attempts to step through the current program one statement at a time. If the program has been stopped before line N, a break is set before line N+1. If the user is stopped after line N, the break is set before line N+2. These breaks contain "pause" as their sole request list, and thus are self-resetting. If the statements being stepped do not execute in sequence, the stepping can be unsuccessful. Note that PL/1 and FORTRAN do-loops, and conditional statements in all languages, do not execute sequentially.

- continue_to, ct
  continue_to LINE

  This request inserts a temporary breakpoint before the LINE specified, then continues. The effect is as if the user had typed the following:

  before LINE: pause
  continue

  Example:

  ```
  ct 11
  ```

- call, cl
  call OBJECT (ARGUMENTS)

  This request calls the external procedure named with the arguments given. ARGUMENTS should be a list of arguments to the called procedure, separated by commas. If the procedure

expects arguments of a certain type, those given are converted to the expected type. The value request (see above) can be used to invoke a function, with the same sort of conversion occurring. If the procedure has no arguments, an empty argument list "()" must be given.

Examples:

```
PL1:
   call sub ("abcd", p -> p2 -> bv, 25o, addr(k))
COBOL;
   call eat-master (a of b of new-unit, REC-LEVEL)
FORTRAN;
   call gamma (43, marigold(i), substr(cs,3))
```

- goto, g
  goto LINE

This request transfers control from probe to the statement specified and initiates execution at that point. The syntax of LINE is given below. It is an error to use this request to goto a line in a procedure that is not active. Because of compiler optimization, it can be dangerous to use this request.

Examples:

goto label_var	transfer to value of label variables
goto action (4)	transfer to value of label constant
goto 110	transfer to statement on line 110
goto $110	transfer to line with label 110
goto $c,1	transfer to the statement after the current one

CONDITIONAL PREDICATES: Probe provides two forms of conditional execution. The "if" request evaluates a conditional expression, and executes a request list if the expression is true. The "while" request repeatedly executes a request list, testing the conditional expression before each execution. The format of a conditional expression is:

EXPRESSION OP EXPRESSION: where OP can be <=, <, =, ^=, > or >=. When the current language is FORTRAN, .le., .lt., .eq., .ne., .gt., and .ge. are also accepted.

- if
  if CONDITIONAL EXPRESSION: REQUESTS

  This request is most useful in a break request, where it can be used to cause a conditional halt. REQUEST can be a single request, or several probe requests, enclosed in parentheses and separated by semicolons.

- while, wl
  while CONDITIONAL EXPRESSION : REQUESTS

  Examples:

```
if a < b: let p = addr(a)
while p ^= null: (v p -> r.val; let p = p -> r.next)
if ijk .ne. 8: halt
```

REQUESTS TO CONTROL PROBE: It is possible to control probe's behavior in a few ways - the length of error messages, the amount of printing done by breaks and by the value request can all be controlled. The current language can be specified explicitly. In addition, the streams used by probe for input and output can be controlled.

- modes, mode
  modes {MODES}

  The modes request sets various modes internal to probe that change the way it functions. If no arguments are given, the current modes are printed. MODES can be any combination of the following. If conflicting modes are set, the last one in the request determines the setting of the mode.

  Most modes take a single argument that is either a LENGTH or a BOOLEAN.

  A LENGTH is either "long" ("lg"), "short" ("sh"), or "brief" ("bf"), and is used to specify the kind or amount of printing to be done by a given part of probe. The amount of output produced is greatest for "long" and least for "brief", with "short" in between. In some cases, "short" and "brief" are the same.

  BOOLEAN is used to turn a mode or feature on or off. It may be either "yes", "on", or "true", or "no", "off", or "false".

error_messages, em LENGTH
controls the length of the text used for an error message. The default is long.

qualification, qf LENGTH
controls the way variable names are printed by the value request. The default is "brief", which causes only the last name of a structure to be printed. If it is "long", names are printed fully qualified. This mode only affects the printing of PL/1 names. FORTRAN names are never qualified (because the concept of qualification is not used in FORTRAN. COBOL names are always printed in the "brief" format.

value_print, vp LENGTH
controls the circumstances under which the value request prints the name of a variable. The default is "short" which prints the name only for structures or arrays. If it is "long" the name is always printed, and if it is "brief" the name is never printed.

value_separator, vs STRING
causes the value request to print STRING between the name of a variable and its value, if the name is being printed. Only the first 32 characters of STRING are used. The default is " ".

prompt BOOLEAN
controls whether or not a prompting string is printed on the terminal when probe is listening for requests. It is off by default.

prompt_string STRING
specifies the string to be used for prompting. The initial value is "probe^[(^d)^;^s^]:^2x". The STRING is used in a call to ioa_$nnl, where the first argument is a bit (1) that is on if the current invocation of probe is a recursive one, and the second is the current depth.

● input_switch, isw
isw {SWITCH}

This request causes probe to take all further command input from the switch named. If no SWITCH is supplied, user_input is used. If there are any other requests in the input line or break request list that contain this request, they are ignored without comment. Input is read from the switch until either a new input_switch request is read, or all available characters are processed, in which case a message is printed and input is reset to user_input. If any errors occur, input is reset to user_input. The switch SWITCH must be attached and open before this request is given.

- output_switch, osw
  osw { SWITCH }

  This request causes probe to direct all its output to the switch named. If SWITCH is not specified, user_output is used.

- language, lng
  language {LANG}

  If no argument is given, this request prints the name of the "current language". Otherwise LANG should be the name of one of the supported probe languages. Names accepted are: pl1, fortran, ft, and cobol.

MISCELLANEOUS REQUESTS:

- display, ds
  display {*} VARIABLE {FORMAT} {N}

  The display request displays an arbitrary location in a selected format. If an asterisk appears before VARIABLE, indirection is specified and the value of the variable specifies the address of the storage to be displayed. Otherwise, the address of VARIABLE is the address of the first location displayed. It is an error to use display with a VARIABLE that has no storage (such as a format constant) or with a literal constant, unless indirection is used, in which case VARIABLE can be an addressing constant (such as label constant), a pointer constant, or an expression with a pointer result.

  FORMAT can be one of the following:

  octal, o
  N is the number of (36 bit) words dumped.
  ascii, character, ch
  N is the number of characters dumped. A non-printable character is printed as ".".
  instruction, i
  N is the number of instructions dumped. If the instruction has descriptors, they are dumped with the instruction.
  pointer, ptr, its
  N is the number of ITS pointers displayed.

  The default FORMAT is octal, and the default for N is 1.

Examples:

```
ds * 253|100 octal 20      dumps 20 words in octal
ds foo ascii 64            displays the first 64 characters
                              of foo
```

- stack, sk
  sk {{M ,} N} {all}

  This request traces the stack backwards and displays the first N frames. If M is not given, the highest numbered frame is the first frame displayed, otherwise the M'th frame is the first displayed. If N is not given, all frames are displayed, and M cannot be specified. System support frames are not displayed unless "all" is given.

  For each block, the frame number is given, as is the name of any condition raised in the block.

  Examples:

```
stack                      traces the whole stack
stack 2                    displays the two most recent
                              frames
stack 3,2                  displays two frames starting
                              with frame 3
```

- args
  args

  The args request displays the names and values of the arguments to the current procedure.

- symbol, sb
  symbol VARIABLE {long}

  This request displays the attributes of the variable specified and the name of the block in which it is declared. If the size or dimensions of the variable are not constant an attempt is made to evaluate the size or extent expression; if the value cannot be determined an asterisk (*) is displayed instead. If "long" appears after the name of the identifier, and if the identifier is a PL/I structure or COBOL record, the attributes of all members of the structure or record are displayed as well.

- execute, e
  execute STRING

  This request passes the quoted string to the Multics command processor for execution. This request is useful in break request lists and conditional execution lists, where the .. escape cannot be used.

  Example:

```
e "ioa_ ""stopped at a break """
```

SYNTAX OF AN EXPRESSION: An expression can be made from variable references, constants, and probe builtin functions, which can be combined using the arithmetic operators +, -, *, and / for addition, subtraction, multiplication and division. Parentheses can be used to indicate order of evaluation. Operations of multiplication and division are performed first, then those of addition and subtraction.

COBOL "abbreviated" expressions are not supported.

SYNTAX OF A VARIABLE: Variables can be simple identifiers, subscripted references, structure qualified references, and locator qualified references. Subscripts can also be expressions.

Spaces are significant in the names of FORTRAN and COBOL names. A FORTRAN name cannot contain embedded spaces. Case is insignificant in COBOL names in FORTRAN names when the object segment was compiled with either "-fold" or "-card".

Examples:

COBOL:

```
data-elem
log-type of gen-record (3)
gen-record.log-type(3)
```

PL1:

```
ignatz (p -> lemma - 3)
```

The block in which a variable reference is resolved is normally determined by the source pointer, but can be altered by providing a different block in brackets after the variable name. A block can be specified in the following ways:

Example:

level N	the block and frame at level N
-N	the Nth previous invocation of the current block
LINE	the block that contains LINE, in its most recent invocation.
OBJECT	the block named. It can be internal to the current procedure, or external.

WARNING: Specifying a block explicitly does not change probe's "current language". It is possible that the block named is in another language than the current block. Even if this is so, data is referenced in terms of the current language.

SYNTAX OF A CONSTANT: The attributes of a constant are determined by the appearance of the constant. Probe recognizes arithmetic constants (fixed or floating, binary or decimal), string constants (character or bit, in any radix from 1 to 4), and pointer constants.

The maximum length of a string constant is 256 characters.

Examples:

-123	fixed dec (3)
10b	fixed bin (2)
45.37	fixed dec (4,2)
4.73e10	float dec (3)
4.21f10	fixed dec (3,-8)
2.1-0.3i	complex decimal (2,1)
12345670o	fixed bin (24) entered in octal
"abc"	character string
"quote""instring"	character string with embedded quote
"1010"b	binary bit string
"FA07"b4	hexadecimal bit string
"1222"b2	quatenary bit string
256|1200	pointer
232|7413(9)	pointer with bit offset
true	FORTRAN logical constant
'Nix Olympia'	FORTRAN string constant

Note that the segment number and word offset of a pointer are specified in octal, but the bit offset, if any, is specified in decimal.

PROBE BUILTINS: Many builtin functions are provided. They can be referenced as if they were external functions, but if no argument is needed, then the argument list can be omitted. The substr and unspec builtins can be used as pseudo-variables.

```
addr (A)
addrel (P, N)
baseptr (N)
length (S)
maxlength (S)
null ()
ptr (P, N)
rel (P)
segno (P)
substr (S, N)
unspec (A)
```

In the list above, A stands for any reference to storage, N stands for any expression that yields a number, P for any expression that yields a pointer value, and S for any expression that yields a string.

All builtins are equivalent to the Multics PL/1 builtins of the same name, except for segno and ptr. The ptr builtin is like the Multics PL/1 ptr builtin, but also can be supplied with a bit offset after the word offset. The segno builtin is like the Multics PL/1 baseno builtin, but its result is an integer instead of a bit-string.

Remember that probe reads numbers in decimal, so a reference to "baseptr(64)" means the same as "baseptr(100o)".

Builtins can be prefaced with the "$" character to distinguish them from program variables of the same name.

For the following examples, assume the following declarations, depending on your choice of language:

```
PL1:
    dcl i fixed bin;
    dcl cs char (8);
FORTRAN:
    integer i
    char*8 cs
```

COBOL:
```
    77 i usage is comp-6.
    77 cs pic a(8).
```

(No claim is made that these are exactly equivalent data, but they are close enough for discussion.)

Also assume that cs has the value "abcdef " and that i is 2.

addr (i)	the address of i
v substr (cs, i, 3)	displays "bcd"
let substr (cs, 4, 1) = " "	sets cs to "abc ef"
v length (cs)	displays 8
value maxlength(cs)	also displays 8
v baseptr (254o)	displays 254|0

SYNTAX OF A LINE:

A LINE is used by probe to define a source statement or a location in the object segment. It can be a label, a line number, or a special probe symbol. Lines in include files can be specified by giving the file number before the line number. The compilation listing specifies the correspondance between file numbers and source files. A statement can be specified relative to another statement. A label that looks like a line number may be specified by preceding it with a dollar sign. This convention must be used for all FORTRAN labels, because they are otherwise indistinguishable from numbers. COBOL also allows numeric labels, and the dollar sign must be used in this case also.

Examples:

34	line number 34
2-59	line 59 in include file 2
foo(3)	subscripted label constant
label,3	third statement after one labelled by "label"
$100	statement whose label is "100"
$c,3	the statement three statements after the current one
$b	the statement containing the breakpoint that caused the current invocation of probe

NOTES ON TERMINOLOGY

active
a procedure is said to be active if its execution is ongoing or suspended by an error, quit signal, breakpoint, or call. An active procedure is distinguished from one that has never been run, has completed execution or has been interrupted and aborted by a Multics release command, in that an active procedure has at least one stack frame associated with it.

automatic storage
a storage class for which space is allocated dynamically in a stack frame upon block invocation. As a result, variables of this class only have storage assigned to them, and hence a legitimate address and value, when the block in which they are declared is active. PL/1 variables, by default, belong to this class. FORTRAN variables must appear in an "automatic" statement in order to belong to this class.

block
corresponds to a PL/1 procedure or begin block or FORTRAN program or subroutine, and identifies a particular group of variable declarations.

breakpoint
a point at which program execution is temporarily interrupted and probe requests executed.

invocation
when a procedure is called recursively, it appears on the stack two or more times, and has storage allocated for it the same number of times. Each instance of the procedure on the stack is considered a separate and distinguishable invocation of the block. The values of automatic variables can be different in different invocations of the same block. The most recent invocation is the topmost in stack trace.

level number
an integer used by probe to uniquely designate each block invocation (i.e., each entry in a stack trace). Level one is the first (least recent) procedure invoked. Level number is NOT necessarily the same as either of the numbers given after the word "level" in a ready message. The first of this pair gives the count of command levels in effect and gives the value n+1, where n is the number of programs (or groups of programs) whose execution has been suspended, the second gives the number of stack frames in existence and since the probe stack includes quick blocks, this number is less than or equal to the level number of the last command level in the stack trace.

quick block
internal procedures and begin blocks that satisfy certain requirements (e.g., are not called recursively, do not contain on, signal, or revert statments, etc.) have their automatic storage allocated by the blocks that call them. Hence, they do not actually have their own stack frames, but share the one of the caller. Certain system commands, such as trace_stack, ignore these blocks. The probe command, however, includes them in a stack trace, and treats them as if they were the same as any other blocks. The quickness of a block can be determined from a program listing containing information about the storage requirement of the program (produced with the -symbols, -map, or -list control arguments). For example, procedure "quick" shares stack frame of external procedure "main".

stack
if a procedure A calls another procedure B, the execution of A is suspended until B returns. If B in turn calls C, this is an ordered list of procedure or subroutine calls indicating which program called which other program, and which will return to which. This ordered list is called the "stack". In probe, a trace of the stack can be displayed by use of the stack request. The list is given in top-down fashion with the most recently called procedure listed first:

```
3       C
2       B
1       A
```

The numbers are level numbers.

stack frame
when a block is invoked (that is, a procedure is called or a begin block is entered), storage is allocated for its automatic variables. The area allocated is called a stack frame and logically corresponds to each entry in the stack.

static storage
a storage class for which space is allocated once per process, effectively at the time the procedure is first referenced. As a result, variables of this class always have a legitimate address and value. Regular FORTRAN variables, and those in a common block, have static storage. PL/1 variables must be explicitly declared.

statement map
a table in the symbol section of an object segment that relates locations in the text section (executable mode) to

source line numbers. This table is produced by a language translator when -table or -brief_table is specified.

support procedure
a system utility routine that provides runtime support for other procedures (e.g., the procedure that allocates storage as requested by a PL/1 allocate statement).

symbol table
a table in the symbol section of an object segment that contains information about the variables (symbols) used in the program. A symbol table is produced by a language translator when the -table control argument is specified.

SUMMARY OF REQUESTS

request	abbrev	function
.		cause probe to identify itself
..		escape to command processor
after	a	set break after a statement
args		print arguments to current procedure
before	b	set a break before a statement
call	cl	call an external procedure
continue	c	restart break
continue_to	ct	insert temporary break and continue
display	ds	display storage in selected format
execute	e	pass string to command processor
goto	g	transfer to a statement
halt	h	in break text, establish a probe level
help		print information about probe
if		execute probe requests if condition is true
input_switch	isw	read probe requests from switch
language	lng	set probe langauge
let	l	assign a value to a variable
list_builtins	lb	list all probe builtins
list_help	lh	list all topics
list_requests	lr	list all probe requests
modes	mode	control probes behavior
output_switch	osw	direct probe output to a switch
pause	pa	stop a program once
position	ps	set the source pointer and print source line
quit	q	return from current probe invocation
reset	r	delete breaks
source	sc	print source lines
stack	sk	trace the stack
status	st	list breakpoints
step	s	advance one statement and halt
symbol	sb	display attributes of a variable
use	u	set source pointer
value	v	display value of an expression
where	wh	display value of probe pointers
while	wl	execute commands while condition is true

process_dir (pd)

SYNTAX AS A COMMAND:

pd

SYNTAX AS AN ACTIVE FUNCTION:

[pd]

FUNCTION: returns the pathname of the process directory of the process in which it is invoked.

SYNTAX AS A COMMAND:

pf {program_names} {-control_args}

FUNCTION: a performance measuring tool that analyzes the time spent executing each source statement of a program, along with other parameters of interest, after the program is run.

ARGUMENTS:

program_names
are pathnames or reference names of programs to be analyzed. Any program_name that does not include "<" or ">" characters is assumed to be a reference name. They need not be specified if the -input_file control argument is used.

CONTROL ARGUMENTS:
Control arguments apply to all programs specified, and can be given in any order.

-print, -pr
prints the following information for each statement in the specified program(s):

1. Line Number.

2. Statement Number
   if more than one statement on the line.

3. Count
   the number of times the statement was executed.

4. Cost
   an approximation to the accumulated execution time for the statement. Equal to the number of instructions executed plus ten times the number of external operators called.

5. Stars (asterisks)
   an indication of the percentage of total cost (or time, for long_profile data) used in the statement. The number of stars is selected according to the table in the "List of Stars" section below.

6. Names of all external operators called by the statement.

For -long_profile (actual accumulated time) data, item 4 is changed to the following:

4a. Time
actual execution time for the statement in virtual CPU microseconds, including all time spent in any operators or subroutines invoked by the statement.

4b. Average Time
Time divided by Count (the average execution time for one execution of the statement).

4c. Page Faults
page faults incurred in executing the statement.

-no_header, -nhe
used with -print to suppress column headings.

-sort STR
used with -print to sort profile information into descending order of the specified field STR, which can be any of the fields in the "List of Fields" section below.

-first N, -ft N
used with -sort to print only the first N values.

-long, -lg
used with -print to include in the output information for statements that have never been executed.

-brief, -bf
used with -print to exclude from the output all information for statements that have never been executed. This is the default.

-list, -ls
creates a profile listing for all specified programs. The profile listing file is given a name consisting of the first program name with the language suffix replaced by the pfl suffix. It is placed in the working directory. The information described above for the -print control argument is placed in columns to the left of each source line in the profile listing.

-source_dir path, -scd path
used with -list when the source segments to be listed have been moved from the directories in which they were compiled. If -source_dir is specified, only the directory specified by pathname path is searched for source segments.

-line_length N, -ll N
used with -list to specify an output width of N characters. The default is 132.

-plot STR
plots a bar graph, on any supported graphics terminal, of the values of the specified field STR. STR can be any of the fields in the "List of Fields" section below. Use of this control argument requires that the site has installed the Multics Graphics System, and that the setup_graphics command has been executed. See the Multics Graphics System, Order No. AS40, for more information.

-from N, -fm N
used with -print or -plot to begin the output with the data for line number N. The default is 1.

-to N
used with -print or -plot to end the output with the data for line number N. The default is the line number of the last executable statement.

-max_points N, -mp N
used with -plot to specify the maximum number of points (line numbers) to be plotted (the graphics resolution). The default is 250. The Multics Graphics System is capable of plotting up to 1024 points.

-output_file path, -of path
causes the profile data for the specified program_names to be stored in the profile data file specified by path. The file is created if it does not already exist and is overwritten if it already exists. The pfd suffix is added to path if it is not already present. The profile data is stored in a format acceptable to the -input_file control argument. The format of pfd data files is described by the pl1 include file pfd_format.incl.pl1. The stored data is determined by the program_names specified, the -comment control argument and whether the compilation was done using the -profile or -long_profile options. The name a program was compiled with is saved in the profile data file. If program_name specifies a bound object segment, profile data about each component of the bound object segment is saved.

-comment STR, -com STR
used with the -output_file control argument to include STR with the stored profile data as a comment. This control argument can also be used with -plot. If STR is to include blanks or other characters recognized as special by the command processor, it should be enclosed in quotes. STR can be up to 128 characters long.

-input_file path, -if path
causes the profile data to be retrieved from the profile data file specified by path. Use of this control argument causes the current (internal static) profile data, if any, to be ignored. The pfd suffix is appended to path if it is not already present. If any program_names are specified, they select a subset of the stored data for analysis. If no program_names are specified, all data stored in the profile data file is used. This control argument is inconsistent with -output_file.

-reset, -rs
resets (zeros) all current (internal static) profile data for the named program(s). When -reset is specified, the resetting is done as the very last operation if -print, -list, -plot, or -output_file are also specified. This control argument is inconsistent with -input_file and -hardcore.

-hardcore, -hard
indicates that the specified programs are supervisor (hardcore) segments. The current (internal static) profile data for such programs is retrieved from the address space of the supervisor. Hardcore programs compiled with the -profile or -long_profile control arguments must be installed by generating a Multics System Tape and rebooting Multics. See System Programming Tools, Order No. AZ03, for a description of the generate_mst command. Note that the current (internal static) profile data for hardcore programs cannot be reset (zeroed).

-search_dir path, -srhd path
used with -hardcore to add path to an internal search list of hardcore object directories. Up to 8 directories can be specified. If no search list is specified, >ldd>hard>o is searched for copies of the specified program(s).

LIST OF STARS:
4 stars: 20% to 100%
3 stars: 10% to 20%
2 stars: 5% to 10%
1 star: 2.5% to 5%
no stars: 0% to 2.5%
one period: Statement was not executed.

NOTES: The program to be analyzed must be compiled using the -profile (-pf) control argument of the cobol, fortran and pl1 commands, or using the -long_profile (-lpf) control argument of the pl1 command. The -long_profile compiler control argument is used to acquire exact elapsed time statistics and is much more expensive to use than the -profile compiler control argument.

If none of the control arguments -print, -list, -plot, -output_file, or -reset is specified, -print is assumed.

When analyzing several runs of the same program(s) on various test cases, -reset should be specified. If -reset is not specified, the current (internal static) profile data is accumulated (added) for all runs.

If several identical control arguments are specified, only the last one is used, except for -search_dir, as explained above.

The current (internal static) data acquired by programs compiled with the -long_profile control argument is subject to small perturbations due to asynchronous events outside the control of the data acquisition mechanism. Therefore, -long_profile results are most reliable when obtained from long-executing programs or from multiple executions of the same program.

The execution time for -long_profile programs can be up to ten times as long as normal due to the overhead of acquiring CPU time and paging data from the supervisor. This overhead is subtracted from the current profile data before any further processing is done.

There are two forms of profile data, current and stored. Current data is in a form suitable for direct incrementing by the program(s) being analyzed and is stored using the pl1 internal static storage class. Current profile data (except for hardcore programs) can be reset by the -reset control argument. Stored profile data is permanent data as stored by the -output_file control argument.

Profile listing files (pfl) and profile data files (pfd) are automatically stored as multisegment files if they are too large to fit into a single segment. This feature allows very large bound object segments to be analyzed and very large source segments to be listed.

The time and page_faults fields are only available for sorting or plotting if the program was compiled with -long_profile. If the program was not compiled with -profile, the cost field is sorted or plotted instead.

Profile data generated from statements in include files are printed only if the -from or -to control arguments are not specified. Include file profile data cannot be plotted. Include files which generated profile data are listed after the main source program. If the -source_dir control argument is specified, include files are searched for first in the specified source directory, and then in the directory in which they were compiled.

EXAMPLES: The following command lines compile a PL/I program with -profile, execute the program once to acquire current profile data, and print the 5 most expensive statements.

```
! pl1 factorial -profile
   PL/I 24c
! factorial
   n          n!
   1          1
   2          2
   3          6
   4          24
   5          120
   6          720
   7          5040
   8          40320
   9          362880
   10         3628800
```

```
! profile factorial -sort cost -first 5

   Program: factorial
   LINE STMT   COUNT      COST STARS   OPERATORS
     20           45      1710 ****    call_int_other, return
     10           10       440 ***     call_int_this
                                          call_ext_out_desc
     18           55       220 **
     18           10       120 *       return
     11           10        50
   -------
   Totals:       144      2604
```

The following command line saves the current profile data in factorial.pfd.

```
! profile factorial -of factorial
```

The following command line creates a profile listing in factorial.pfl from the source segment factorial.pl1 and the profile data file factorial.pfd. The listing is prepared for a printer with only 50 columns.

```
! profile -if factorial -ls -ll 50
```

program_interrupt (pi)

SYNTAX AS A COMMAND:

pi

FUNCTION: allows the users of certain editors, subsystems, and other interactive programs to reenter those programs at known places after having interrupted the process by issuing a quit signal.

NOTES: The documentation of each individual program specifies whether or not it accommodates the program_interrupt feature and exactly what its behavior is following such an interrupt.

Generally speaking, when wanting to reenter a program known to accommodate the program_interrupt feature, the user issues the program_interrupt command (or pi) after returning to command level by issuing the quit signal. The interrupted program usually aborts what it was doing and resumes interaction with the user.

If a program_interrupt is mistakenly directed at a program not having this feature, the system default error handler prints a message and returns the user to command level. At this point, the system is still holding the work that was originally interrupted by the quit signal. The user can restart the program quit out of by invoking the start command. If the user does not desire to continue execution of the program, the release command should be issued to return to the command level prior to the command just "quit" out of. Refer to the descriptions of the release and start commands.

To make use of the program interrupt facility, a program or subsystem must establish a condition handler for the program_interrupt condition. When the user invokes the program_interrupt command, it signals the program_interrupt condition, and the handler established by the program or subsystem is invoked. For a discussion of conditions see "The Multics Condition Mechanism" and "List of System Conditions and Default Handlers" in the MPM Reference Guide.

program_interrupt (pi)

EXAMPLES: The edm command has a handler for the program_interrupt condition that stops whatever the editor is doing and looks for a request from the user's terminal. Thus, a user of edm who inadvertently types "p100" (to print 100 lines) can kill this printout by issuing a quit signal and then typing program_interrupt. The edm command responds by printing "Edit." and then waiting for editing requests.

SYNTAX AS A COMMAND:

pg {-control_arg} {command_line}

FUNCTION: executes a specified command line and prints information about how its execution is progressing in terms of CPU time, real time, and page faults.

CONTROL ARGUMENTS:
if present, progress performs only the function specified by that control argument. No command_line argument can follow except in the case of -brief (-bf). The control argument can be one of the following:

-off
suppresses the incremental messages (see "Notes on Output Messages" below) printed during execution of a command line previously initiated, but does not suppress the message printed when that command line is finished. This control argument can be used to suppress messages while debugging.

-on
restores the printing of incremental messages during execution of the command line.

-brief, -bf
permits only the message at completion of the command line to be printed. The command_line argument is used following this control argument.

-output_switch name, -os name
directs output from the progress command to be printed on the I/O switch named name. The default switch is user_i/o.

-cput N
prints incremental messages every N seconds of virtual CPU time. The default is -cput 10.

-realt N
prints incremental messages every N seconds of real time instead of virtual CPU time.

OPTIONAL ARGUMENTS:

command_line
is a character string made up by concatenating all the arguments to progress (excluding the first if it is a control argument) with blanks between them. The string is executed as a command line.

NOTES ON OUTPUT MESSAGES: After every 10 seconds of virtual CPU time (assuming the default triggering value is used), progress prints out a message of the form:

```
ct/rt = pt%, ci/ri = pi% (pfi)
```

where:

ct is the number of virtual CPU seconds used by the command line so far.

rt is the total real seconds used so far.

pt is the ratio of virtual to real time used by the command so far.

ci is the incremental virtual CPU time (since the last message).

ri is the incremental real time.

pi is ci expressed as a percentage of ri.

pfi is the number of page faults per second of virtual CPU time (since the last message).

When the command line finishes, progress prints the following message:

```
finished: ct/rt = pt% (pft)
```

where pft is the number of page faults per second of virtual CPU time for the execution of the entire command.

EXAMPLES: In the following example, the user wants to see how execution is progressing for the compilation of a PL/I source program (named newseg.pl1) using the -list control argument to the pl1 command.

```
! progress pl1 newseg -list
  PL/I
  10/30 = 33%, 10/30 = 33% (26)
  20/50 = 40%, 10/20 = 50% (17)
  30/123 = 24%, 10/73 = 13% (20)
  finished: 33/150 = 22% (22)
```

SYNTAX AS A COMMAND:

qx

FUNCTION: used to create and edit segments in Multics. The qedx editor cannot be called recursively. This description of the qedx editor summarizes the editing requests and addressing features provided by qedx.

NOTES: The invocation of qedx puts you in the editor in edit mode, where the editor waits for you to type a qedx request. To create a new segment, you might perform the following steps:

1. Invoke qedx and enter input mode by typing one of the input requests (e.g., append) as the first qedx request.

   a) Enter text lines into the buffer from the terminal.

   b) Leave input mode by typing the escape request sequence as the first characters of a new line.

2. Inspect the contents of the buffer and make any necessary corrections using edit or input requests.

3. Write the contents of the buffer into a new segment using the write request.

4. Exit from the editor using the quit request.

To edit an existing segment, you might perform the following steps:

1. Invoke qedx and read the segment into the buffer by giving a read request as the first qedx request.

2. Edit the contents of the buffer using edit and input requests as necessary. (The editor makes all changes on a copy of the segment, not on the original. Only when you issue a write request does the editor overwrite the original segment with the edited version.)

3. Using the write request, write the contents of the modified buffer either back into the original segment or, perhaps, into a segment of a different name.

4. Exit from the editor using the quit request.

You can create and edit any number of segments with a single invocation of the editor as long as the contents of the buffer are deleted before work is started on each new segment.

Complete tutorial information on qedx is available in the qedx Text Editor Users' Guide, Order No. CG40.

NOTES ON ADDRESSING: Most editing requests are preceded by an address specifying the line or lines in the buffer on which the request is to operate. Lines in the buffer can be addressed by absolute line number; relative line number, i.e., relative to the "current" line (+2 means the line that is two lines ahead of the current line, -2 means the line that is two lines behind); and context (locate the line containing /any string between these slashes/). Current line is denoted by period (.); last line of buffer, by dollar sign ($).

NOTES ON REGULAR EXPRESSIONS: The following characters have specialized meanings when used in a regular expression. A regular expression is the character string between delimiters, such as in a substitute request, or a search string. You can reinvoke the last used regular expression by giving a null regular expression (//).

* signifies any number (or none) of the preceding character.

^ when used as the first character of a regular expression, signifies the (imaginary) character preceding the first character on a line.

$ when used as the last character of a regular expression, signifies the (imaginary) character following the last character on a line.

. matches any character on a line.

LIST OF ESCAPE SEQUENCE REQUESTS:

\f exits from input mode and terminates the input request; returns the user in edit mode. It is used constantly when editing a document, and is the key to understanding the difference between input mode and edit mode.

\c suppresses the meaning of the escape sequence or special character following it.

\b(X) redirects editor stream to read subsequent input from buffer X.

\r temporarily redirects the input stream to read a single line from your terminal.

NOTES ON REQUESTS: In the list given below, editor requests are divided into four categories: input requests, basic edit requests, extended edit requests, and buffer requests. The input requests and basic edit requests are sufficient to allow a user to create and edit segments. The extended requests give the user the ability to execute commands in the Multics system without leaving the editor and also to effect global changes. Because the extended requests are, in general, more difficult to use properly, they should be learned only after mastering the input and basic edit requests. The buffer requests require a knowledge of auxiliary buffers. (Since the nothing and comment requests are generally used in macros, they are included with the buffer requests.) The buffer requests, used with any of the other requests, and special escape sequences allow the user to make qedx function as an interpretive programming language through the use of macros.

The character given in parentheses is the actual character used to invoke the request in qedx and does not always bear a relation to the name of the request. The second part of each entry shows the format, default in parentheses, and brief description. For the value of ADR, see "Notes on Addressing" above; for the value of regexp, see "Notes on Regular Expressions" above.

LIST OF INPUT REQUESTS: These requests enter input mode and must be terminated with \f.

append (a)
Enter input mode, append lines typed from the terminal after a specified line.

ADRa (.a) append lines after specified line.

change (c)
Enter input mode, replace the specified line or lines with lines typed from the terminal.

ADR1,ADR2c (.,.c) change existing line(s); delete and replace.

insert (i)
Enter input mode, insert lines typed from the terminal before a specified line.

ADRi (.i) insert lines before the specified line.

LIST OF BASIC EDIT REQUESTS:

delete (d)
Delete specified line or lines from the buffer.

ADR1,ADR2d (.,.d) delete line(s).

print (p)
Print specified line or lines on the terminal; special case print needs address only.

ADR1,ADR2p (.,.p) print line(s).

print line number (=)
Print line number of specified line.

ADR= (.=) print line number.

quit (q)
Exit from the editor.

q exit from qedx editor.

read (r)
Read specified segment into the buffer.

ADRr path ($r path) append contents of path after specified line.

substitute (s)
Replace specific character strings in specified line or lines.

ADR1,ADR2s/regexp/string/ (.,.s/regexp/string/) substitute every string matching regexp in the line(s) with string. If string contains &, & is replaced by the characters which matched regexp. First character after s is delimiter; it can be any character not in either regexp or string.

Strings matching regexp do not overlap and the result of substitution is not rescanned.

write (w)
Write current buffer into specified segment.

ADR1,ADR2w {path} (1,$w path) write lines into segment named path. If path omitted, a default pathname used if possible, otherwise error message printed.

LIST OF EXTENDED EDIT REQUESTS:

execute (e)
passes the remainder of a request line to the Multics command processor (i.e., escape to execute other Multics commands) without leaving the qedx editor.

e <command line>

global (g)
Print, delete, or print line number of all addressed lines that contain a match for a specified character string.

ADR1,ADR2gX/regexp/ (1,$gX/regexp/) perform operation on lines that contain a match for regexp; X must be d for delete, p for print, or = for print line numbers.

exclude (v)
Print, delete, or print line number of all addressed lines that do not contain a specified character string.

ADR1,ADR2vX/regexp/ (1,$vX/regexp/) perform operation on lines that do not contain a match for regexp; X must be d for delete, p for print, or = for print line numbers.

LIST OF BUFFER REQUESTS:

buffer (b)
switches to specified buffer (i.e., switches all subsequent editor operations to the specified buffer).

b(X) go to buffer named X; destroy old contents of buffer X.

move (m)
moves the specified line or lines into the specified buffer.

ADR1,ADR2m(X) (.,.m(X)) move line(s) from current buffer into buffer named X; destroy old contents of buffer X.

status (x)
prints a summary status of all buffers currently in use.

x gives the current status of all buffers in use.

nothing (n)
does not perform a task (used to address a line with no other action).

ADRn (.n) set value of "." to line addressed.

comment (
ignores the remainder of this request line.

ADR" (.") ignores rest of line; used for comments.

NOTES ON SPACING: The following rules govern the use of spaces in editor requests.

1. Spaces are taken as literal text when appearing inside of regular expressions. Thus, /the n/ is not the same as /then/.

2. Spaces cannot appear in numbers, e.g., if 13 is written as 1 3, it is interpreted as 1+3 or 4.

3. Spaces within addresses except as indicated above are ignored.

4. The treatment of spaces in the body of an editor request depends on the nature of the request.

RESPONSES FROM THE EDITOR: In general, the editor does not respond with output on the terminal unless explicitly requested to do so (e.g., with a print or print line number request). The editor does not comment when you enter or exit from the editor or change to and from input and edit modes. The use of frequent print requests is recommended for new users of the qedx editor. If you inadvertently request a large amount of terminal output from the editor and wish to abort the output without abandoning all previous editing, you can issue the quit signal (by pressing the proper key on your terminal, e.g., BRK, ATTN, INTERRUPT), and, after the quit response, you can reenter the editor by invoking the program_interrupt (pi) command (fully described in the MPM

qedx (qx)

Commands). This action causes the editor to abandon its printout, but leaves the value of "." as if the printout had gone to completion.

If an error is encountered by the editor, an error message is printed on your terminal and any editor requests already input (i.e., read ahead from the terminal) are discarded.

If you exit from qedx by issuing the quit signal, and subsequently invoke qedx in the same process, the message "qedx: Pending work in previous invocation will be lost if you proceed; do you wish to proceed?" is printed on the terminal. You must type a "yes" or "no" answer.

NOTES ON MACRO USAGE: You can place elaborate editor request sequences (called macros) into auxiliary buffers and then use the editor as an interpretive language. This use of qedx requires a fairly detailed understanding of the editor. To invoke a qedx macro from command level, you merely place your macro in a segment that has the letters qedx as the last component of its name, then type:

```
! qedx path optional_args
```

where:

1. path
   specifies the pathname of a segment from which the editor is to take its initial instructions. Such a set of instructions is commonly referred to as a macro. The editor automatically concatenates the suffix qedx to path to obtain the complete pathname of the segment containing the qedx instructions.

2. optional_args
   are optional arguments that are appended, each as a separate line, to the buffer named args (the first optional argument becomes the first line in the buffer and the last optional argument becomes the last line). Arguments are used in conjunction with a macro specified by the path argument.

   The editor executes the qedx requests contained in the specified segment and then waits for you to type further requests. If path is omitted, the editor waits for you to type a qedx request.

qedx (qx)

NOTES ON I/O SWITCHES: While most users interact with the qedx editor through a terminal, the editor is designed to accept input through the user_input I/O switch and transmit output through the user_output I/O switch. These switches can be controlled (using the iox_ subroutine described in the MPM Subroutines) to interface with other devices/files in addition to the user's terminal. For convenience, the qedx editor description assumes that the user's input/output device is a terminal.

SYNTAX AS AN ACTIVE FUNCTION:

[query arg]

FUNCTION: asks the user a question and returns the value true if the user's answer to the question is "yes" or false if the user's answer is "no"; if the user's answer is anything else, the query active function prints a message asking for a "yes" or "no" answer.

ARGUMENTS:

arg
is the question to be asked. If the question contains spaces or other command language characters, it must be enclosed in quotes.

NOTES: The format_line active function (see format_line in this manual), can be used to insert other active function values into the question.

EXAMPLES: The following lines from an exec_com segment allow the user to control the continued execution of the exec_com.

```
&if [query "Do you wish to continue?   "]
&then
&else &quit
```

SYNTAX AS A COMMAND:

```
quotient numA numB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[quotient numA numB]
```

FUNCTION: returns the result of numA divided by numB.

NOTES: See the description of divide, which returns only the integer portion of quotient, in this manual.

EXAMPLES: The following interaction illustrates the quotient active function.

```
! string [quotient 5 4]
  1.25
! string [quotient 1 3]
  0.33333333333333333333333333333333333333333333333333333333333333
! string [quotient 5 2]
  2.5
```

SYNTAX AS A COMMAND:

rdm {input_spec} {-control_args}

FUNCTION: provides a facility for examining and manipulating messages sent by the send_mail and mail commands.

ARGUMENTS:

input_spec
tells read_mail where to read messages. The source can be a mailbox specified by pathname or user name.

If no input_spec appears in the command line, the user's default mailbox (>udd>Project_id>Person_id>Person_id.mbx) is read. Since only one source can be read at a time, it is an error to include more than one input_spec in the read_mail command line. The permissible forms for input_spec are:

-log
reads from the user's logbox instead of from the user's mailbox. The logbox is the mailbox Person_id.sv.mbx in the home directory.

-mailbox path, -mbx path
reads from the mailbox specified by path instead of from the user's default mailbox. The mbx suffix is added to path if it is not present.

-save path, -sv path
reads from the mailbox path instead of from the user's default mailbox. The suffix .sv.mbx is added to path if it is not present. This control argument is equivalent to "-mailbox path.sv".

-user Person_id.Project_id
reads from the specified user's mailbox. This control argument is useful if a segment of the name Person_id.Project_id exists in the working directory.

STR
is any argument that does not begin with a minus sign (-). If it contains either of the characters > or <, it is interpreted as "-mailbox STR". The mbx suffix is added if it is not present. If STR does not contain > or <, it is interpreted as "-user STR".

CONTROL ARGUMENTS:

-brief, -bf
shortens informative messages printed by read_mail.

-header, -he
prints message headers by the print (pr) request to read_mail. (See the section on headers in the description of the send_mail command.) This is the default.

-interactive_messages, -im
operates on interactive messages from send_message as well as mail messages from send_mail. If this control argument is not given, interactive messages are ignored.

-list, -ls
prints a summary of the messages in the mailbox before entering the request loop.

-long, -lg
prints the full text of informative messages of the read_mail command, as opposed to -brief. This is the default.

-no_header, -nhe
omits headers when printing messages via the print (pr) request. Instead, a brief header line is printed giving the message number, the sender, the subject, and the date-time sent, of the form:

```
#3 (71 lines) 07/22/78 12:20 Mailed By: Jones.Publications
Subject: Manual Ready
```

-no_interactive_messages, -nim
operates only on send_mail messages, not on interactive messages sent by send_message. This is the default.

-no_list, -nls
does not print a summary of messages before entering the request loop. This is the default.

-no_print, -npr
does not print messages before entering the request loop. This is the default.

-no_prompt
does not prompt for read_mail requests when inside the requests loop. This control argument is equivalent to -prompt "". The default prompt is "read_mail(N):", where N is the recursion level if greater than one.

-own
operates only on the user's own messages instead of on all the messages. This control argument can be useful when examining another user's mailbox.

-print, -pr
prints the messages in the mailbox before entering the request loop.

-prompt STR
changes the prompt for read_mail request lines to STR. If STR is "", the user is not prompted.

-quit
exits after performing any operations specified by control arguments. The default is to enter the request loop.

-request STR, -rq STR
specifies an initial request line to be executed by read_mail before entering the request loop. Thus, the command line:

```
read_mail -rq "print last;quit" -brief
```

prints the last message in the user's mailbox and returns to command level.

NOTES ON GENERAL USE: A user whose identification is Person_id.Project_id has the default mailbox:

```
>udd>Project_id>Person_id>Person_id.mbx
```

To read the contents of this mailbox, the user can perform the following steps:

1) Invoke read_mail with no arguments.

2) Issue the "list" request to obtain a summary of the messages. This summary contains one line for each message, for example:

```
1 (11) 07/23/78 18:03 Jones.Publications subject
```

giving the relative number of the message, the number of lines in parentheses, the date-time sent, the sender's name, and as much of the subject as will fit on the line, if a subject was specified to the send_mail command.

read_mail (rdm)

3) Read any selected message n by issuing the request "print n" or read all the messages by saying "print all".

4) Delete any selected message n by issuing the request "delete n" or delete the entire contents of the mailbox by saying "delete all".

5) Save any selected message n in a specified mailbox by issuing the "save n PATH" request, or say "log n" to save message n in the default logbox:

   >udd>Project_id>Person_id>Person_id.sv.mbx

6) Issue the "quit" request to exit read_mail.

The user can read other mailboxes by invoking read_mail with arguments. The command line:

```
rdm >udd>Publications>Jones>Jones
```

reads from the mailbox with the specified pathname. Another way to read this mailbox is to type:

```
rdm Jones.Publications
```

which constructs the identical pathname from the specified Person_id.Project_id pair.

The command line "read_mail -log" reads from the default logbox in which the "log" request saves messages.

EXAMPLES: A typical dialogue with read_mail follows. The user's input lines are indicated by an ! character.

```
! read_mail
     You have 4 messages.

     read_mail:  ! list

   Msg#  Lines  Date      Time   From                 Subject
      1    (3)  07/17/78 13:48  Jones.Publications   new<MORE>
      2    (1)  07/17/78 20:18  Doe.Software         Whe<MORE>
      3    (2)  07/18/78 09:23  Smith.Maintenance
      4    (1)  07/18/78 11:51  Brown.Hardware       Don<MORE>
```

read_mail (rdm)

```
read_mail:  !  print

#1 (3 lines) 07/17/78 13:48  Mailed By:  Jones.Publications
Date:  17 July 1978 13:48 est
From:  Jones.Publications
Subject:  new manuals

The two new manuals you asked about have arrived in the
warehouse.  Can I have your order?
----1----

read_mail:  !  delete; print all

#2 (1 line)  07/17/78 20:18  Mailed By:  Doe.Software
Date:  17 July 1978 20:18 est
From:  Doe.Software

Where's the new read_mail command description?
----2----

#3 (2 lines) 07/18/78 09:23  Mailed By:  Smith.Maintenance
Date:  17 July 1978 09:23 est
From:  Smith.Maintenance

All index references for Chapter 14 should be Chapter 13.
TR 2187 forwarded to developers for study.
----3----

#4 (1 line)  07/18/78 11:51  Mailed By:  Brown.Hardware
Date:  17 July 1978 11:51 est
From:  Brown.Hardware

Don't be alarmed.  It's in the author_maintained library.
----4----

read_mail:  !  log 3; delete all

All messages have been deleted.

read_mail:  !  quit

r 1034 1.342 4.198 153
```

read_mail (rdm)

In this example, the user invokes read_mail, gets a summary listing of the mailbox contents, prints all the messages, logs the third one, and deletes them all. There was no subject given in the summary list for the third message because one was not given at the time it was sent. If mail has been sent by the mail command, the first 12 characters of the message are printed as shown in messages 2 and 4 in the summary list.

DELETION

Messages deleted by the "delete" request remain in the mailbox until the "quit" request is issued to exit read_mail. In the meantime, deleted messages can be un-deleted by issuing the "retrieve" request. If messages 2 and 4 were deleted by mistake, for example, the request "retrieve 2 4" causes them to reappear.

REQUEST LINE SYNTAX

A request line beginning with ".." is treated as a special escape used to pass command lines directly to the standard command processor.

Other request lines have identical syntax to Multics command lines. Arguments containing spaces or other command language characters must be quoted. Iteration is specified by means of parentheses, for example:

```
(log delete) 3
```

Semicolon is used to separate multiple requests on a line, for example:

```
delete 4;print all
```

Each request accepts a particular set of arguments. These are described in detail in an alphabetical list of requests at the end of this command description.

read_mail (rdm)

NOTES ON REQUEST FUNCTIONS:

Brackets in a request line invoke read_mail request an internal repertoire. Request functions are listed along with requests at the end of this command description.

HELP

The "help" request describes how to use read_mail. To see how to use a particular request, type:

help REQUEST_NAME

NOTES ON REQUESTS:

In the following list, read_mail requests are divided into three categories. The first, simple requests, includes the ones mentioned so far. The second, copying requests, includes the save request and others used to move messages around. The third, advanced requests, lists several advanced requests.

Most requests have short names that can be used instead to save typing. A more complete description of each request, including its calling sequence, appears at the end of this section.

LIST OF SIMPLE REQUESTS:

?
prints a summary of the available read_mail requests.

.
identifies the current state of read_mail.

delete (dl)
deletes specified messages from the mailbox being read.

help
prints information on how to use read_mail.

list (ls)
prints a summary of the specified messages.

read_mail (rdm)

print (pr)
   prints the specified messages.

quit (q)
   exits from read_mail.

retrieve (rt)
   retrieves deleted messages.

LIST OF COPYING REQUESTS:

append
   copies messages to the end of an existing ASCII file.

copy (cp)
   copies messages into another mailbox without changing them.

forward (fwd)
   copies messages to another mailbox, adding "Redistributed" header fields.

log
   copies messages to the default logbox, adding original sender fields to the header.

preface
   copies messages to the beginning of an existing ASCII file.

save (sv)
   copies messages to a specified savebox, adding original sender fields to the header.

write (w)
   copies messages to the end of an old or new ASCII file.

ADVANCED REQUESTS

execute (e)
   executes a Multics command line after expanding read_mail request functions in the line.

reply (rp)
   sends a reply to the senders of specified messages.

..
   passes the command line directly to the standard command processor.

read_mail (rdm)

NOTES ON REQUEST FUNCTIONS:

These requests are used in active strings. They return strings consisting of one or more message numbers separated by spaces, except for the mailbox request function, which returns the absolute pathname of the mailbox being read.

current
first
last
next
previous
all
mailbox, mbx

NOTES ON MESSAGE SPECIFIERS: Message specifiers refer to messages in the mailbox. They are composed of message numbers, keywords, the arithmetic operators + and -, and qedx-type regular expressions for string matching. Ranges are composed of two expressions separated by colon (:), for example:

6:last-3

Message numbers are integers. These are assigned by read_mail when listing or printing the messages in the mailbox. Messages, including deleted messages, keep their numbers during an invocation of read_mail.

The available keywords and their short forms are:

first, f
last, l
previous, p
next, n
current, c
all, a

The first four are used like message numbers, for example:

last-1

The "current" message is initialized to 1 and changed by various requests. The "all" keyword denotes the range of all messages, and is equivalent to first:last.

read_mail (rdm)

Simple regular expressions are character strings enclosed in slashes (/). Slashes inside the match string must be preceded by ".". More complicated expressions can be built from simple ones, the connector "&" for logical AND, and the connector "|" for logical OR. Any of these expressions can be preceded by a keyword, for example:

```
last/artificial/&/intelligence/
```

which specifies the last message containing both of the strings "artificial" and "intelligence".

Because of the syntax of the request language, regular expressions containing special characters such as quote, space, and parenthesis must be enclosed in quotes.

Keywords can be used as prefixes for regular expressions. As prefixes, they mean the first, last, previous, or next message matching the given regular expression. The message specifier current/STR/ is undefined if the current message does not match STR. If a regular expression is not prefixed with a keyword, the default keyword is "all".

Normally, message specifiers refer to messages that have not been deleted. The "all" keyword refers to all undeleted messages, and "first" refers to the first undeleted message. If the -all control argument is specified to a request, however, deleted messages are included in the ranges.

If a range is specified and -all is not specified, there must be at least one message within the range that has not been deleted. For instance, if the mailbox originally had 20 messages in it, and 10 and 12 are the only ones left, it is perfectly valid to say "print 4:11". This request prints only message number 10.

The values used for "last", "first", "next", and "previous" also change depending on whether -all is specified. If -all is not specified, they refer only to existing messages. Therefore, in the example above, the request line "print last-4:1" is the same as "print 10 12" and the request line "print last-4:1 -all" is the same as "print 16:20 -all". The last two request lines print deleted messages.

Some examples of message specifiers are:

```
1          message number 1
1:3        messages 1 through 3
/foo/      all messages containing the string "foo"
last-3     the third from last message
1-3:1      the last four messages
next+4     the message five after the current one
p-2        three messages previous
c:c+4      the current message and the next four
c+1:1      the next through last messages
/a/|/b/    all messages containing either "a" or "b"
"1/ it /" the last message containing " it "
```

LIST OF REQUESTS:

The syntax of available read_mail requests are:

```
?
.
..
insert
copy {SPECS} path, cp {SPECS} path
current {-control_arg}
delete {SPECS}, dl {SPECS}
execute STR, e STR
first {-control_arg}
forward SPEC ADDRESSES, fwd SPEC ADDRESSES
help {STR}
last {-control_arg}
list {SPECS}, ls {SPECS}
log {-control_arg} {SPECS}
next {-control_arg}
preface {SPECS} path
previous {-control_arg}
print {-control_arg} {SPECS}, pr {-control_arg} {SPECS}
quit {-control_args}, q {-control_args}
reply {SPECS} {-control_args}, rp {SPECS} {-control_args}
retrieve {SPECS}, rt {SPECS}
save {-control_arg} {SPECS} path,
   sv {-control_arg} {SPECS} path
write {-control_args} {SPECS} path, w {SPECS} path
```

LIST OF REQUEST DESCRIPTIONS:

The various read_mail requests are described below. For a description of headers and header fields, see the section entitled Headers under the send_mail command in this document.

?
prints a summary of the available read_mail requests.

.
prints a line identifying read_mail and giving the pathname of the mailbox being read, the message count, and the current message number, as in:

```
read_mail 4.3:  Message #7 of 11, 0 deleted   >udd>X>x>lies.mbx
```

The string "read_mail 4.3" indicates the version number of read_mail. If the recursion level is greater than one, the identifying string is followed by a number in parentheses, for example:

```
read_mail 4.3 (2):
```

..
passes the rest of command line directly to the standard command processor, without processing by the read_mail request processor. The ".." string must be the first two characters of the request line.

append {SPECS} path
appends the specified messages complete with header to an existing ASCII file. The suffix mail is added to path if it is not present. If the file does not already exist, the user is asked whether to create it.

copy {SPECS} path, cp {SPECS} path
copies the specified messages verbatim into the mailbox designated by path. The mbx suffix is added to path if it is not present. Unlike the save and log requests, this request does not add Date and From header fields if they are missing.

current {-control_arg}
returns the number of the current message, or 0 if the current message has been deleted. If -all is specified, it returns the number of the current message whether or not that message has been deleted, or 0 if there are no messages.

delete {SPECS}, dl {SPECS}
deletes the specified messages. If no messages are specified, the current one is deleted. Deleted messages can be retrieved before exiting read_mail by using the retrieve (rt) request.

execute STR, e STR
passes the concatenation of STRs with intervening spaces to the Multics command processor. This request is different from ".." because it is first parsed as a read_mail request line. The read_mail request interpreter expands read_mail request

functions, strips quotes, and performs iteration before the line is passed on to the command processor. Therefore, the request:

```
e ioa_ [current]
```

prints the number of the current message, whereas:

```
..ioa_ [current]
```

produces the error message "Segment current not found." The ".." escape should normally be used to execute Multics command lines from with read_mail.

The execute request function can be used to invoke a Multics active function from within read_mail. For example:

```
save [execute date]
```

saves the current message in a savebox whose name is the current date.

first {-control_arg}
returns the number of the first message that has not been deleted, or 0 if all messages have been deleted. If -all is specified, it returns the number of the first message whether or not that message has been deleted, or 0 if there are no messages.

forward SPEC ADDRESSES, fwd SPEC ADDRESSES
forwards the messages indicated by the single message specifier to ADDRESSES. This request adds three fields to the header to record where the message came from: Redistributed-Date, Redistributed-By, and Redistributed-To. Iteration is needed to give more than one message specifier.

help {STR}
prints information about the read_mail command. If specified, STR is the name of a read_mail request or one of the topics "requests", "control_args", and "changes". If STR is "*", the available read_mail topics are listed. If STR is not specified, introductory information on the use of read_mail is printed followed by a list of topics.

last {-control_arg}
returns the number of the last message that has not been deleted, or 0 if all messages have been deleted. If -all is specified, it returns the number of the last message whether or not that message has been deleted, or 0 if there are no messages.

list {SPECS}, ls {SPECS}
prints a summary line for each of the specified messages, for example:

```
3  (52)   07/10/78 14:20  Jones.Publications     subject
```

The list request function returns a list of the numbers of the specified messages separated by spaces.

log {-control_arg} {SPECS}
saves the specified messages in the user's logbox, the mailbox named Person_id.sv.mbx in the home directory. Date and From header fields are added to those messages that do not have them. If the control argument -delete (-dl) is specified, the messages are deleted after they are logged. If no messages are specified, the current one is logged.

next {-control_arg}
returns the number of the next message that has not been deleted, or 0 if all messages have been deleted. If -all is specified, it returns the number of the next message whether or not that message has been deleted, or 0 if there is no next message.

preface {SPECS} path
operates the same as append, but inserts messages at the beginning of the ASCII file.

previous {-control_arg}
returns the number of the last previous message that has not been deleted, or 0 if all messages have been deleted. If -all is specified, it returns the number of the previous message whether or not that message has been deleted, or 0 if there is no previous message.

print {-control_arg} {SPECS}, pr {-control_arg} {SPECS}
prints the specified messages. If the -header (-he) control argument is specified, both the header and the text are printed. This is the default. If -no_header (-nhe) is specified, only the text is printed, preceded by a summary line and the contents (if the fields are present) of the From and Subject fields. The printing format is described at the beginning of this section.

quit {-control_args}, q {-control_args}
exits the request loop and the read_mail command. Control arguments can be -no_modify or -nm to prevent any modifications made to the mailbox during the invocation of read_mail, -force or -fc to suppress a question printed if new messages have arrived since read_mail was invoked.

reply {SPECS} {-control_args}, rp {SPECS} {-control_args}
allows the user to reply to the specified messages. A header is constructed for each reply, naming the sender of each message as the primary recipient of the reply and the recipients of each message as the secondary recipients of the reply. The text of all the replies is the same. This request prompts for "Message:", accepts a message text ending with a ".", and then enters the send_mail request loop. When the quit request is issued to send_mail, the user is returned to read_mail.

The send_mail control arguments that can be used by the reply request can be chosen from the following:

-acknowledge, -ack
-brief, -bf
-fill, -fl
-input_file path, -if path
-line_length N, -ll N
-log
-long
-message-id, -mid
-no_acknowledge, -nack
-no_fill, -nfi
-no_log
-no_message_id, -nmid
-no_request_loop, -nrql
-no_subject, -nsj
-request_loop, rql
-save path, -sv path
-subject STR, -sj STR
-terminal_input, -ti

retrieve {SPECS}, rt {SPECS}
causes the specified messages, if deleted, to be un-deleted. This action is allowed until the request loop is exited. When the user exits read_mail, all messages deleted by the delete (dl) request are actually deleted from the mailbox and can no longer be retrieved.

save {-control_arg} {SPECS} path, sv {-control_arg} {SPECS}
saves the specified messages in the mailbox designated by path. The .sv.mbx suffix is added to path if it is not present. If the savebox does not exist, the user is asked whether to create it. Date and From fields are automatically added to any messages that do not have them. These fields preserve information about the origin of the message. If the control argument -delete (-dl) is specified, delete the messages after saving them. If no messages are specified, the

read_mail (rdm)

current one is saved.

write {-control_args} {SPECS} path, w {-control_args} {SPECS} path
appends the specified messages to the ASCII file designated by path. The mail suffix is added to path if it is not present. If no messages are specified, the current one is written. Date and From fields are added to any messages that do not have them. The -extend and -truncate (-tc) control arguments accepted by the file_output command can be used here.

NOTES: Requests, which handle messages on an individual basis, allow the user to print messages, delete messages, forward messages, save messages either in a segment suitable for printing offline or in a mailbox, and print summary information for specified messages. In its standard mode of operation, read_mail enters a request loop where it reads requests from the terminal until the quit request causes it to exit. By default, requests are prompted by the string "read_mail: ". See also the print_mail and send_mail commands in this manual. See "Notes on Extended Access" in the print_mail description for an explanation of mailbox access.

ready (rdy)

SYNTAX AS A COMMAND:

rdy

FUNCTION: types out an up-to-date ready message whose format is optionally set by the general_ready command. The default ready message if general_ready is not used gives the time of day and the amount of CPU time and page faults used since the last ready message was typed. If the user is not at the first command level, i.e., if some computation has been suspended and the stack frames involved not released, the default ready message also contains the number of the current command level.

NOTES: See the descriptions of ready_on, ready_off, and general_ready in this manual.

EXAMPLES:

r 9:47 3.61 29

r 15:03 .47 12 Level 2

ready_off (rdf)

SYNTAX AS A COMMAND:

rdf

FUNCTION: turns off the ready message typed on the terminal after the processing of each command line. Automatic typing of the message is suspended until a ready_on command is given.

NOTES: See the descriptions of ready, ready_on, and general_ready in this manual.

ready_on (rdn)

SYNTAX AS A COMMAND:

rdn

FUNCTION: causes a ready message to be automatically typed on the terminal after each command line has been processed.

NOTES: Since automatic printing of the ready message is in effect until ready_off is invoked, the ready_on command is generally used only to "cancel" the ready_off command.

See the descriptions of ready, ready_off, and general_ready in this manual.

SYNTAX AS A COMMAND:

rl {-control_arg}

FUNCTION: releases the stack history that was automatically preserved after a quit signal or unclaimed signal. That is, the Multics stack is returned to a point immediately prior to the stack frame of the command that was being executed when the most recent quit signal or unclaimed signal occurred.

CONTROL ARGUMENTS:

-all, -a
releases the stack history preserved (and not already released) after all previous quit and/or unclaimed signals rather than after only the most recent quit or unclaimed signal.

SYNTAX AS A COMMAND:

rn {-control_arg} path1 name1 {... {-control_arg} pathn namen}

FUNCTION: replaces a specified segment, multisegment file, directory, or link name by a specified new name, without affecting any other names the entry might have.

ARGUMENTS:

pathi
is the pathname of a segment, multisegment file, directory, or link. The star convention is allowed.

namei
specifies the new name that replaces the storage system entryname portion of pathi. The equal convention is allowed.

CONTROL ARGUMENTS:

-name, -nm
indicates that the path argument that follows it is an entryname containing special command system symbols (e.g., < or *). This control argument allows the user to rename strangely named segments. This control argument disables the star convention in the argument that it precedes.

ACCESS REQUIRED: The user requires modify permission on the containing directory.

NOTES: Since two entries in a directory cannot have the same entryname, special action is taken by this command if namei already exists in the directory specified by pathi. If the entry having the entryname namei has an alternate name, entryname namei is removed and the user is informed of this action; the renaming operation then takes place. If the entry having the entryname namei has only one name, the entry must be deleted in order to remove the name. The user is asked if the deletion should be done; if the user answers "no", the renaming operation does not take place.

EXAMPLES:

The command line:

```
! rename alpha beta >sample_dir>gamma delta
```

renames alpha, in the user's working directory, to beta and renames gamma, in the directory >sample_dir, to delta.

The command line:

```
! rename -name *stuff junk
```

renames the segment *stuff, in the working directory, to junk.

SYNTAX AS A COMMAND:

rq

FUNCTION: repeats the last query (by the command_query_ subroutine, described in the MPM Subroutines) if it has not yet been answered.

NOTES: This command is useful for reinterpreting questions (asked by other commands) that are garbled.

If no question has been asked, or if the latest question was answered, the error message "No pending query." is printed.

The repeat_query command does not completely restore the environment in effect at the time of the original query. For example, nonstandard attachments of I/O switches are not restored.

EXAMPLES: Suppose that the system starts to print a question while the user is typing. The query looks like:

```
E@foo.pl1?
```

The user signals QUIT and invokes the repeat_query command. The system prints:

```
Do you want to delete the old segment foo.pl1?
```

The user answers and continues.

Alternatively, the user can use the ".." command escape to issue the repeat_query command:

```
E@foo.pl1?  ! ..rq
```

The system responds with:

```
Do you want to delete the old segment foo.pl1?
```

The user then answers and continues.

repeat_query (rq)

Another use is to return to a query after interrupting a command line issued within the query:

```
   Do you want to delete the old segment foo.pl1?

!  ..print foo.pl1 1
   foo: proc;
   (user signals QUIT)

!  rq

   Do you want to delete the old segment foo.pl1?  ! yes
```

SYNTAX AS A COMMAND:

re {-control_args}

FUNCTION: causes the system condition handler to print its message for a condition that has already been handled and for which stack history is preserved.

CONTROL ARGUMENTS:

-depth i, -dh i
indicates which instance of saved fault information is to be used for the message (the most recent instance is depth 1). This control argument can appear only once per command line. The default is 1.

-all, -a
prints messages corresponding to all existing sets of condition information.

-brief, -bf
prints the short form of the message.

-long, -lg
prints the long form of the message.

NOTES: If no control argument is specified, the default action results in the selection of slightly less extensive condition information than that printed by the -long control argument.

The message mode options for this command have no effect on the operation of the default error handler as such.

EXAMPLES: The following example illustrates some ways the reprint_error command can be used.

```
! change_error_mode -brief

! simf

  Error: seg_fault_error

! reprint_error -long
  depth 1:
```

reprint_error (re)

```
      Error: Segment-fault error by command_processor_|404
      (>system_library_1>bound_command_loop_|3046)
      referencing >udd>m>Smith>simf|3
      (offset is relative to base of segment)
      Incorrect access on entry.

   !  set_acl simf re

   !  simf

      Error: simfault_000001

      call_den_test_gate$ob

      Error: out_of_bounds while in ring 1

   !  reprint_error -all -brief
      depth 1:
      Error: out_of_bounds while in ring 1
      depth 2:
      Error: simfault_000001
      depth 3:
      Error: seg_fault_error

   !  reprint_error -long -depth 2
      depth 2:
      Error: Attempt by >udd>m>Smith>simf$simf|13
      to reference through null pointer
      (simfault_000001 condition)

   !  reprint_error -long -depth 1
      depth 1:
      Error while processing in ring 1:
      out_of_bounds at deh_test1|103
      (>system_library_tools>bound_deh_test_|347
      referencing stack_1|720703 (in process dir)
      Attempt to access beyond end of segment.
      Entry into lower ring was by
      call_den_test_gate$ob|115
      (system_library_tools>bound_deh_test_|115)
      referencing den_test_gate_$gob
```

resolve_linkage_error (rle)

SYNTAX AS A COMMAND:

rle virtual_entry

FUNCTION: is invoked to satisfy the linkage fault after a process encounters a linkage error.

ARGUMENTS:

virtual_entry
is a virtual entry specifier. For an explanation of virtual entries, see the description of the cv_entry_ subroutine in the MPM Subsystem Writers' Guide.

NOTES: The program locates the virtual entry specified as an argument and patches the linkage information of the process so that when the start command is issued the process continues as if the original linkage fault had located the specified virtual entry.

EXAMPLES: The example given below is a typical situation in which the program is running and a linkage error is encountered. The resolve_linkage_error command is issued, correcting the linkage error and allowing the program to continue.

```
! myprog
     Error:  Linkage error by >udd>m>vv>myprog|123
     referencing subroutine$entry
     Segment not found.
     r 1234 2.834 123.673 980 level 2, 26

! rle mysub$mysub_entry
     r 1234 0.802 23.441 75 level 2, 26

! start
     <myprog is running>
```

SYNTAX AS A COMMAND:

reserve_resource -control_arg

FUNCTION: reserves a resource or group of resources for use by the calling process. The reservation takes effect immediately and lasts until cancelled by the cancel_resource command or by process termination.

-resource STR, -rsc STR
specifies a description of the resources to be reserved. If this resource description contains spaces or special characters, it must be enclosed in quotes. This resource description can also have control arguments and is described in more complete detail below.

NOTES ON RESOURCE DESCRIPTION: A resource description describes certain devices and volumes by name or by attributes and an optional number. It has the following format:

{-resource_type} resource_spec1 ... {-resource_type resource_specn}

That is, a series of at least one resource_spec where all but the first must be preceded by the -resource_type or -rsct control argument. The first need not be preceded by the control argument.

The format of a resource_spec is as follows:

volume_type name1 {names}
or:
device_type {names}
or:
device_type {-control_args}

where:

volume_type
can be either tape_vol or disk_vol. At least one name must be specified with volume_type, and it is the name of the volume, for example, 050102.

device_type
can be either tape_drive or disk_drive. Names are the names of devices such as tape_01, and if names are specified with a device_type, no control arguments are allowed. Names are not required, therefore a device_type can be specified with no names or control arguments. If names are not specified, the control_args for use with device_type can be chosen from the following:

-attributes STR, -attr STR
the attribute string STR consists of a string of attributes with values separated by commas with no spaces. For tape drives the attributes are:

mode=
track=
den=

For disk drives the only attribute is:

model=

Suitable values for these attributes can be found by using the list_resource_types command also in this manual.

-number N, -nb N
is the number of identical resources of the type desired.

EXAMPLES:

```
rsr -rsc "tape_vol 50102 u-309 -rsct tape_drive -attr
     track=9,den=800 -nb 2"
```

This command line reserves four resources: two tapes, 050102 and u-309; two tape drives, both being 9-track and capable of 800 bpi operation.

resource_usage (ru)

SYNTAX AS A COMMAND:

ru {-control_arg}

FUNCTION: prints a month-to-date report of the resource consumption of the user.

CONTROL ARGUMENTS:
These are used to select portions of the available resource usage information. Only one of the following can be specified:

-total, -tt
prints the user's dollar totals figures including the month-to-date dollar charge, the monthly spending dollar limit, and the absolute total spending.

If the project administrator has set an absolute dollar limit for the user (this limit is independent of the monthly spending limit), this absolute limit is printed, along with the date on which this limit was last reset and the limit's reset interval. The absolute total spending mentioned above is the dollar charge against this absolute limit. In cases where no absolute limit has been set, the absolute total spending represents charges running from the date that the user was registered on the system.

-brief, -bf
prints a header describing the resource usage reporting period, followed by the month-to-date dollar charge, the monthly spending dollar limit and three dollar totals figures giving the user's interactive, absentee, and I/O daemon usage.

-long, -lg
prints the most comprehensive picture of the user's resource usage. This display includes the information selected by the -brief control argument and includes an expanded report of interactive, absentee, I/O daemon, and device usage.

For interactive usage, the dollar charge is broken down according to shift, monthly dollar limit per shift, charged virtual CPU time, charged terminal connect time and charged memory units expressed in thousands. Absentee usage is presented in terms of usage per queue: number of dprint/dpunch requested pieces, charged virtual CPU time, and charged lines of printed or punched output expressed in thousands.

The device usage category includes charges for tape (time spent with a drive assigned), tape mounts, disk (time spent with a disk drive assigned), disk mounts, and logical volumes (time spent with a private logical volume attached). In addition, a site can define devices corresponding to the various lines (tty or network) by which the system is accessed, and set prices for their usage.

NOTES: If no control argument is specified, the default action results in the selection of slightly less extensive resource usage information than that printed by the -long control argument; namely, all dollar charges are printed but resource usage expressed as time is not printed.

The system calculates a user's month-to-date dollar charges when it creates the user's process. A user wishing the most up-to-date figures should issue the new_proc command prior to typing resource_usage.

Notice that in a given usage report, shift and queue numbers may not appear in consecutive order because only shifts or queues with accrued charges are listed.

If no dollar limit stop has been set by a user's project administrator, the resource usage report indicates this by the printing of "open" as the dollar limit entry.

It is not possible for a user to invoke this command to obtain information about another user's resource consumption.

SYNTAX AS AN ACTIVE FUNCTION:

[response arg {-control_arg}]

FUNCTION: asks the user a question and returns the answer typed by the user. The answer is not returned in quotes; the command processor therefore treats the answer as several strings if it contains spaces. To treat the returned string as a single argument to the command processor, the]] feature of the command language can be used.

See also the query active function, which requires a yes or no answer and returns true or false.

ARGUMENTS:

arg
is the question to be asked. If arg contains spaces or other command language characters, it must be enclosed in quotes.

CONTROL ARGUMENTS:

-non_null
indicates that the user must give a response. If the user replies with an empty (blank) line, the response active function prints a message explaining that a null response is not allowed and repeats the question.

-accept STRs
where STRs are the only responses accepted from the user. If a STR contains spaces or other command language characters, it must be enclosed in quotes. If the user responds to the question with an answer that is not one of the specified STRs, the response active function prints a message explaining that the user's answer is unacceptable, lists the acceptable answers, and repeats the question.

EXAMPLES:

Assume that dpc is an abbreviation for:

```
do "dp -cp [response ""Number of copies?    ""] &f1"
```

then the following interaction:

```
! dpc report_1.runout memo_phone.runout
  Number of copies?    ! 2
```

gets the user two copies of each runout segment.

Assume that the exec_com segment named x.ec contains the following line:

```
dprint -header [response "What header?"
      -non_null] -copy 2 report.print
```

then the following interaction:

```
! ec x
  What header?   ! <carriage return>
  response:  Null response not allowed, please retype.

  What header?   ! Jones
```

prints two copies of report.print with a header of Jones. The use of the -non_null control argument ensures that a header follows the -header control argument to the dprint command; otherwise, the -copy control argument to the dprint command would be interpreted as the header and the number 2 would be interpreted as the segment name.

Assume the k.ec exec_com segment contains the following line:

```
dp -rqt [response "Which rqt?" -accept printer unlined
   remote] resume
```

then the following interaction:

```
! ec k

  Which rqt?  ! plotter
  response:  'plotter' is not an acceptable answer.
  Acceptable answers are:
        'printer'
        'unlined'
        'remote'

  Which rqt?  ! printer
```

enters a printer request for one copy of the resume segment.

SYNTAX AS A COMMAND:

```
rv str
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[rv str]
```

FUNCTION: returns the characters of a specified string in reverse order.

EXAMPLES:

```
! string [rv abcdef]
  fedcba
```

reverse_after (rvaf)

SYNTAX AS A COMMAND:

rvaf strA strB

SYNTAX AS AN ACTIVE FUNCTION:

[rvaf strA strB]

FUNCTION: performs the same function as the after active function, but in reverse order.

NOTES: The reverse_after active function returns that part of strA following the last occurrence of strB in strA (after uses the first occurrence). If strB occurs last in strA or does not occur at all, a null string is returned.

[reverse_after strA strB]

is the same as:

[reverse [before [reverse strA] [reverse strB]]]

when strB appears in strA. It is a null string when strB does not appear in strA.

EXAMPLES:

```
! string [reverse_after abcdef123def456 def]
456
! string [rvaf acebdf g]

! string XY[rvaf 17.245et17 17]ZZ
XYZZ
```

reverse_before (rvbe)

SYNTAX AS A COMMAND:

rvbe strA strB

SYNTAX AS AN ACTIVE FUNCTION:

[rvbe strA strB]

FUNCTION: performs the same function as the before active function, but in reverse order.

NOTES: The reverse_before active function returns that part of strA preceding the last occurrence of strB in strA (before uses the first occurrence). If strB occurs first and nowhere else in strA, a null string is returned. If strB does not occur in strA, strA is returned.

[reverse_before strA strB]

is the same as:

[reverse [after [reverse strA] [reverse strB]]]

when strB appears in strA. It is the same as strA when strB does not appear in strA.

EXAMPLES:

```
! string [reverse_before abcdef123def456 def]
  abcdef123
! string [rvbe acebdf g]
  acebdf
! string XY[rvbe 17.245et17 17]ZZ
  XY17.245e+ZZ
```

reverse_decat (rvdecat)

SYNTAX AS A COMMAND:

rvdecat strA strB C

SYNTAX AS AN ACTIVE FUNCTION:

[rvdecat strA strB C]

FUNCTION: performs the same function as the decat active function, but in reverse order.

NOTES: The reverse_decat active function returns the decatenation of strA with respect to the last occurrence of strB in strA (decat uses the first occurrence). The value for C is any three digit bit string expressed as 0 or as 1 characters such as 000,001,...,111. The last occurrence of strB found in strA divides strA into three parts: the part prior to strB, the part matching strB, and the part following strB. Digits of C correspond to these three parts. The return string contains the parts of strA whose corresponding bit in C is 1. The parts are returned in their original order of appearance in strA.

[reverse_decat strA strB C]

is the same as:

[reverse [decat [reverse strA] [reverse strB] [reverse C]]]

when strB appears in strA. It is also the same as:

[decat strA strB C]

when strB does not appear in strA.

EXAMPLES:

```
! string [rvdecat abcdef123defghi def 110]
  abcdef123def
! string [rvdecat abcdef g 100]
  abcdef
```

reverse_index (rvindex)

SYNTAX AS A COMMAND:

```
rvindex strA strB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[rvindex strA strB]
```

FUNCTION: performs the same function as the index active function, but in reverse order.

NOTES: The reverse_index active function returns the index (character position) in strA of the beginning of the last occurrence of strB (index uses the first occurrence). If strB does not appear in strA, 0 is returned.

```
[reverse_index strA strB]
```

is the same as:

```
[length strA] - [index [reverse strA] [reverse strB]]
     + 2 - [length strB]
```

when strB appears in strA. It is 0 when strB does not appear in strA.

EXAMPLES:

```
! string [rvindex abc123defghi123jkl 123]
  13
! string [rvindex "Now is the time." hte]
  0
! string [rvindex abcdefghi ef]
  5
```

SYNTAX AS A COMMAND:

rvsrh strA strB

SYNTAX AS AN ACTIVE FUNCTION:

[rvsrh strA strB]

FUNCTION: performs the same function as the search active function, but in reverse order.

NOTES: The reverse_search active function returns the index (character position) of the last character in strA that appears in strB (search returns the first such character). If no characters of strA appear in strB, 0 is returned.

[reverse_search strA strB]

is the same as:

[length strA] - [search [reverse strA] strB] + 1

when a character of strB appears in strA. It is 0 when a character of strB does not appear in strA.

EXAMPLES:

```
! string [rvsrh "abc = 213" 0123456789]
9
! string [rvsrh "abc = def" 0123456789]
0
```

reverse_verify (rvverify)

SYNTAX AS A COMMAND:

rvverify strA strB

SYNTAX AS AN ACTIVE FUNCTION:

[rvverify strA strB]

FUNCTION: performs the same function as the verify active function, but in reverse order.

NOTES: The reverse_verify active function returns the index (character position) of the last character in strA that does not appear in strB (verify returns the first such character). If all characters of strA appear in strB, 0 is returned.

[reverse_verify strA strB]

is the same as:

[length strA] - [verify [reverse strA] strB] + 1

when the characters of strA do not appear in strB. It is 0 when all characters of strA appear in strB.

EXAMPLES:

```
! string [rvverify "abc = 123" 0123456789]
6
! string [rvverify "abc = def" 0123456789]
9
! string [rvverify 21435 0123456789]
0
```

SYNTAX AS A COMMAND:

```
rtrim strA strB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[rtrim strA strB]
```

FUNCTION: returns a character string trimmed of specified characters from the right.

NOTES: The rtim active function finds the last character of strA not in strB, trims characters from strA following this character and returns the trimmed result. Space characters are trimmed if strB is omitted.

EXAMPLES:

```
! string [rtrim 000305.000 0]
  000305.
! string [rtrim [ltrim 000305.000 0] 0]
  305.
! string X[rtrim "   This is it.   "]Y
  X   This is it.Y
```

SYNTAX AS A COMMAND:

run {-control_args} {program} {program_args}

FUNCTION: provides the user with a temporary, somewhat isolated, environment for the execution of programs.

ARGUMENTS:

program
is the reference name or pathname of the main program for the run unit.

program_args
are the arguments passed to the main program or the exec_com specified by -exec_com. See "Notes on the exec_com Feature" below.

CONTROL ARGUMENTS:

-exec_com path, -ec path
executes the exec_com path after the environment of the run unit is established.

-no_exec_com, -nec
always invokes the main program directly.

-limit N, -li N
interrupts the run unit every N seconds of virtual CPU time to ask the user "Time limit reached. Do you want to continue the program?"

-new_reference_names, -nrn
begins the run unit without any reference names known. The reference name control arguments are explained further in "Notes on Reference Name Control Arguments". This is the default.

-copy_reference_names, -crn
begins the run unit with a copy of the reference names initiated prior to the run unit. See "Notes on Reference Name Control Arguments" below.

-old_reference_names, -orn
uses the same reference names inside and outside the run unit. See "Notes on Reference Name Control Arguments" below.

NOTES: The -new_reference_names, -copy_reference_names, and -old_reference_names control arguments are mutually incompatible.

NOTES ON FUNCTION: The run command is primarily intended to aid users in executing programs written on systems that have a different definition of program than the Multics system has. Within the Multics system, a process is a program. Although many separately compiled "programs" can be executed in a process, they all share such program environment items as FORTRAN common blocks, PL/I external static variables, reference names, and file openings. This can cause problems for users with programs that depend on, for example, FORTRAN common being reinitialized each time the program is executed. However, within the Multics system a run unit is also a program. The run command executes the specified program in a temporary program environment that is separate from the rest of the process and from any other run unit.

The program attributes that are managed (restored, reset, etc.) by the run command are:

- PL/I internal static storage.
- PL/I external static variables whose names do not contain "$".
- FORTRAN common blocks whose names do not contain "$".
- allocations of PL/I based and controlled storage when the program does not specify a particular area.
- files used only through programming language I/O constructs.
- reference names and search rules.

All of these are restored to their prior state when the run unit terminates. An optional feature, specified by the -old_reference_name control argument, does not restore search rules or reference names.

NOTES ON REFERENCE NAME CONTROL ARGUMENTS: The three reference name control arguments affect the management of the address space (the segments known to the process). Reference names are the names by which the known segments are initiated. The

default action during environment restoration is to terminate all segments made known (initiated for the first time) inside the run unit. When this happens, all the reference names used inside the run unit are discarded and the names in use prior to the run unit are restored.

The -nrn control argument causes segments to be terminated and reference names and search rules to be restored at the end of the run unit. The run unit begins without any reference names. This option should be chosen when the user wants to be sure that the program is calling all the right subroutines and not inadvertantly getting some subroutine that happened to have the same name but was used earlier in the process.

The -crn control argument also causes segments to be terminated and reference names and search rules to be restored at the end of the run unit. The run unit begins with the reference names already initiated. This option is suitable when the user does not expect name conflicts with already known segments.

The -orn control argument does not terminate segments and does not manipulate reference names at the end of the run unit. It is generally safer. It is necessary in certain situations, such as when files are opened before the run unit begins.

NOTES ON THE exec_com FEATURE: The run command uses an exec_com segment if the -exec_com control argument is specified or if -no_exec_com is not specified and the segment program_name.run.ec is found in the same directory as the main program. Otherwise, the specified main program is invoked directly.

If an exec_com segment is used, all command arguments after the run control arguments are passed to the exec_com.

If an exec_com is not used, the first non-control argument is interpreted as the name of the main program for the run.

NOTES ON SEARCH RULES: The search rules in effect at the beginning of the run unit are always the same as those used just before the run unit. In order to get the default system search rules, an exec_com must be used that invokes the

set_search_rules command with no arguments before executing the main program.

If -old_reference_names (-orn) is specified, any changes to the search rules remain when the run unit ends. Otherwise, the search rules are restored to the values they had at the beginning of the run.

NOTES ON TERMINATING RUN UNIT: There are several ways to terminate a run unit.

- return from the main program or exec_com invoked by the run command.
- execute a stop statement in PL/I or FORTRAN.
- invoke the stop_run command. This can be done from a program, an exec_com, or interactive command level.
- invoke the release (rl) command.

Either executing a stop statement or calling stop_run causes the finish condition to be signalled. User code can optionally be called during run unit termination. Refer to the description of add_epilogue_handler_ in the MPM Subsystem Writers' Guide.

NOTES ON RUN UNIT LIMITATIONS: Run units incur significant overhead costs. Run units should be used primarily for debugging and executing user written programs.

Any files attached/opened via iox_ or io_call during the run unit must be explicitly detached/closed before the run unit terminates.

If any files used in the run unit are opened before the run unit begins, the -orn control argument must be used.

The trace and change_error_mode commands should not be used inside run units.

The answer command's command line cannot include the run command.

The profile command cannot be used on an object segment that was executed inside a run unit unless the object segment's perprocess static switch was turned on. (See "Notes to Subsystem Writers" below.)

The list_external_variables, delete_external_variables, and reset_external_variables commands do not handle any PL/I external static variables or FORTRAN common blocks in the pre-run unit environment that object segments with perprocess static use.

NOTES TO SUBSYSTEM WRITERS: If a procedure's internal static storage and linkage section are to be left alone during the run unit, the object segment must be given the perprocess_static attribute. This can be done either by including the global keyword "Perprocess_Static;" in the bindfile (with no parameters) for bound segments, or by including the perprocess_static pseudo-op in alm procedures, or by typing:

```
! perprocess_static_sw_on segname
```

There is a corresponding perprocess_static_sw_off command. Both are described in the MPM Subsystem Writers' Guide.

Object segments without perprocess static that are used both inside and outside the run units should not have internal static pointers to named temporary external segments. Each execution of such an object segment in a run unit destroys the previous contents of the segment. Instead, the internal static pointer should point to a segment managed by the Multics temporary segment facility (see the description of get_temp_segments_ in the MPM Subroutines). If using temporary segments is inappropriate because the information must be cumulative, the object segment must have perprocess static.

The run command sets up a "condition wall" so that procedures before the run command on the stack do not get control (as the result of a signalled condition) until the run unit is terminated.

EXAMPLES: The following command line shows how to run the program prog2 with -old_reference_names:

```
! run -orn prog2 prog2_arg1
```

The following exec_com uses the default search rules and invokes a program whose arguments were given to the run command:

```
&command_line off
set_search_rules
prog2 &f1
&quit
```

The run command line that uses this exec_com might be:

```
! run prog2_ec prog2_arg1
```

The example shown above is useful when the user wants to call a library subroutine and not a private subroutine that has the same name and that was already used in the process. The example should not be used with the -orn control argument because the search rules would be changed permanently, and the main purpose of the example is to avoid using existing reference names.

The following example shows how to invoke an arbitrary command from an exec_com within the environment of a run unit:

```
&command_line off
&f1
&quit
```

SYNTAX AS A COMMAND:

rc name {-control_args}

FUNCTION: explicitly initiates execution of a COBOL program.

ARGUMENTS:

name
is the reference name or pathname of the "main program" in which execution is to be initiated. If a pathname is specified, the specified segment is initiated with a reference name identical to the entryname portion of the pathname. Otherwise, the search rules are used to locate the segment. If the name specified in the PROG-ID statement of the COBOL program (i.e., the entry point name) is different from the current reference name of the object segment, then the name specified here must be in the form A$B where A is the pathname or reference name of the segment and B is the PROG-ID as defined in the IDENTIFICATION DIVISION of the source program.

CONTROL ARGUMENTS:

-cobol_switch N, -cs N
sets one or more of the eight COBOL-defined "external switches" on, where N is a number from 1 to 8 (or a series of numbers separated by spaces) that corresponds to the numbered external switch. At the outset of the run unit, the default setting of these external switches is off. (The eight external switches are defined in the Multics COBOL Reference Manual, Order No. AS44.)

-no_stop_run, -nsr
avoids establishment of a handler for the stop_run condition. (See "Notes" below.)

-sort_dir path, -sd path
specifies the directory to be used during execution of this run unit for temporary sort work files. If this control argument is not specified, the process directory is assumed.

-sort_file_size N, -sfs N
is the floating point representation of the estimated average size in characters of the files to be sorted during execution of this run unit. This information is used to optimize sorting. If not specified, 1e6 is assumed (i.e., one million characters).

NOTES: This command is not needed to execute COBOL object programs on Multics; it is used to simulate an environment in which traditional COBOL concepts can be defined easily. This command cannot be called recursively.

This command enables the user to explicitly define and start execution of a COBOL run unit. A run unit is either explicitly started by the execution of the run_cobol command or implicitly started by the execution of a COBOL object program either by invocation from command level or from a call by another program written in COBOL or another language. A run unit is stopped either by the execution of the STOP RUN statement in a COBOL object program or by invocation of the stop_cobol_run command. For the duration of time after a run unit is started and before it is stopped, it is said to be active. All COBOL programs executed while a run unit is active are considered part of that run unit.

A run unit is a subset of a Multics process; it is stopped when the process is ended. Also, when all programs contained in a run unit are cancelled, the run unit is stopped (refer to the cancel_cobol_program command). Only one run unit can be active at any given time in a process. Therefore, the run_cobol command cannot be invoked recursively. Additionally, if a run unit has been started implicitly (as described above), the run_cobol command cannot be used until that run unit has been stopped; i.e., the run_cobol command does not terminate a currently active run unit.

The explicit creation of a run unit with the run_cobol command performs the following functions:

1. Establishment of a "main program", i.e., a program from which control does not return to the caller. The EXIT PROGRAM statements, when encountered in such a program, have no effect, as required in the COBOL definition. An implicitly started run unit has no "main program". The EXIT PROGRAM statement in all programs contained in such a run unit always causes control to be returned to the caller, even if the caller is a system program, e.g., the command processor.

2. Setting of the COBOL external switches. These switches are set to off unless otherwise specified by the -cobol_switch control argument.

3. User control of the action taken when a STOP RUN statement is executed in a COBOL object program. The action normally taken for STOP RUN is cancellation of all programs in the run unit, closing any files left open. After this has been done, the data associated with any of the programs is no longer available. Thus in a debugging environment, it may be useful to redefine the action taken for STOP RUN. When the run unit is explicitly initiated with the run_cobol command, the STOP RUN statement causes the signalling of the stop_run condition for which a handler is established that performs the normal action described above. If the -no_stop_run control argument is specified, this handler is not established, thus allowing the user to handle the signal using other Multics commands. If the user has not explicitly provided a handler for stop_run and specifies the -no_stop_run control argument, an unclaimed signal results.

The name specified in the run_cobol command line need not be a COBOL object program. It can be a program produced by any language compiler that provides a meaningful interface with COBOL programs (e.g., PL/I, FORTRAN).

Refer to the following related commands:

display_cobol_run_unit, dcr
stop_cobol_run, scr
cancel_cobol_program, ccp
run

SYNTAX AS A COMMAND:

rf paths {-control_args}

FUNCTION: is used to type out text segments in manuscript form.

ARGUMENTS:

paths
are the pathnames of input segments or multisegment files named entryname.runoff. The runoff suffix must be the last component of each entryname; however, the suffix need not be supplied in the command line. If two or more pathnames are specified, they are treated as if runoff had been invoked separately for each one. The segments are printed in the order in which they occur in the invocation of the command.

CONTROL ARGUMENTS:
can be chosen from the following list. Any control argument specified anywhere in the command invocation applies to all segments; control arguments can be intermixed arbitrarily with segment names. Control arguments must be preceded by a minus sign.

-ball N, -bl N
converts output to a form suitable for an N typeball on a unit equipped with a selectric-type typing element. Acceptable ball numbers are 041, 012, 015, and 963. The default is the form of the terminal device being used. Use of this control argument overrides any specification set by the -device control argument (below).

-character, -ch
flags certain key characters in the output by putting the line containing the key character in a segment named entryname.chars. The normal output is not affected. Page and line numbers referring to the normal output appear with each flagged line, and reminder characters, enclosed by color-shift characters, are substituted for the key characters. The default set of key and reminder characters corresponds to those unavailable with a 963 typeball, as follows:

Key	Reminder
left square bracket	<
right square bracket	>
left brace	(
right brace	)
tilde	t
grave accent	'

The key and reminder characters can be changed by use of the .ch control line; specifying a blank reminder character removes the associated key character from the set of key characters. If a key character would print normally in the output, it should also appear in a .tr control line to turn it into a blank in the output.

-device N, -dv N
prepares output compatible with the device specified. This is usually used when the output is stored in a segment to be printed elsewhere. Suitable devices are terminals 2741, 1050, 37, and the bulk output printers, 202 or 300. Use of this control argument overrides any specification set by use of the -ball control argument; if both are used in one invocation of runoff, the last one encountered prevails.

If neither -device nor -ball is specified, the default device type is that from which the user is logged in; any unrecognized device type is assumed to support the entire ASCII character set.

-from N, -fm N
starts printing at the page numbered N. If the -page control argument is used, printing starts at the renumbered page N.

-hyphenate, -hph
When this control argument is used, a procedure named hyphenate_word_, that the user supplies, is invoked to perform hyphenation when the next word to be output does not fit in the space remaining in a line (see "Hyphenation Procedure Calling Sequence" at the end of this description). Otherwise, no attempt is made to hyphenate words.

-indent N, -in N
indents output N spaces from the left margin (default indentation is 0 except for "-device 202," which is the default for -segment and has a default indentation of 20; see also -number below). This space is in addition to whatever

indentation is established by use of the .in control word.

-no_pagination, -npgn
suppresses page breaks in the output.

-number, -nb
prints source line numbers in the left margin of the output; minimum indentation of 10 is forced.

-page N, -pg N
changes the initial page number to N. All subsequent pages are similarly renumbered. If the control line .pa is used within the segment, the -page control argument is overridden and the page is numbered according to the .pa control line.

-parameter arg, -pm arg
assigns the argument arg as a string to the internal variable "Parameter".

-pass N
processes the source segments N times to permit proper evaluation of expressions containing symbols that are defined at a subsequent point in the input. No output is produced until the last pass.

-segment, -sm
directs output to the segment or multisegment file named entryname.runout. This control argument assumes by default that the material is to be dprinted, so the segment is prepared compatible with device 202 unless another device is specified; thus, unless overridden by the -indent control argument, each printed line in the output segment is preceded by 20 leading spaces so that the text is approximately centered on the page when dprinted.

-stop, -sp
waits for a carriage return from the user before beginning typing and after each page of output (including after the last page of output).

-to N
ends printing after the page numbered N.

-wait, -wt
waits for a carriage return from the user before starting output, but not between pages.

NOTES: Output lines are built from the left margin by adding text words until no more words fit on the line; the line is then justified by inserting extra blanks to make an even right margin. Up to 20 lines each of headers and footers can be printed on each page. The pages can be numbered, lines can be centered, and equations can be formatted. Space can be allowed for diagrams. Detailed control over margins, spacing, headers, justification, numbering, and other aspects of format is provided by control lines that begin with a period. Although the control lines are interspersed within the text, they do not appear in the output segment. The output can be printed page by page to allow positioning of paper, or it can be directed into a segment. Characters not available on the device to which output is directed are replaced by blanks. If special symbols must be hand drawn, a separate segment can be created that indicates where each symbol should be placed. The user can define variables and cause expressions to be evaluated; he also has the ability to refer to (and sometimes modify) variables connected with the workings of the runoff command.

A runoff input segment contains two types of lines: control lines and text lines. A control line begins with a period; all other lines are considered text lines. A two-character control word appears in the second and third character positions of each control line. The control word can take a parameter that is separated from the control word by one or more spaces. Lines that are entirely blank are treated as if they contained a .sp 1 control line.

Text lines contain the material to be printed. If an input line is too short or too long to fill an output line, material is taken from or deferred to the next text line. A line beginning with a space is interpreted as a break in the text (e.g., the beginning of a new paragraph) and the previous line is printed as is.

Tab characters (ASCII HT) encountered in the input stream are converted to the number of spaces required to get to the next tab position (11, 21, ...). Nonprinting control characters in the input segment are discarded in the output segment. The .tr control word can be used to print these control characters in the output segment.

When an input text line ends with any of the characters ".", "?", "!", ";", or ":", or with ".", "?", or "!" followed by a double quote or ")", two blanks precede the following word (if it is placed on the same output line), instead of the normal single blank.

The maximum number of characters per input or output line is 361; this permits 120 underlined characters plus the newline character.

Terminology

The following paragraphs describe various terms that are used throughout the runoff description.

FILL AND ADJUST MODES

Two separate concepts are relevant to understanding how runoff formats output: fill mode and adjust mode. In fill mode, text is moved from line to line when the input either exceeds or cannot fill an output line. Adjust mode right justifies the text by inserting extra spaces in the output line, with successive lines being padded alternately from the right and from the left. Initial spaces on a line are not subject to adjustment. Fill mode can be used without adjust, but in order for adjust to work, fill mode must be in effect.

LINE LENGTH

The line length is the maximum number of print positions in an output line, including all spaces and indentations, but not including margins set or implied by the -device, -indent, or -number control arguments.

BREAK

A break ensures that the text that follows is not run together with the text before the break. The previous line is printed out as is, without padding.

SPACING BETWEEN LINES

Vertical spacing within the body of the text is controlled by the three control words: .ss, .ds, and .ms (for single, double, and multiple spacing respectively). Single spacing is the default. Multiple spacing is set by the control line .ms N where N-1 is the number of blank lines between text lines.

PAGE EJECT

A page eject ensures that no text after the control line causing the page eject (e.g., .bp for "begin page") is printed on the current page. The current page is finished with only footers and footnotes at the bottom, and the next text line begins the following page.

MARGINS

There are four margins on the page vertically. The first margin on the page is the number of blank lines between the top of the page and the first header; this margin is set by the .m1 control word. The second, set by .m2, specifies the number of lines between the last header and the first line of text. The third (.m3) is between the last line of text and the first footer. The fourth (.m4) is between the last footer and the bottom of the page. The default for the first and fourth margins is four lines; for the second and third, two lines.

PAGE NUMBERS

As the output is being prepared, a page number counter is kept. This counter can be incremented or set by the user. The current value of the counter can be used in a header or footer through the use of the symbol "%". A page is called odd (even) if the current value of the counter is an odd (even) number.

The page numbers can be output as either arabic (the default) or roman (using the .ro control word).

HEADERS AND FOOTERS

A header is a line printed at the top of each page. A footer is a line printed at the bottom of each page. A page can have up to 20 headers and 20 footers. Headers are numbered from the top down, footers from the bottom up. The two groups are completely independent of each other. Provision is made for different headers and footers for odd and even numbered pages. Both odd and even headers (footers) can be set together by using the .he (.fo) control words. They are set separately by using the .eh, .oh, .ef, and .of control words.

A header/footer control line has two arguments, the line number (denoted in the control line descriptions as "#"), and the title.

The line number parameter of the control line determines which header or footer line is being set. If the number is omitted, it is assumed to be 1, and all previously defined headers or footers of the type specified (odd or even) are cancelled. Once set, a line is printed on each page until reset or cancelled.

The title part of the control line begins at the first nonblank character after the line number. This character is taken to be the delimiting character, and can be any character not used in the rest of the title. If the delimiting character appears less than four times, the missing last parts of the title are taken to be blank. The three parts of the title are printed left justified, centered, and right justified, respectively, on the line. Any or all parts of the title can be null. Justification and centering of a header or footer line are derived from the line length and indentation in effect at the time of the definition of the header or footer, and are used whenever that line is output, regardless of the values at the time of use. Any occurrence of the special character "%" within a title is replaced by the current value of the page counter whenever the title is printed. To cause a percent character to be printed, "%%" must be written in the title. The special character can be changed; see the .cc control word.

Omitting the title in the control line cancels the header or footer with that number, including its space on the page (e.g., ".he 4" cancels the fourth header). A blank line in the header or footer can be achieved by a title consisting entirely of one delimiting character (e.g., ".fo 3 $" makes the third footer a blank line). Omitting both number and title of a header (footer) cancels all headers (footers) of the type specified (e.g., ".oh" cancels all headers that were specified by any .oh control line).

Expressions and Expression Evaluation

An expression can be either arithmetic or string, and consists of numbers and operators in appropriate combinations. All operations are performed in integer format, except that string comparisons are performed on the full lengths of the strings.

The order of precedence for the operators is:

^ (bit-wise negation), - (unary)
*, /, \ (remainder)
+, - (binary)
=, <, >, ≠, ≤, ≥ (all are comparison operators that yield -1 for true or 0 for false)
& (bit-wise AND)
| (bit-wise OR), ≡ (bit-wise equivalence)

Other guidelines in the use of expressions are as follows:

1) Parentheses can be used for grouping.

2) Blanks are ignored outside of constants.

3) Octal numbers consist of "#" followed by a sequence of octal digits.

4) String constants are surrounded by the double quote character; certain special characters are defined by multiple-character sequences that begin with the * character, as follows:

** asterisk character
*" double-quote character

*b	backspace character
*n	newline character
*t	horizontal-tab character
*s	space character
*cN	character whose decimal value is N (where N is 1 to 3 digits)

5) Concatenation of strings is performed by the juxtaposition of the strings involved, in order, left to right.

6) For positive i, k,

```
string_expression(i)
```

and:

```
string_expression(i, k)
```

are equivalent to the PL/I substr builtin function references:

```
substr(string_expression, i)
```

and:

```
substr(string_expression, i, k)
```

respectively.

7) For negative i, the substring is defined as starting -i characters from the rightmost end of the string; for negative k, the substring ends -k characters from the end of the string.

8) Evaluation of substrings takes place after any indicated concatenations; string operations have higher precedence than all the binary operations.

9) In any context other than a .sr control line or in a string comparison, a string expression is converted to an integer in such a way that a one-character string results in the ASCII numeric value of the character.

Expression evaluation takes place under the following conditions:

1) In .sr and .ts control lines.

2) In all control lines that accept an "N" or "+N" argument.

Definition and Substitution of Variables

Variables can be defined by the use of the .sr control line; their values can be retrieved thereafter by a symbolic reference. Names of the variables are composed of the uppercase and lowercase alphabetic characters, decimal digits, and "_", with a maximum length of 361 characters. When a variable is defined, it is given a type based on the type of the expression that is to be its value, either arithmetic or string. Variables that are undefined at the time of reference yield the null string. When enclosed in quotes, a null string is equivalent to an arithmetic 0. Thus, if a variable called var has not been set, this "%var%" is equivalent to 0 when used as an expression.

In substitution of variables, the name of the variable is enclosed by "%"; other occurrences of the character '%" encountered during substitution of variables are replaced by the value of the page counter; if a "%" character is to occur in the resulting output, it must be coded as "%%" (but see also the .cc control word).

Substitution of variables can occur:

1) In control lines that take an expression argument if a "%" is found as either the first or second character of the argument (substitution of variables takes place before expression evaluation).

2) In .ur control lines.

3) In all titles ('part1'part2'part3'), whether in header/footer control lines or as equation lines.

Many of the variables internal to runoff are available to the user (a complete list is given at the end of this description). These variables include control argument values

(or their defaults), values of switches and counters, and certain special functions. However, the user need not worry about naming conflicts, since an attempt to redefine an internal variable that is not explicitly modifiable is ignored; i.e., the operation of the command is unaffected.

Two special built-in symbols in runoff are provided for use in footnote and equation numbering: "Foot" contains the value of the next footnote number available (or the current footnote if referred to from within the text of the footnote) and "Eqcnt" is provided for equation numbering. The value of "Foot" is incremented by one when the closing .ft of a footnote is encountered. Any reference to "Eqcnt" provides the current value and causes its value to be incremented by one automatically; thus its value should be assigned to a variable, and the variable should then be used in all further references to that equation number.

Default Conditions

When no control words are specified, runoff prints the text single spaced, right adjusted, with no headers, no footers, and no page numbers.

If page numbers are substituted in headers or equations, they are arabic.

A page consists of 66 lines, numbered 1 through 66. The first line is printed on line 7, and the last on line 60, if no headers or footers are used. If headers are used, there are four lines of top margin (.m1 4), the headers, two blank lines (.m2 2), and then the text. If footers are used, there are two lines skipped after the text (.m3 2), footers printed, and four lines of bottom margin (.m4 4).

A line is 65 characters long; the left margin is that of the typewriter. The output is compatible with whatever is normal for the device from which the runoff command is executed. The entire segment is printed, with no wait before beginning or between pages.

NOTES ON CONTROL LINE FORMATS: The following discussion gives a description of each of the control words that can be interspersed with the text for format control. Control lines do not cause an automatic break unless otherwise specified. Arguments of the control words are in the following form:

#	integer constant
N	integer expression
±N	integer expression preceded by optional + or - sign
<expression>	arbitrary expression (string or integer)
c	character
cd	character pair
f	segment name
'part1'part2'part3'	a title whose parts are to be left justified, centered, and right justified respectively.

.ad
Adjust: text is printed right justified. Fill mode must be in effect for right justification to occur. Fill mode and adjust mode are the default conditions. This control line causes a break.

.ar
Arabic numerals: when page numbers (% variable) are substituted into text or control lines as a result of a .ur control line or into a title or equation as it is printed, they are in arabic notation. This is the default condition.

.bp
Begin page: the next line of text begins on a new page of output. This control line causes a break.

.br
Break: the current output line is finished as is, and the next text line begins on a new output line.

.cc c
Control character: this control line changes the character used to surround the names of symbolic variables when they are referenced to c. The default special character is "%". The character specified by c must thereafter be used to

refer to symbolic variables, while percent signs are treated literally. Either ".cc %" or .cc restores the percent sign as the special character.

.ce N
Center: the next N text lines are centered. Control lines and blank lines are not counted as part of the N lines being centered. If N is missing, 1 is assumed. This control line implies ".ne N" (or ".ne 2N" if double spacing) so that all lines centered are on the same page. A break occurs.

.ch cd..
Characters: each occurrence of the character c is replaced in the chars segment (the segment named entryname.chars) by the character d, set off by color-shift characters. If the d character is blank, or an unpaired c character appears at the end of the line, the c character is not flagged; it either occurs as itself in the chars segment or not at all if no other character on the line was flagged.

.ds
Double space: begin double spacing the text. This control line causes a break.

.ef # 'part1'part2'part3'
Even footer: this defines even page footer line number #. If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all even footers defined by any .ef control line are cancelled. For more information, see the previous discussion entitled "Headers and Footers."

.eh # 'part1'part2'part3'
Even header: this defines even page header line number #. If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all even headers defined by any .eh control line are cancelled. For more information, see the previous discussion entitled "Headers and Footers."

.eq N
Equation: the next N text lines are taken to be equations. If N is missing, 1 is assumed. This control line implies ".ne N" (or ".ne 2N" if double spacing) so that all equations are on the same page. The format of the equations should be 'part1'part2'part3' just as in headers and footers.

.ex text
Execute: the remainder of the control line (text) is passed to the Multics command processor. Substitution of variables can occur if the first or second character of text is "%".

.fh 'part1'part2'part3'
Footnote header: before footnotes are printed, a demarcation line is printed to separate them from the text. The format of this line can be specified through the title in the .fh control line. This title is printed in the same manner as headers/footers and equations. The default footnote header is a line of underscores from column one to the right margin.

.fi
Fill: this control line sets the fill mode. In fill mode, text is moved from line to line to even the right margin, but blanks are not padded to justify exactly. Fill mode is the default condition. This control line causes a break.

.fo # 'part1'part2'part3'
Footer: even and odd footers are set at the same time; this is equivalent to:

```
.ef # 'part1'part2'part3'
.of # 'part1'part2'part3'
```

If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all footers are cancelled. For more information, see the discussion entitled "Headers and Footers."

.fr c
Footnote reset: this control line controls footnote numbering according to the argument c. Permitted values of c are:

t Footnote counter is reset at the top of each page. This is the default condition.
f Footnote counter runs continuously through the text.
u Suppress numbering on the next footnote.

.ft
Footnote: when .ft is encountered, all subsequent text until the next .ft line is treated as a footnote. Any further text on the .ft line is ignored. If a footnote occurring near the bottom of a page does not fit on the

page, as much as necessary is continued at the bottom of the next page. If a footnote reference occurs in the bottom or next to bottom line of a page, the current page is terminated and the line with the footnote reference is printed at the top of the next text page.

.gb STR
Go back: the current input segment is searched from the beginning until a line of the form ".la STR" is found; "STR" in this case means "the rest of the line." Processing is continued from that point.

.gf STR
Go forward: same as .gb, except search forward from the current position in the input segment.

.he # 'part1'part2'part3'
Header: even and odd headers are set at the same time. This is equivalent to:

```
.eh # 'part1'part2'part3'
.oh # 'part1'part2'part3'
```

If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all headers are cancelled. For more information, see the discussion entitled "Headers and Footers."

.if f <expression>
Insert file: the segment specified by f is inserted into the text at the point of the ".if f" control line. The inserted segment can contain both text and control lines. No break occurs. The effect is as if the control line were replaced by the segment. Inserts can be nested to a maximum depth of 30. The argument f is the entryname of a runoff input segment. If the runoff suffix is not included in the .if line, it is supplied. The input file is located by use of the translator search list that has the synonym trans. For more information on search lists, see the search facility commands and, in particular, the add_search_paths command description in this manual. If a second argument is provided, it is evaluated in the same fashion as the expression in .sr, and its value and type are associated with the identifier "Parameter"; if no second argument is provided the value of "Parameter" remains unchanged (or undefined). (In either case, the prior value of "Parameter" is not pushed down).

.in ±N
Indent: the left margin is indented N spaces by padding N leading spaces on each line. The right margin remains unchanged. By default N is 0. The margin can be reset with another ".in N" request. Either .in or ".in 0" resets the original margin. If N is preceded by a plus or a minus sign, the indentation is changed by N rather than reset. This control line causes a break.

.la STR
Label: defines the label STR for use as the target of the .gb or .gf control word.

.li N
Literal: this request causes the next N lines to be treated as text, even if they begin with a period (.). If N is not specified, 1 is assumed.

.ll ±N
Line length: the line length is set to N. The left margin stays the same, and no break occurs. If N is not specified, 65 is assumed. If N is preceded by a plus or a minus sign, the line length is changed by N rather than reset.

.ma ±N
Margins: top and bottom margins are set to N lines. If N is preceded by a plus or a minus sign, the margin is changed by N rather than reset. The margin is the number of lines printed above the first header and below the last footer. If N is not specified, 4 is assumed. This control line is equivalent to:

```
.m1 ±N
.m4 ±N
```

Note: Care should be taken in using a top or bottom margin of less than three lines if output is to be directed to an off-line printer (-sm). Such printers typically are set up to skip automatically the first and last 3 lines on each page. Runoff takes this into account by putting out fewer newlines to compensate for the printer; the compensation cannot work for .m1 or .m4 less than 3. It is possible to get around this problem by using the -device 037 control argument when invoking runoff and special control arguments to the

command that set up requests for the off-line printer (dprint).

.mp +N
Multiple pages: format the output text so that it prints on every Nth page (i.e., skips N-1 blank sheets of paper between printed pages). This control line is valid only for output intended for the bulk printer. If N is not specified, 1 is assumed.

.ms +N
Multiple space: begin multiple spacing text, leaving (N-1) blank lines between text lines. If N is preceded by a plus or a minus sign, the spacing is changed by N rather than reset. If N is not specified, 1 is assumed. This control line causes a break.

.m1 +N
Margin 1: the margin between the top of the page and the first header is set to N lines, or changed by N if N is signed. If N is not specified, 4 is assumed. See note in description of .ma.

.m2 +N
Margin 2: the number of blank lines between the last header and the first line of text is set to N, or changed by N if N is signed. If N is not specified, 2 is assumed.

.m3 +N
Margin 3: the number of blank lines printed between the last line of text and the first footer is set to N, or changed by N if N is signed. If N is not specified, 2 is assumed.

.m4 +N
Margin 4: the margin between the last footer and the bottom of the page is set to N lines, or changed by N if N is signed. If N is not specified, 4 is assumed. See note in description of .ma.

.na
No adjust: the right margin is not adjusted. This does not affect fill mode; text is still moved from one line to another. This control line causes a break.

.ne N
Need: a block of N lines is needed. If N or more lines

remain on the current page, text continues as before; otherwise, the current page is ejected and text continued on the next page. The number of lines remaining is calculated by subtracting from the current page length the sum of the number of lines already printed and the number of lines reserved for footers, footnotes, and bottom margins. No break is implied; if a line is partially formatted but not yet printed when the .ne is encountered, it is ignored in the calculation of lines remaining (i.e., it is neither printed nor in possession of reserved space). Similarly, a footnote or footer defined after the .ne does not have space reserved at the time the .ne is encountered. If N is not specified, 1 is assumed. If several .ne control lines occur consecutively, the N's are not added together; only the largest N has effect.

.nf
No fill: fill mode is suppressed, so that a break is caused after each text line. Text is printed exactly as it is in the input segment. This control line causes a break.

.of # 'part1'part2'part3'
Oddfooter: this defines odd page footer line number #. If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all footers defined by any .of control line are cancelled. For more information, see the discussion entitled "Headers and Footers."

.oh # 'part1'part2'part3'
Oddheader: this defines odd page header line number #. If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all headers defined by any .oh control line are cancelled. For more information, see the discussion entitled "Headers and Footers."

.op
Odd page: the next page number is forced to be odd by adding 1 to the page number counter if necessary. A break is caused and the current page is ejected. No blank even page is made; the even page number is merely skipped.

.pa +N
the current line is finished as is (i.e., a break occurs) and the current page is ejected. The page number counter is set to N, or is changed by N if N was signed. If N is omitted, the page number counter is incremented by 1.

.pi N
Picture: if N lines remain on the present page, N lines are spaced over; otherwise, the text continues as before until the bottom of the page is reached, N lines are skipped on the next page before any text is printed. Headers are printed normally; the space resolved is below the headers. This option can be used to allow for pictures and diagrams. If several .pi control lines occur consecutively, each N is added to the number of lines pending and the total is checked against the space remaining on the page. All pending space is allotted together. If the total is greater than the usable space on a page, the next page contains only headers and footers and the rest of the space is left on the following page. If N is not specified, 1 is assumed.

.pl ±N
Page length: the page length is set to N lines. If N is not specified, 66 is assumed. If N is preceded by a plus or a minus sign, the page length is changed by N rather than reset.

.rd
Read: one line of input is read from the user input I/O switch; this input line is then processed as if it had been encountered instead of the .rd control line. Thus it can be either a text line or a control line; a break occurs only if the replacement line is one that would cause a break.

.ro
Roman numerals: when page numbers (% variable) are substituted into text or control lines as a result of a .ur control line or into a title or equation as it is printed, they are in lowercase roman notation. This can be reset to arabic numerals (the default) by use of the .ar control line.

.rt
Return: cease processing characters from the current input segment. If the current input segment was entered by a .if control line in another segment, return to the line following the .if control line.

.sk N
Skip: N page numbers are skipped before the next new page by adding N to the current page number counter. No break in text occurs. This control line can be used to leave out a page number for a figure. If N is not specified, 1 is assumed.

.sp N
Space N lines: If N is not specified, 1 is assumed. If not enough lines remain on the current page, footers are printed and the page ejected, but the remaining space is not carried over to the next page. The N blank lines are produced in addition to any that may occur automatically due to a .ds or .ms control line. For example, if .sp 4 is used with .ss or .ms 1, in effect four blank lines appear between two text lines, with .ds or .ms 2, five lines appear, with .ms 3, six lines.

After skipping the space, the equivalent of a .ne 2 is performed in an attempt to avoid separating the first line of a paragraph at the bottom of a page from the rest of the paragraph on the next page. The .ne feature can be avoided, if the user so desires, by using a blank line rather than .sp. Otherwise, a blank line is treated as if it were a .sp 1 control line.

This control line causes a break.

Note: A series of .sp control lines such as:

```
.sp a
.sp b
```

is not always equivalent to a single .sp control line whose argument is the sum of the individual arguments:

```
.sp a+b
```

If the .sp a finishes a page, causing a page ejection, b blank lines are produced at the top of the new page. If .sp a+b is used, the space does not appear at the top of the next page.

.sr name <expression>
Set reference: associates value of <expression> with the identifier name. The type of name is set to the type of <expression> (either numeric or string); if the expression is not provided or cannot be properly evaluated, a diagnostic

message is printed. The name identifier can be either a user-defined identifier or one of the built-in symbols that the user can set (see "Built-In Symbols" below).

.ss
Single space: begin single spacing text. This is the default condition. This control line causes a break.

.tr cd..
Translate: the nonblank character c is translated to d in the output. An arbitrary number of cd pairs can follow the initial pair on the same line without intervening spaces. An unpaired c character at the end of a line translates to a blank character. (Translation of a graphic character to a blank only in the output is useful for preserving the identity of a particular string of characters, so that the string is neither split across a line, nor has padding inserted within it.) If several .tr control lines are used in a segment, the cd pairs are "added together." Also a particular c character can be translated to a different d character by using a new .tr control line to override the previous translation. To cancel a cd pair (i.e., have the c character print out as itself), use another .tr control line of the form ".tr cc". A .tr control line with no cd pair is ignored.

.ts N
Test: process the next input line if the value of N does not equal zero (false). If N is not specified, 1 is assumed.

.ty STR
Type: write STR (i.e., the rest of the control line) onto the error_output I/O switch. Substitution of variables can occur if the first or second character of STR is "%". If STR is omitted, a blank line is written onto the I/O switch.

.un N
Undent: the next output line is indented N spaces less than the current indentation. Adjustment, if in effect, occurs only on that part of the line between the normal left indentation and the right margin. If N is not specified, its value is the current indentation value (i.e., the next output line begins at the current left margin). This control line causes a break.

.ur text
Use reference: the remainder of the .ur control line (text) is scanned, with variables of the form "%name%" replaced by their corresponding values (converted back to character string form if they were numeric). The line thus constructed is then processed as if it had been encountered in the original input stream (e.g., it can be another control line, including possibly another .ur).

.wt
Wait: read one line from the user_input I/O switch and discard it (see the .rd control word description).

.*
This line is treated as a comment and ignored. No break occurs.

.~
This line is treated as a comment and ignored with respect to the output segment. However, the line is printed in the appropriate place in the chars output segment.

Summary of Control Arguments

-ball N, -bl N
Convert output to a form suitable for an N typeball.

-character, -ch
Create entryname.chars, listing page and line numbers with red reminder characters where certain characters, normally not printable, must be drawn in by hand.

-device N, -dv N
Prepare output compatible with device N.

-from N, -fm N
Start printing at the page numbered N.

-hyphenate, -hph
Call user-supplied procedure to perform hyphenation.

-indent N, -in N
Set initial indentation to N.

-no_pagination, -npgn
Suppress page breaks.

-number, -nb
Print source segment line numbers in output.

-page N, -pg N
Change the initial page number to N.

-parameter arg, -pm arg
Assign arg as a string to the internal variable "Parameter".

-pass N
Make N passes over the input.

-segment, -sm
Direct output to the segment or multisegment file named entryname.runout, where entryname is the name of the input segment.

-stop, -sp
Wait for a carriage return before each page.

-to N
Finish printing after the page numbered N.

-wait, -wt
Wait for a carriage return before the first page.

Summary of Control Words

The following conventions are used to specify arguments of control words:

#	integer constant
c	character
cd	character pair
exp	expression (either numeric or string)
N	integer expression
+N	+ indicates update by N; if sign not present, set to N
f	segment name
t	title of the form 'part1'part2'part3'

runoff (rf)

Request	Break	Default	Meaning
.ad	yes	on	Right justify text
.ar	no	arabic	Arabic page numbers
.bp	yes		Begin new page
.br	yes		Break, begin new line
.cc c	no	%	Change special character from % to c
.ce N	yes	N=1	Center next N lines
.ch cd....	no		Note "c" in chars segment as "d"
.ds	yes	off	Double space
.ef # t	no		Defines even footer line #
.eh # t	no		Defines even header line #
.eq N	yes	N=1	Next N lines are equations
.ex text	no		Call command processor with "text"
.fh t	no	line of underscores	Format of footnote demarcation line
.fi	yes	on	Fill output lines
.fo # t	no		Equivalent to: .ef # t .of # t
.fr c	no	t	Control footnote numbering: "t" reset each page "f" continuous "u" numbering suppressed for next footnote
.ft	no		Delimits footnotes
.gb STR	no		"go back" to label STR
.gf STR	no		"go forward" to label STR
.he # t	no		Equivalent to: .eh # t .oh # t
.if f exp	no		Segment f.runoff inserted at point of request; value of "exp" assigned to "Paramater"
.in +N	yes	N=0	Indent left margin N spaces
.la STR	no		Define label STR
.li N	no	N=1	Next N lines treated as text
.li +N	no	N=65	Line length is N
.ma +N	no	N=4	Equivalent to: .m1 +N .m4 +N

runoff (rf)

Request	Break	Default	Meaning
.mp +N	no	N=1	Print only every N-th page
.ms +N	yes	N=1	Multiple space N lines
.m1 +N	no	N=4	Margin above headers set to N
.m2 +N	no	N=2	Margin between headers and tect set to N
.m3 +N	no	N=2	Margin between text and footers set to N
.m4 +N	no	N=4	Margin below footers set to N
.na	yes	off	Do not right justify
.ne N	no	N=1	Need N lines; begin new page if not enough remain
.nf	yes	off	Do not fill output lines; print them exactly as entered
.of # t	no		Defines odd footer line #
.oh # t	no		Defines odd header line #
.op	yes		Next page number is odd
.pa +N	yes		Begin page N
.pi N	no	N=1	Skip N lines if N remain; otherwise skip N lines on next page before any text
.pl +N	no	N=66	Page length is N
.rd	no		Read one line of text from the user_input I/O switch and process it in place of .rd line
.ro	no	arabic	Roman numeral page numbers
.rt	no		"Return" from this input segment
.sk N	no	N=1	Skip N page numbers before next new page
.sp N	yes	N=1	Space N lines
.sr sym exp	no		Assign value of "exp" to variable named "sym"
.ss	yes	no	Single space
.tr cd....	no		Translate nonblank character c into d on output
.ts N	no	N=1	Process the next input line only if N is not zero
.ty STR	no		Write "STR" onto the error_output I/O switch
.un N	yes	left margin	Indent next text line N spaces less
.ur text	no		Substitute values of variables in "text", and scan the line

Request	Break	Default	Meaning
			again
.wt	no		Read one line of text from the user_input I/O switch and discard it (for synchronization with terminal)
.*	no		Comment line; ignored
.~	no		Comment line; ignored

Built-in Symbols

Only those symbols marked yes in the Set column can have values assigned by the user.

All symbols are of type Number unless they are specified to be of type String.

Control words and control arguments that affect the values of the variables are indicated in parentheses: (x/y) indicates that x sets the switch to true (-1), and y sets it false (0); (a) or (a, b, c) indicates that it is affected by a or by a, b and c.

Symbol	Set	Value
Ad		Adjust (.ad/.na)
Ce		Number of lines remaining to be centered (.ce)
CharsTable	yes	Translation table for chars segment output (String) (.ch)
Charsw	yes	A chars segment is being created (-character)
ConvTable	yes	Translation table for output. Product of DeviceTable and TrTable (String) (.tr, -device)
Date		Date of this invocation of runoff; format is mm/dd/yy (String)
Device	yes	Type of device output is to be formatted for (-device, -ball, -segment)

Symbol	Set	Value
DeviceTable	yes	Translation table for physical device (String) (-device)
Eq		Equation line counter (.eq)
Eqcnt	yes	Equation reference counter (incremented each reference)
ExtraMargin	yes	Indent entire text this many spaces (-segment, -device, -indent)
Fi		Fill switch (.fi/.nf)
FileName		Name of current primary input segment (String)
Filesw		True if output is going to a segment (-segment)
Foot	yes	Footnote counter (.ft, .fr)
FootRef	yes	Footnote reference string in footnote body (String)
Fp	yes	First page to print (set at the beginning of each pass to the value of From)
Fr		Footnote counter reset switch
From	yes	First page to print (-from)
Ft		Footnote processing switch (.ft)
Hyphenating	yes	True if an attempt to break a word should be made (-hyphenate)
In		Indent to here (.in)
InputFileName		Name of current input segment (String) (.if)
InputLines		Current line number in current source file
LinesLeft		Number of usable text lines left on this page
Ll		Line length (.ll)
Lp	yes	Last page to print (initialized each pass from To)
Ma1		Space above header (.ma, .m1)
Ma2		Space below header (.m2)
Ma3		Space above foot (.m3)
Ma4		Space below foot (.ma, .m4)
Ms		Spacing between lines (ss = 1, ds = 2, etc.) (.ms, .ss, .ds)
MultiplePagecount		Form feeds between pages to printer (.mp)
NestingDepth		Index into stack of input files (.if)

Symbol	Set	Value
Nl		Last used line number
NNp	yes	Next page number (-page, .pa)
NoFtNo		True to suppress number on next footnote reference (.fr)
NoPaging	yes	True if no pagination is desired (-no_pagination)
Np	yes	Current page number (.pa, -page, initialized each pass from Start)
PadLeft		Alternate left/right padding switch (.un, .ad)
Parameter	yes	Argument passed during insert processing (-parameter, .if)
Passes	yes	Number of passes left to make (= 1 when printing is being performed) (-pass)
Pi		Space needed for pictures (.pi)
Pl		Page length (.pl)
Print	yes	Whether or not to print ((Fp $\leq$ Np $\leq$ Lp) & (Passes $\leq$ 1))
Printersw		Output is intended for bulk printer (-device, -segment)
PrintLineNumbers	yes	True if source line numbers are to be printed in output (-number)
Roman		Roman numeral pagination (.ro/.ar)
Selsw		True if typeball other than 963 is being used (-ball)
Start	yes	Initial page number (-page)
Stopsw	yes	Stop between pages of output (-stop)
TextRef	yes	Footnote reference string in main text (String)
Time		Local time, in seconds, since January 1, 1901.
To	yes	Last page to be printed (-to)
TrTable	yes	Translation table for user-supplied substitutions (String) (.tr)
Un		Undent to here (.un)
Waitsw	yes	Wait for input before printing first page (-wait)

NOTES ON HYPHENATION PROCEDURE CALLING SEQUENCE: The runoff command provides a means whereby a user-supplied program can be called whenever the space available on a line is less than the length of the next word (including attached punctuation, if any). The mechanism is activated by use of the -hyphenate control argument, and the PL/I calling sequence is provided below.

```
declare hyphenate_word_ entry(char(*) unaligned, fixed bin,
   fixed bin);

call hyphenate_word_(string, space, break);
```

LIST OF HYPHENATION CONTROL ARGUMENTS:

string
is the text word that is to be split. (Input)

space
is the number of print positions remaining in the line. (Input)

break
is the number of characters from the word that should be placed on the current line; it should be at least one less than the value of space (to allow for the hyphen), and can be 0 to specify that the word is not to be broken. Thus if the word "calling" is to be split, and 6 spaces remain in the line, the procedure should return the value 4 (adjustment is performed after hyphenation). (Output)

EXAMPLES: The following pages show the creation of a runoff segment and the result of invoking the runoff command on that segment. For an explanation of any of the control lines, refer to the respective control word definition earlier in this command description. Particularly notice the following:

1) The line length control is given before any headers and footers. A user who wants a line length other than the default one specifies it before specifying his headers and footers; if the user does not, the headers and footers on the first page are formatted for the default line length.

2) The .sr control line associates the page number count, at the time the title "RUNOFF SAMPLE PAGE" is printed, with the identifier rfsample. Refer to the last line of the segment (the .ur control line) to see how this reference is used.

3) The translate character (!) is used both to "count" spaces (see the a, b, and c items of 2, below) and to prevent an unattractive line split (see the last line of the segment.

```
qedx
a
.pl 84
.ll 80
.tr !
.fo 1 $$3-%$AG92$
.fo 2 $
.fo 3 $runoff sample  page$runoff sample page$runoff sample
page$
.brf
.bbl 1
.he "_____""_____"
.he 2 $
.he 3 XrunoffXXrunoffX
.he 4 8_____88_____8
.m1 6
.m2 3
.m3 2
.m4 6
.sp 7
.ce
RUNOFF SAMPLE PAGE
.sr rfsample %
.sp 2
.inl 0
```

The runoff command lets the user format his text segments through a variety of control words. The control words specify such things as:

```
.sp 2
.in 10
.un 5
```

1. Page length and line length (.pl and .ll respectively). If not specified by the user, these control word are given default values of:

```
.sp
.in +5
.li
.pl 66
.br
.li
.ll 65
.in -5
.sp
.un 5
```

2. Headers and footers, for all pages or for just odd numbered or just even numbered pages. The control words for headers and footers are as follows:

```
.sp
.in +5
.un 5
```

a.!!!Headers and footers on both odd and even numbered pages (.he and .fo)

```
.sp 1
.un 5
```

b.!!!Header and footers on just odd numbered pages (.oh and .of)

```
.sp 1
.un 5
```

c.!!!Headers and footers on just even numbered pages (.eh and .ef)

```
.sp 1
.in -5
.un 5
```

3. Margins that control vertical spacing in relation to the top of the page, headers, text, footers, and the bottom of the page. These margins are defined as follows:

```
.sp
.in +5
```

```
.un 5
a.   Between top of page and first header (.m1)
.spb
.un 5
.bp
b.   Between last header and first line of text (.m2)
.sp
.un 5
c.   Between last line of text and first footer (.m3)
.sp
.un 5
d.   Between the last footer and bottom of page (.m4)
.in -5
.sp
If  not  specified by  the  user, these  margins  are given
default values of:
.sp
.in +5
.nf
.li 4
.m1 4
.m2 2
.m3 2
.m4 4
.fi
.in 0
.sp 2
     To see the runoff segment  that created this page, see
the preceding pages.
\f
w example.runoff
q
<ready message>

rf example -wt
```

runoff (rf)

RUNOFF SAMPLE PAGE

The runoff command lets the user format his text segments through a variety of control words. The control words specify such things as:

1. Page length and line length (.pl and .ll respectively). If not specified by the user, these control word are given default values of:

```
.pl 66
.ll 65
```

2. Headers and footers, for all pages or for just odd numbered or just even numbered pages. The control words for headers and footers are as follows:

   a. Headers and footers on both odd and even numbered pages (.he and .fo) spf
   b. Header and footers on just odd numbered pages (.oh and .of) spf
   c. Headers and footers on just even numbered pages (.eh and .ef) spf

3. Margins that control vertical spacing in relation to the top of the page, headers, text, footers, and the bottom of the page. These margins are defined as follows:

   a. Between top of page and first header (.m1)

   b. Between last header and first line of text (.m2)

   c. Between last line of text and first footer (.m3)

   d. Between the last footer and bottom of page (.m4)

   If not specified by the user, these margins are given default values of:

```
.m1 4
.m2 2
.m3 2
.m4 4
```

To see the runoff segment that created this page, see the preceding pages.

runoff_abs (rfa)

SYNTAX AS A COMMAND:

rfa paths {rf_args} {ear_args} {dp_args} {-control_args}

FUNCTION: submits an absentee request to process text segments, creating an output segment for each text segment in the user's working directory.

ARGUMENTS:

paths
are the pathnames of segments to be processed by the runoff command. They need not specify the runoff suffix; however, this suffix must be the last component of each segment's name. If more than one pathname is specified, each segment is considered a separate runoff task.

rf_args
can be one or more control arguments accepted by the runoff command.

ear_args
can be one or more control arguments accepted by the enter_abs_request command, except -brief (-bf).

dp_args
can be one or more control arguments accepted by the dprint command, except -brief (-bf) and -truncate (-tc).

CONTROL ARGUMENTS:

-queue N, -q N
specifies in which priority queue the request is to be placed ($N \leq 3$). The output files are also dprinted in queue N. The default queue is 3.

-copy N, -cp N
specifies the number of copies of the segment to be dprinted ($N \leq 4$). The default is 1.

-hold, -hd
specifies that the output segments created by runoff should neither be queued for printing nor deleted. Each output segment is formatted for printing on a selectric-type terminal, with a 963 type ball, unless some other output form is specified by one of the runoff control arguments.

NOTES: The name of the output segment is the name of the text segment with the suffix runoff replaced by runout. The absentee process then uses the dprint command to queue each output segment for printing and deletion. Printing and deletion can be withheld if desired. If the -output_file (-of) control argument (one of those recognized by the enter_abs_request command) is not specified, the absentee process's output segment is placed in the user's working directory with the name path1.absout, where path1 is the first argument to the command.

Control arguments and pathnames can be mixed freely in the command line. All control arguments apply to all pathnames. An unrecognizable control argument causes the absentee request not to be submitted.

The -indent control argument applies to the segment produced by the runoff command and is not passed to the dprint command. If the -indent control argument is not specified, the default indentation is 20 spaces.

When doing several runoffs, it is more efficient to give several pathnames in one command, since only one process is set up with one command. Thus the cost of process initialization need be incurred only once.

See also the descriptions of enter_abs_request and runoff in this manual.

safety_sw_off (ssf)

SYNTAX AS A COMMAND:

ssf {paths}

FUNCTION: turns off the safety switch of a segment, directory, or multisegment file, thus permitting the entry to be deleted.

ARGUMENTS:

paths
are pathnames of segments, directories, and multisegment files. If one of the paths is -wd or -working_directory, the safety switch of the working directory is turned off. If no paths are specified, the safety switch of only the working directory is turned off. The star convention can be used.

NOTES: See "Directory Contents" in the MPM Reference Guide for a description of the safety switch. See also the description of safety_sw_on in this manual.

EXAMPLES:

The command line:

```
! ssf test.pl1 check.fortran
```

turns off the safety switch of the segments test.pl1 and check.fortran in the working directory.

The command line:

```
! ssf *.temp_dir -wd
```

turns off the safety switch of all segments, directories, and multisegment files (in the working directory) with a two-component name ending in temp_dir, and turns off the safety switch of the working directory.

The command alone, with no arguments:

```
! ssf
```

turns off the safety switch of the working directory.

safety_sw_on (ssn)

SYNTAX AS A COMMAND:

ssn {paths}

FUNCTION: turns on the safety switch of a segment, directory, or multisegment file, thus protecting it from deletion.

ARGUMENTS:

paths
are pathnames of segments, directories, or multisegment files. If one of the paths is -wd or -working_directory, the safety switch of the working directory is turned on. If no paths are specified, the safety switch of only the working directory is turned on. The star convention can be used.

NOTES: When the user invokes a command to delete a segment, directory, or multisegment file that has the safety switch turned on, a query is printed. The specified item is deleted only if the user answers "yes." See "Directory Contents" in the MPM Reference Guide for a description of the safety switch.

Since the delete_dir command already asks the user whether to delete the specified directories, no further query is made. That is, the delete_dir command functions in the same manner whether the safety switch is on or off. See the description of delete_dir in this manual.

EXAMPLES:

The command line:

```
! safety_sw_on *.alpha
```

turns on the safety switches of all segments, or multisegment files, and directories found in the working directory with two-component names ending in alpha.

The command alone, with no arguments:

```
! ssn
```

turns on the safety switch of the working directory.

SYNTAX AS A COMMAND:

save_on_disconnect

FUNCTION: reverses the effect of the no_save_on_disconnect command, re-enabling process preservation across hangups in the user's process.

NOTES: This command is only meaningful if process preservation was in effect for the process at login time, either by default or because the -save_on_disconnect control argument was specified on the login command line.

SYNTAX AS A COMMAND:

search strA strB

SYNTAX AS AN ACTIVE FUNCTION:

[search strA strB]

FUNCTION: returns the integer representing the character position in strA of the leftmost occurrence of any character contained in strB. If no character of strB occurs in strA, 0 is returned.

EXAMPLES: The following lines from an exec_com segment demonstrate how the search active function can be used to check that argument 1 does not contain either of the special characters used by the star convention. If the argument does not contain special characters, execution continues; if it does contain special characters, a message is printed and execution stops.

```
&if [nequal [search &1 *?] 0]
&then &goto continue
&print Star name not permitted:  &1
&quit
&label continue
```

The following interactions also illustrate the search active function.

```
! string [search "Paul, Mary;" ",;"]
5

! string [search "Harry" ",;"]
0
```

SYNTAX AS A COMMAND:

segs star_names {-control_arg}

SYNTAX AS AN ACTIVE FUNCTION:

[segs star_names {-control_arg}]

FUNCTION: returns the entrynames or absolute pathnames of segments that match one or more star names.

ARGUMENTS:

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per segment is returned; i.e., if a segment has more than one name that matches a star_name, only the first match found is returned.

Since each entryname (or pathname) returned by segments is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

EXAMPLES: The following interaction illustrates the use of the segments active function.

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w    0  empty_seg
  re     1  test
  r w    1  test.list
  r w    1  test.pl1
  re     1  prog
  r w    1  prog.list
  r w    1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w  513  prog.output
  r w  257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                >udd>Apple>Jones>temp_seg_2
  prog.temp1                >udd>Apple>Jones>temp_seg_1
  junk                      >udd>Apple>Jones>empty_seg

! string [segs *.pl1]
  prog.pl1 test.pl1
! string [segs *]
  prog test empty_seg
```

send_mail (sdm)

SYNTAX AS A COMMAND:

sdm {addresses} {-control_args}

FUNCTION: transmits a message to one or more recipients specified by Person_id.Project_id or mailbox pathname. It either accepts an input file or reads text from the terminal, then either sends the message or reads requests for editing, copying and sending. The message is automatically prefixed by a header, whose standard fields give the author(s), the intended recipients, and a brief summary of the contents. These fields are understood by the read_mail and print_mail commands, also in this manual. See "Extended Access" in the print_mail description for an explanation of mailbox access.

ARGUMENTS:

addresses
identifies the authors and the recipients of a message. An address refers to a mailbox, either by pathname or by owner. The permissible forms of address are:

STR
is any argument that does not begin with a minus sign (-). If it contains either of the characters > or <, it is interpreted as:

-mailbox STR

If STR does not contain > or <, it is interpreted as:

-user STR

-mailbox path, -mbx path
specifies a mailbox pathname. The mbx suffix is added to path if it is not present.

-user_id Person_id.Project_id
specifies a user as an address. The corresponding mailbox pathname is:

>udd>Project_id>Person_id>Person_id.mbx

This control argument is useful when a segment named Person_id.Project_id.mbx exists in the working directory.

Addresses can be qualified by the -comment control argument. The use of -comment does not affect the destination of the message.

-comment STR, -cmt STR
places STR in the header field associated with the address (including -mailbox and -user) directly preceding. If this control argument does not directly follow an address, STR is placed by itself in the header field most recently referred to. Comments are enclosed in parentheses. For example, the command line:

```
sdm Jones.Pubs -comment "send_mail person"
```

creates the header field:

```
To: Jones.Pubs (send_mail person)
```

Any addresses appearing on the command line before the first -cc, -from, -reply_to, or -to control argument are considered primary recipients of the message. (See the description of the -to control argument below.)

The -cc, -from, -reply_to, and -to control arguments apply to all subsequent addresses until the next of these control arguments is given. Any other intervening control arguments do not affect this interpretation.

For example, the sequence:

```
addr1 -from addr2 addr3 -cc addr4 -to addr5
```

causes addr1 and addr5 to be processed by -to, addr2 and addr3 to be processed by -from, and addr4 to be processed by -cc.

If conflicting control arguments (for instance, -header and -no_header) are specified, the last one takes effect.

CONTROL ARGUMENTS:
can be interspersed with the addresses and can be chosen from the following:

-abort
requests that send_mail not send the message unless it can be successfully delivered to all specified recipients. This is the default.

-acknowledge, -ack
requests that a message be sent to the user of send_mail by each recipient of the message after they have read the message via read_mail or print_mail. The user's name is placed in the Acknowledge-To header field.

-brief, -bf
suppresses printing of the message:

Mail delivered to ADDRESS.

when mail is sent.

-cc ADDRESSES
adds subsequent ADDRESSES as secondary recipients of the message. Mail is sent to these addresses when the send request is issued with no arguments. (See Requests below.) These addresses are placed in the cc header field. (See "Headers" above".) There are no secondary recipients by default.

-fill
reformats the text of the message according to "fill-on" and "align-left" modes in compose, before sending, entering the editor, or entering the request loop. The line length used is 72 unless specified by the -line_length control argument. If the -fill control argument is not specified, the message text is left unchanged.

-from ADDRESSES
adds subsequent addresses as authors of the message. These addresses are placed in the From header field, overriding the user's name placed there by default.

-header, -he
generates a message header. This is the default.

-in_reply_to STR, -irt STR
places STR in the In-Reply-To field of the header. This field is not present by default.

-input_file path, -if path
sends a message contained in a file. The file is sent without entering the request loop unless -request (-rq) or -request_loop (-rql) is specified. If -input_file is not specified, the user is prompted for the message text ("Message:").

-line_length N, -ll N
specifies a line length to be used when adjusting text via

-fill or the fill request. The default line length is 72.

-log
sends a copy of the message to the user's logbox, the mailbox named Person_id.sv.mbx in the home directory. The user's name is added to the cc header field. (See "Headers" above".)

-long, -lg
prints the "Mail delivered to ADDRESS" message when mail is sent. This is the default.

-message_id, -mid
adds a Message-ID field to the header, containing a unique identifier for the message. This is the default.

-no_abort
specifies that the message be sent to as many recipients as possible even if it cannot be sent to all specified recipients.

-no_acknowledge, -nack
prevents send_mail from requesting that each recipient of the message acknowledge reading the message. This is the default.

-no_fill, -nfi
sends the message as typed with no formatting adjustments. This is the default.

-no_header, -nhe
does not add the normal message header to the message. The only header fields added are those explicitly requested by control arguments or requests.

-no_log
specifies that a copy of the message is not to be sent to the user's logbox. This is the default.

-no_message_id, -nmid
specifies that a Message-ID field is not to be added to the header.

-no_prompt
does not prompt for request lines when inside the request loop. The default prompt is "send_mail(N):", where N is the recursion level if greater than one.

-no_request_loop, -nrql
sends the message without entering the request loop. If an error occurs while sending the message, the request loop is entered anyway. This is the default.

-no_subject, -nsj
specifies that a Subject field is not to be added to the header.

-prompt STR
sets the prompt for the request loop to the ioa_ control string STR. If STR is "", the user is not prompted.

-reply_to ADDRESSES, -rpt ADDRESSES
adds subsequent addresses to the Reply-To header field. This field is not present by default. (See "Headers" above".)

-request STR, -rq STR
executes STR as a line of requests after reading the message text from the appropriate source. If the quit (q) request is not included in STR, the request loop is entered after STR is executed.

-request_loop, -rql
enters the request loop before sending a file via -input_file (-if) or after "." is typed to terminate the input text. The default is to automatically send the message and quit.

-save path, -sv path
sends a copy of the message to the savebox path. The suffix .sv.mbx is added to path if it is not present. If the savebox does not exist, the user is asked whether to create it. The user's name with a comment containing the entry name of the savebox is added to the cc header field. (See "Headers" above".)

-subject STR, -sj STR
places STR in the Subject field of the header. If STR is "", no Subject field is created. If this control argument is not specified, the user is asked for a subject with the prompt "Subject:". A blank response causes the Subject field to be omitted.

-terminal_input, -ti
prompts the user for the message text ("Message:"). The user then types the message text terminated by a line consisting of a period ("."). This is the default.

-to ADDRESSES
adds subsequent ADDRESSES as primary recipients of the message. Addresses not preceded by any of the above control arguments are also primary recipients. All of these addresses are placed in the To header field (see "Headers" above). Mail is sent to them when the send request is issued with no arguments (see "Notes on Request Descriptions" below). There

are no primary recipients by default.

EXAMPLES OF GENERAL USE: To send mail to Person_id.Project_id, the user types:

sdm Person_id.Project_id

The send_mail command prompts "Subject:" and accepts a single line of text, then prompts "Message:" and accepts the text of the message to be sent. There are two ways to terminate the input text and send the message.

A line consisting of a period (".") sends the message as it was typed and returns to command level.

The character sequence \f invokes the qedx editor on the text that has been typed. The remainder of the line after \f is processed as editor requests. Unlike using qedx from command level, the user is not required to issue read (r) or write (w) requests to reflect changes in the message text. By default, when the qedx quit (q) request is typed, the send_mail request loop is entered.

The character sequence \fq in text input causes the send_mail request loop to be entered directly.

By default, each line of send_mail requests is prompted by the string "send_mail". If one or more invocations of send_mail have been interrupted by the current invocation, this prompt also includes a recursion level, for example "send_mail(3):".

The "send" request sends the messages to specified addresses or to the addresses specified on the send_mail command line.

The "save" request saves a copy of the message to be sent in a specified mailbox. The "log" request saves the message in a particular mailbox called the user's default logbox:

>udd>Project_id>Person_id>Person_id.sv.mbx

The "write" request saves a copy of the message in a specified printable ASCII file. These requests are useful for keeping track of correspondence.

The "qedx" request can be used to edit the message at any time prior to sending.

The "quit" request exits send_mail and returns to command level. If the message has not been sent anywhere, send_mail asks "Do you wish to send the message?" and accepts a yes or no reply.

EXAMPLES OF REQUEST LINE SYNTAX: A line beginning with ".." is treated as a special escape used to pass command lines directly to the standard command processor.

Other request lines have identical syntax to Multics command lines. Arguments containing spaces or other command language characters must be quoted, for example:

```
subject "food (not right away)"
```

Iteration is specified by means of parentheses, for example:

```
save (rdm sdm)_bugs
```

Semicolon is used to separate multiple requests on a line, for example:

```
log;send Jones.Mktg
```

Each request accepts a particular set of arguments. These are described in detail in an alphabetical list of requests at the end of this description.

NOTES ON REQUEST FUNCTIONS: Brackets in a request line invoke send_mail request functions, which operate like Multics active functions but belong to an internal repertoire. Request functions are listed along with requests below.

EXAMPLES OF SENDING BY PATHNAME:

To send to a mailbox by pathname, type:

```
! sdm path
```

if the pathname contains > or <, otherwise use the -mailbox control argument:

```
! sdm -mbx insult_mail
```

To send a message contained in a file, type:

```
! sdm DESTINATION -input_file path
```

If the -log control argument is specified, the message is automatically saved in the default logbox when it is sent.

The following is a sample dialogue with send_mail. The short paragraphs to the right of the dialogue are an explanation of the transactions. The user's Person_id.Project_id is Jones.Aeroproj. The user's input lines are indicated by an exclamation point.

```
! sdm Jones.Aeroproj Brown.Mktg -cc Smith.Mktg

   Subject:  !  New Model Biplanes
   Message:
!  We got a big shipment of the new     The command
!  modal biplanes                       automatically enters
!  I think these are a great            input mode and text
!  improvement over the                 is typed.
!  old traditional 707s, since they
!  are stabler
!  and carry many more people.
!  What do you think?
!
!  -- Dave Jones
!  \f                                   Enters qedx editor.
!   1,$s/modal/model/                   Make corrections in
!  /model/                              text using qedx
   We got a big shipment of the new     editing tools.
   model biplanes
!  .2s/stabler/more stable/p
   old traditonal 707s, since they
   are more stable
```

send_mail (sdm)

```
! q                                        Exit qedx and enter
                                           send_mail request loop.

   send_mail:  !  send                     Sends the message.
   Mail delivered to Jones.Aeroproj.       1 copy is put in the
                                           user's home directory.
   Mail delivered to Brown.Mktg.           1 copy is sent to Brown.
   Mail delivered to Smith.Mktg.           Smith.Mktg is a
                                           secondary redipient.

   send_mail:  !  quit
   r 1052 .791 4.132 109
```

NOTES ON REQUESTS: In the following list, send_mail requests are divided into five categories. The first, simple requests, includes several that are useful for the basic operation of send_mail. The second, editing requests, contains two requests for reformatting a message before it is sent. The third, copying requests, includes the save request and others used to move messages around. The fourth, header requests, contains operations on the header fields of a message. These header fields are described later in this section. The fifth category lists several advanced requests.

Most requests have short names that can be used instead to save typing. A more complete description of each request, including its calling sequence, appears at the end of this section.

LIST OF SIMPLE REQUESTS:

?
prints a summary of the available send_mail requests.

.
identifies the current state of send_mail.

help
prints information on how to use send_mail.

list (ls)
prints a short summary of the message.

print (pr)
prints the text of the message.

print_header (prhe)
   prints the header of the message.

quit (q)
   exits from send_mail.

send
   sends the message.

LIST OF EDITING REQUESTS:

fill (fi)
   reformats the message text as in "fill" mode.

qedx (qx)
   invokes the qedx editor on the message text.

LIST OF COPYING REQUESTS:

append
   copies the message to the end of an existing ASCII file.

copy (cp)
   copies the message to specified mailboxes.

log
   copies the message to the default logbox.

preface
   copies the message to the beginning of an existing ASCII file.

save (sv)
   copies the message to specified saveboxes.

write (w)
   copies the message to an old or new ASCII file.

LIST OF HEADER REQUESTS:

cc
   adds to the cc: header field or prints its contents.

from
   adds to the From: header field or prints its contents.

in_reply_to (irt)
   replaces the In-Reply-To: header field or prints its contents.

message_id (mid)
prints the Message-Id: header field.

remove (rm)
removes entries from specified header fields.

reply_to (rpt)
adds to the Reply-To: header field or prints its contents.

subject (sj)
replaces the Subject: header field or prints its contents.

to
adds to the To: header field or prints its contents.

LIST OF ADVANCED REQUESTS:

apply (ap)
applies a Multics command to a file containing the text of the message.

execute (e)
executes a Multics command line after expanding send_mail request functions in the line.

..
passes the command line directly to the command processor.

NOTES ON HEADERS: Messages created by send_mail begin with a message header. This header contains information used by the read_mail, print_mail and send_mail commands to facilitate listing, saving, forwarding, and replying to messages.

The message header is separated from the text by one or more blank lines. It consists of one or more fields. Each field consists of an identifier, a colon, and one or more entries. Multiple entries are separated by commas. If a field is too long to fit on one line, it is continued on successive lines. (See the following example.)

A sample header is:

```
Date:     25 May 1978 14:54-EDT
From:     Jones.Mktg
Subject:  headers in send_mail
To:       Smith.Mktg Brown.Mktg
          "{mbx >udd>Pubs>Doe>mlsys.sv}"
cc:       Miller.Pubs
```

NOTES ON ADDRESSES IN HEADERS: Several header fields contain lists of addresses. There are two distinct formats used for an address in a header field:

Person_id.Project_id
specifies a user identifier. It is generated by the -user form of address or a STR address that is not a mailbox pathname.

{mbx path}
specifies a mailbox pathname. The mbx suffix is not included in path.

In addition, any of the above can be followed by a comment in parentheses. This comment is supplied by the -comment control argument. For example:

```
{mbx >udd>Mktg>Jones>mlsys} (Mail System Developers)
```

HEADER FIELDS

The standard header fields are listed below in the order they appear in a message. The Date and From fields are always present. The others are optional.

Redistributed-Date
Redistributed-By
Redistributed-To
specify information about the forwarding of the message. These fields are added only by forward request of read_mail.

Date
contains the date and time when the message was first transmitted, for example:

```
5 June 1978 23:45 edt
```

From
contains a list of addresses identifying the author(s) of the message. If -from is not specified and the from request is not issued, this field names the user who invoked send_mail.

Subject
contains a brief description of the contents of the message. This field is present only if a non-null string is given to the -subject control argument, a non-blank line is given to the "Subject:" prompt, or the subject request is issued with one or more arguments.

Sender
contains the address of the user who actually sent the message. This field is created only if the -from control argument or the from request is used.

Reply-To
contains a list of addresses to which a reply should be sent. When present, read_mail uses the addresses given in this field instead of the addresses in the From field as recipients of a reply request. This field is created only if the -reply_to control argument or the reply_to request is used.

To
contains a list of addresses naming the primary recipients of the message. The send request when invoked with no arguments sends the message to all of these addresses. This field is created only if primary recipients are specified on the command line or if the to request is used.

cc
contains a list of addresses naming the secondary recipients of the message. The send request when invoked with no arguments sends the message to these addresses. This field is created only if the -cc control argument or the cc request is used, or if either -save or -log is specified, in which case a cc field entry is created for the person creating the message, with a comment containing the name of the mailbox being saved into, as in:

```
Smith.Mktg (mail_system.sv)
```

Acknowledge-To
contains an address to which an acknowledgement message should be sent after someone receiving this piece of mail reads it. This field is generated only when the -acknowledge control argument to send_mail is used.

Message-ID
contains a unique character string identifier for the message. This string is generated by send_mail, and is useful when reading messages to detect multiple copies of the same message.

In-Reply-To
contains an ASCII string describing the message to which this message is a reply. This field is present only if the -in_reply_to control argument or the in_reply_to request is used.

List of Requests

The available send_mail requests are:

```
?
.
..
append path
apply {-control_arg} STRs,
   ap {-control_arg} STRs
cc {ADDRESSES}
copy path, cp path
execute STRs, e STRs
fill {-control_arg}, fi {-control_arg}
from {ADDRESSES}
help {STR}
in_reply_to {STRs}, irt {STRs}
log
message_id, mid
preface path
print {-control_arg}, pr {-control_arg}
print_header {-control_arg}, prhe {-control_arg}
qedx {-control_arg}, qx {-control_arg}
quit {-control_arg}, q {-control_arg}
remove {ADDRESSES} {-control_args},
   rm {ADDRESSES} {-control_args}
reply_to {ADDRESSES}, rpt {ADDRESSES}
save path, sv path
send {ADDRESSES} {-control_args}
subject {STRs}, sj {STRs}
to {ADDRESSES}
write path, w path
```

The available request functions are:

execute STRs, e STRs
subject, sj

Request Descriptions

?
prints a summary of the available send_mail requests.

.
prints a line identifying send_mail and the current state of the message being created, as in:

```
send_mail 3.6:  23 lines (modified)  Subject:  Zoot Suits
```

The word "modified" indicates that the message has been changed since the last use of the send request. The string "send_mail 3.6" gives the version number of send_mail. If the current recursion level is greater than one, it is given in parentheses, for example:

```
send_mail 3.6 (level 2):  5 lines:
```

..
passes the command line directly to the command processor.

append path
appends the message complete with header to the end of an existing ASCII file. The suffix mail is added to path if it is not present. If the file does not exist, the user is asked whether to create it.

apply {-control_arg} STRs, ap {-control_arg} STRs
places the message in a temporary segment in the process directory, then concatenates STRs with intervening spaces and appends the pathname of the temporary segment. This concatenated command line is passed to the Multics command processor. When the command line has completed, the message in send_mail is replaced with the contents of the temporary segment.

If the -header (-he) control argument is specified, this request operates on both the header and the text. If -no_header (-nhe) or no control argument is specified, it operates on the text only. Control arguments must precede the STRs.

If the message header is changed by the command line, send_mail parses it and updates the lists or primary and secondary recipients, authors, reply addresses, etc.

This request can be used to edit the message with an arbitrary editor, for example:

```
apply teco
```

which edits the message using the teco editor command.

cc {ADDRESSES}
adds any addresses specified to the list of secondary recipients of the message. Mail is sent to these addresses when a subsequent send request is issued with no arguments. The addresses are added to the cc field, which is created if necessary.

If no addresses are specified, the secondary recipients of the message are listed.

copy path, cp path
copies the message into the mailbox designated by path. The mbx suffix is added to path if it is not present.

execute STRs, e STRs
passes the concatenation of STRs with intervening spaces to the Multics command processor. This request is different from ".." because it is first parsed as a send_mail request line. The send_mail request interpreter expands send_mail request functions, strips quotes, and performs iteration before the line is passed on to the command processor.

The execute request function can be used to invoke a Multics active function from within send_mail. The request line:

```
save [execute date]
```

saves the message in a savebox whose name is the current date.

fill {-control_arg}, fi {-control_arg}
reformats the message text according to "fill-on" and "align-left" modes in compose. If the -line_length N (-ll N) control argument is specified, N is used as the line length.

Otherwise, the value specified to the -line_length control argument on the send_mail command line is used, or 72 if -line_length was not specified.

from {ADDRESSES}
adds addresses to the list of authors of the message if any addresses are specified. The addresses are added to the From field of the header.

If no addresses are specified, the authors of the message are listed.

help {STR}
prints information about the send_mail command. If specified, STR is the name of a send_mail request or one of the topics "requests", "control_args", and "changes". If STR is "*", the available send_mail topics are listed. If STR is not specified, introductory information on the use of send_mail is printed followed by a list of topics.

in_reply_to {STRs}, irt {STRs}
replaces the In-Reply-To field of the message (if any) with the concatenation of the STRs with intervening spaces. If no STRs are specified, it prints the contents of the In-Reply-To field.

log
saves a copy of the message in the user's logbox (Person_id.sv.mbx). This request creates the logbox if it does not already exist.

message_id, mid
prints the Message-ID field of this message, creating the field if necessary.

preface path
operates the same as append, but inserts the message at the beginning of the ASCII file.

print {-control_arg}, pr {-control_arg}
prints the message. If -briefheader (-bfhe) or no control argument is specified, a short summary of the message and the text are printed. If the -header (-he) control argument is specified, both the header and the text are printed. If the -no_header (-nhe) control argument is specified, only the text is printed.

print_header {-control_arg}, prhe {-control_arg}
prints the header of the message. If -long (-lg) or no control argument is specified, the entire header is printed as

it will appear in the message when the message is sent. If the -brief (-bf) control argument is specified, a summary of the message is printed. The format of the summary is:

```
(N lines in text):
Subject: STR
To: ADDRESSES
cc: ADDRESSES
```

qedx {-control_arg}, qx {-control_arg}
invokes the qedx editor to modify the message. If the -header (-he) control argument is specified, both the header and the text are edited. If -no_header (-nhe) or no control argument is specified, only the text is edited.

The qedx w (write) request is not necessary to reflect changes in the message to send_mail. The editor request line 1,$dr can be used to restore the original text.

If the message header is changed during editing, send_mail parses it when qedx returns and updates the lists of primary and secondary recipients, authors, reply addresses, etc. Requests to send_mail (subject, reply_to, etc.) are recommended over qedx requests for changing header fields.

quit {-control_arg}, q {-control_arg}
exits the send_mail command. If the message has not been sent, or if it has been modified by qedx since it was last sent, and if the -force (-fc) control argument is not specified, the user is queried before exiting.

remove {ADDRESSES} {-control_args}
rm {ADDRESSES} {-control_args}
must have at least one ADDRESS or one control argument specified.

This request deletes from the list of primary and/or secondary recipients any ADDRESSES appearing before the first -cc, -from, -reply_to, or -to control argument.

ADDRESSES appearing after -cc, -from, -reply_to, or -to are deleted from the corresponding field. If any of these control arguments is followed by -all or -a, the corresponding field is deleted in its entirety.

If the -in_reply_to (-irt), -message_id (-mid), or -subject (-sj) control argument is specified, delete the corresponding field in its entirety. The presence of this class of control argument does not affect the interaction of ADDRESSES and the other control arguments.

For example, the request line:

```
rm Jones.Mktg -sj -from Smith.Pubs Brown.Pubs -to Doe.Mktg
```

removes Jones.Mktg from both the primary and secondary recipient lists, deletes the Subject field from the message, removes Smith.Pubs and Brown.Pubs from the list of authors of the message and removes Doe.Mktg from the primary recipient list only.

reply_to {ADDRESSES}, rpt {ADDRESSES}
adds address, if specified, to the list of addresses to use when sending a reply to this message. These addresses are also appended to the Reply-To field of the header, which is created if neccssary.

If no addresses are specified, the addresses to receive replies are listed.

save path, sv path
saves a copy of the message in the indicated savebox. The suffix sv.mbx is added to path if not already present. If the savebox does not exist, the user is asked whether to create it.

send {ADDRESSES} {-control_args}
transmits the message to the primary and secondary recipients if no arguments are specified.

If any ADDRESSES are specified, the message is transmitted to these ADDRESSES without adding them to the message header.

The control arguments -log and -save path (-sv path) cause a copy of the message to be placed in the logbox and specified savebox, respectively. Other -control_args compatible with the send request are -abort, -acknowledge (-ack), -brief (-bf), -header (-he), -long (-lg), -message_id (-mid), -no_abort, -no_acknowledge (-nack), -no_header (-nhe), -no_message_id (-nmid), -no_notify (-nnt), and -notify (-nt).

subject {STRs}, sj {STRs}
replaces the Subject field of the message (if any) with the concatenation of the STRs with intervening spaces. If no STRs are specified, the contents of the Subject field are printed instead.

As a request function, subject returns the contents of the Subject field as a single, quoted string.

send_mail (sdm)

to {ADDRESSES}
adds ADDRESSES, if specified, to the list of primary recipients of the message. Mail is sent to these addresses when a subsequent send request is issued with no arguments. The addresses are added to the To field of the header, which is created if necessary.

If no ADDRESSES are specified, the primary recipients of the message are listed.

write path {-control_args}
appends the message complete with header to an ASCII file. The suffix mail is added to path if it is not present. The segment is created if necessary. The extend and truncate (-tc) control arguments accepted by the file_output command can be used here. The default is -extend.

SYNTAX AS A COMMAND:

```
   sm Person_id.Project_id {message}
or:
   sm -pathname path {message}
```

FUNCTION: sends messages (one or more, always sent one line at a time) to a given user on a given project.

ARGUMENTS:

Person_id
is the registered name of the recipient.

Project_id
is the name of the recipient's project.

message
is an optional string that can be up to 132 characters long. If message is missing from the command line, send_message types "Input." and accepts lines that it sends, one line at a time, with each newline character. In this case, input is terminated by a line consisting solely of a period.

CONTROL ARGUMENT:

-pathname path, -pn path
causes messages to be sent to a mailbox specified by pathname. The mbx suffix is assumed.

NOTES: For a description of the mailbox, refer to accept_messages and print_mail in this manual.

If the recipient is accepting messages (see accept_messages and defer_messages in this manual), send_message immediately prints each message on the recipient's terminal.

Parentheses, quotes, brackets, and semicolons in the command line have their usual command language interpretation.

The user can receive messages while in send_message input mode, and can therefore carry on a conversation with a single invocation of the command.

EXAMPLES:

If WJones on the Alpha project sends the following to RTSmith on the Beta project by using the command line:

```
! sm RTSmith.Beta need access to your lsg command
```

the message prints on RTSmith's terminal (if RTSmith is accepting messages) as follows:

```
From WJones.Alpha 04/20/76 1200.6 mst Tue: need access to
   your lsg command
```

The command line:

```
! sm Person_id.Project_id testing complete; installation this
week
```

sends:

```
testing complete
```

and prints an appropriate error message (e.g., "Segment installation not found.") because the characters typed after the semicolon are interpreted as another command line.

The command line:

```
! sm Person_id.Project_id so long (for now)
```

sends two lines:

```
so long for
so long now
```

In both of the above examples, the sender's intended message would have been sent if it had been enclosed in quotes (e.g., "so long (for now)").

send_message_acknowledge (sma)

SYNTAX AS A COMMAND:

```
   sma Person_id.Project_id {message}
or:
   sma -pathname path {message}
```

FUNCTION: operates like send_message and requests that the recipient's process return an acknowledgement when the message is read.

ARGUMENTS:

Person_id
is the registered name of the recipient.

Project_id
is the name of the recipient's project.

message
is an optional string that can be up to 132 characters long. If message is missing from the command line, send_message_acknowledge types "Input." and accepts lines that it sends, one line at a time, with each newline character. In this case, input is terminated by a line consisting solely of a period.

CONTROL ARGUMENTS:

-pathname path, -pn path
causes messages to be sent to a mailbox specified by pathname. The mbx suffix is assumed.

send_message_acknowledge (sma)

NOTES:

The acknowledgement says:

From <Person_id.Project_id> <time>: Acknowledged.

if the message is read right away (recipient is accepting messages), or:

From <Person_id.Project_id> <time>:
Acknowledge message of <sent_time>

if the message is read later.

If the recipient has insufficient access to send an acknowledgement, none is sent. No error message is printed.

Notes and examples documented for send_message apply to send_message_acknowledge.

send_message_express (smx)

SYNTAX AS A COMMAND:

```
smx Person_id.Project_id {message}
or:
smx -pathname path {message}
```

FUNCTION: operates like send_message but adds the message to the recipient's mailbox only if the recipient will see it immediately (i.e., is currently accepting messages).

ARGUMENTS:

Person_id
is the registered name of the recipient.

Project_id
is the name of the recipient's project.

message
is an optional string that can be up to 132 characters long. If message is missing from the command line, send_message_express types "Input." and accepts lines that it sends, one line at a time, with each newline character. In this case, input is terminated by a line consisting solely of a period.

CONTROL ARGUMENTS:

-pathname path, -pn path
causes messages to be sent to a mailbox specified by pathname. The mbx suffix is assumed.

NOTES: Notes and examples documented for send_message apply to send_message_express.

send_message_silent (sms)

SYNTAX AS A COMMAND:

```
sms Person_id.Project_id {message}
or:
sms -pathname path {message}
```

FUNCTION: operates like send_message but does not print an error message if the message cannot be sent or will not be received immediately.

ARGUMENTS:

Person_id
is the registered name of the recipient.

Project_id
is the name of the recipient's project.

message
is an optional string that can be up to 132 characters long. If message is missing from the command line, send_message_silent types "Input." and accepts lines that it sends, one line at a time, with each newline character. In this case, input is terminated by a line consisting solely of a period.

CONTROL ARGUMENTS:

-pathname path, -pn path
causes messages to be sent to a mailbox specified by pathname. The mbx suffix is assumed.

NOTES: Notes and examples documented for send_message apply to send_message_silent.

SYNTAX AS A COMMAND:

sa path mode1 {User_id1 ... moden User_idn} {-control_args}

FUNCTION: manipulates the access control lists (ACLs) of segments, multisegment files, and directories. See "Access Control" in the MPM Reference Guide for a discussion of ACLs.

ARGUMENTS:

path
is the pathname of a segment, multisegment file, or directory. If it is -wd or -working_dir, the working directory is assumed. The star convention can be used and applies to either segments and multisegment files or directories, depending on the type of mode specified in mode1.

modei
is a valid access mode. For segments or multisegment files, any or all of the letters rew; for directories, any or all of the letters sma with the requirement that if modify is present, status must also be present. Use null, "n" or "" to specify null access.

User_idi
is an access control name that must be of the form Person_id.Project_id.tag. All ACL entries with matching names receive the mode modei. (For a description of the matching strategy, see "Notes" below.) If no match is found and all three components are present, an entry is added to the ACL. If the last modei has no User_id following it, the Person_id of the user and current Project_id are assumed.

CONTROL ARGUMENTS:

-chase
causes links to be chased when using the star convention. (Links are always chased when path is not a starname.)

-no_chase
causes links to not be chased when using the star convention. This is the default.

-brief, -bf
suppresses error messages of the form "No match for User_id on ACL of <path>", where User_id does not specify all components.

Either of the following control arguments can be specified to resolve an ambiguous choice between segments and directories that occur only when mode*i* is null and the star convention is used in path--

-directory, -dr
specifies that only directories are affected.

-segment, -sm
specifies that only segments and multisegment files are affected. This is the default.

ACCESS REQUIRED: The user needs modify permission on the containing directory.

NOTES: The arguments are processed from left to right. Therefore, the effect of a particular pair of arguments can be changed by a later pair of arguments.

The strategy for matching an access control name argument is defined by three rules--

1) A literal component, including "*", matches only a component of the same name.

2) A missing component not delimited by a period is treated the same as a literal "*" (e.g., "*.Multics" is treated as "*.Multics.*"). Missing components on the left must be delimited by periods.

3) A missing component delimited by a period matches any component.

EXAMPLES:

..* matches only the literal ACL entry "*.*.*".

Multics matches only the ACL entry "Multics.*.*". (The absence of a leading period makes Multics the first component.)

JRSmith.. matches any ACL entry with a first component of JRSmith.

.. matches any ACL entry.

. matches any ACL entry with a last component of *.

"" (null string) matches any ACL entry ending in ".*.*".

EXAMPLES:

The command line:

```
! set_acl *.pl1 rew *
```

adds to the ACL of every segment in the working directory that has a two-component name with a second component of pl1 an entry with mode rew to *.*.* (everyone) if that entry does not exist; otherwise it changes the mode of the *.*.* entry to rew.

The command line:

```
! sa -wd sm Jones.Faculty
```

adds to the ACL of the working directory an entry with mode sm for Jones.Faculty.* if that entry does not exist; otherwise it changes the mode of the Jones.Faculty.* entry to sm.

The command line:

```
! sa alpha.basic rew .Faculty. r Jones.Faculty.
```

changes the mode of every entry on the ACL of alpha.basic with a middle component of Faculty to rew, then changes the mode of every entry that starts with Jones.Faculty to r.

set_bit_count (sbc)

SYNTAX AS A COMMAND:

sbc path1 count1 {... pathn countn}

FUNCTION: sets a specified bit count on a specified segment or multisegment file, and changes the bit count author for that entry to be the user who invoked the command.

ARGUMENTS:

pathi
is the pathname of a segment or multisegment file. If pathi is a link, the bit count of the entry linked to is set.

counti
is the bit count, in decimal, desired for pathi.

ACCESS REQUIRED: The user must have write access on the entry whose bit count is to be set.

NOTES: Setting the bit count on a directory is permitted, but several system modules then regard the directory as a multisegment file.

See "Directory Contents" in the MPM Reference Guide for a description of the bit count of an entry.

SYNTAX AS A COMMAND:

```
set_cc fileNN {-control_arg}
```

FUNCTION: sets the carriage control transformation for a specified FORTRAN formatted file either on or off.

ARGUMENTS:

fileNN
is the name of a FORTRAN file in the range of file01 to file99. If fileNN is out of range, an error message is printed.

CONTROL ARGUMENTS:

-off
turns the carriage control transformation off for the specified FORTRAN file.

-on
turns the carriage control transformation on for the specified FORTRAN file.

NOTES: When the transformation is on, the first character of each line written to the file is changed to a control character in accordance with the following table:

Character	Resulting Control Character
0	Newline 012 (double space)
1	Newpage 014 (page eject)
blank	None (single space)
+	The previous line and the current line are written as a single line split by a carriage return character. This causes the second line to overprint the first. If the file is attached to a terminal, the + is ignored. The result is a single space between lines.

When the transformation is off, the first character is not changed. The default is off for all files except for file06 and file42, for which the default is on.

EXAMPLES: To turn off the carriage control transformation for the FORTRAN file named file06, type the command line:

```
set_cc file06 -off
```

set_fortran_common (sfc)

SYNTAX AS A COMMAND:

sfc paths {-control_arg}

FUNCTION: initializes common storage for a FORTRAN run. Due to the use of dynamic linking in the Multics system, if the first program to reference a common block is not compiled or bound with the block data subprogram that initializes the common block, the common block may not be successfully initialized. The set_fortran_common command allows the user to specify the segments containing the block data subprograms prior to the run.

ARGUMENTS:

paths
is a list of pathnames of segments containing block data subprograms that initialize common.

CONTROL ARGUMENTS:

-long, -lg
specifies that a message is to be printed if a referenced common block has already been allocated.

NOTES: This command is most convenient in the run exec_com used to initialize the environment for a FORTRAN run.

Any common blocks referenced in the specified segments are allocated (if necessary) and initialized. If no initialization information is associated with the referenced common block, it is initialized to binary zeroes. If a common block was previously allocated, it is effectively deleted and reinitialized.

set_iacl_dir (sid)

SYNTAX AS A COMMAND:

sid path mode1 {User_id1 ... moden User_idn} {-control_arg}

FUNCTION: manipulates the directory initial access control lists (directory initial ACLs) of directories.

ARGUMENTS:

path
specifies the directory whose directory initial ACL is to be changed. If it is -wd or -working_directory, the directory initial ACL for the working directory is changed. The star convention can be used.

modei
is the mode associated with User_idi. It can consist of any or all of the letters sma (status, modify, append) except that if "m" is given, "s" must also be given. The strings "null", "n", and "" specifically deny access to User_idi.

User_idi
is an access control name of the form Person_id.Project_id.tag. If one or more of the components is missing, all entries that match User_idi are changed to modei. (For a description of the matching strategy, refer to set_acl in this manual.) If all three components are present, the directory initial ACL entry with that name is changed to modei, or one is added if none exists. If the last modei has no User_idi following it, the user's name and project are assumed.

CONTROL ARGUMENTS:

-ring N, -rg N
identifies the ring number whose directory initial ACL should be set. It can appear anywhere on the line, except between a mode and its associated User_id, and affects the whole line. If present, it must be followed by N (where user's ring $\leq$ N $\leq$ 7). If this control argument is omitted, the user's ring is assumed.

NOTES: A directory initial ACL contains the ACL entries to be placed on directories created in the specified directory. For a discussion of initial ACLs, see "Access Control" in the MPM Reference Guide.

set_iacl_dir (sid)

EXAMPLES:

The command line:

```
sid listings sm * -ring 5
```

adds to the ring 5 directory initial ACL of the listings directory an entry with the mode sm for *.*.* (everyone) if that entry does not exist; otherwise it changes the mode of the *.*.* entry to sm.

The command line:

```
sid -wd sa Jones..
```

changes the mode of all entries with Person_id Jones in the directory initial ACL of the working directory to sa. If no such entries exist, an error message is printed.

set_iacl_seg (sis)

SYNTAX AS A COMMAND:

sis path mode1 {User_id1 ... moden User_idn} {-control_arg}

FUNCTION: manipulates the segment initial access control lists (segment initial ACLs) of directories.

ARGUMENTS:

path
specifies a directory whose segment initial ACL is to be changed. If it is -wd or -working_dir, the segment initial ACL for the working directory is changed. The star convention can be used.

modei
is the mode associated with User_idi. It can consist of any or all of the letters rew (read, execute, write). The strings "null", "n", and "" specifically deny access to User_idi.

User_idi
is an access control name of the form Person_id.Project_id.tag. If one or more of the components is missing, all ACL entries that match User_idi are changed to modei. (For a description of the matching strategy, refer to set_acl in this manual.) If all three components are present, the ACL entry with that name is changed to modei, or one is added if none exists.

CONTROL ARGUMENTS:

-ring N, -rg N
identifies the ring number whose segment initial ACL should be set. It can appear anywhere on the line except between a mode and its associated User_id, and affects the whole line. If present it must be followed by N (where user's ring $\leq$ N $\leq$ 7). If this control argument is omitted, the user's ring is assumed.

NOTES: A segment initial ACL contains the ACL entries to be placed on segments created in the specified directory. For a discussion of initial ACLs see "Access Control" in the MPM Reference Guide.

set_iacl_seg (sis)

EXAMPLES:

The command line:

```
! set_iacl_seg test rew *
```

adds to the segment initial ACL in the test directory an entry with mode rew for *.*.* (everyone) if that entry does not exist; otherwise it changes the mode of the *.*.* entry to rew.

The command line:

```
! sis -wd re Jones.. -rg 5
```

changes the mode of all entries with Person_id Jones in the ring 5 segment initial ACL of the working directory to re. If no such entries exist, an error message is printed.

set_search_paths (ssp)

SYNTAX AS A COMMAND:

ssp search_list {search_paths} {-control_arg}

FUNCTION: allows a user to replace the search paths contained in a specified search list.

ARGUMENTS:

search_list
is the name of a search list. If this search list does not exist, it is created. A warning message is printed if a search list is created and it is not system defined.

search_pathi
is a search path to be added to the specified search list. The search paths are added in the order in which they are specified in the command line. The search path can be an absolute or relative pathname or a keyword. (For a list of acceptable keywords see add_search_paths in this manual.) If no search paths are specified, then the specified search list is set as if it were being initialized for the first time in the user's process.

CONTROL ARGUMENTS:

-brief, -bf
suppresses a warning message for the creation of a search list not defined by the system.

NOTES: The specified search list is replaced by the specified search paths. It is an error to create a new empty search list.

For a complete list of the search facility commands, see the add_search_paths command description in this manual.

set_search_rules (ssr)

SYNTAX AS A COMMAND:

```
set_search_rules {path}
```

FUNCTION: sets the dynamic linking search rules of the user to suit individual needs with only minor restrictions.

ARGUMENTS:

path
is the pathname of a segment containing the ASCII representation of search rules. Search rules are absolute pathnames and any of the keywords listed below in "List of Keywords", one search rule per line. If path is not specified, the search rules are reset to the default search rules.

LIST OF KEYWORDS:

initiated_segments
checks the already initiated segments.

referencing_dir
searches the containing directory of the segment making the reference.

working_dir
searches the working directory.

home_dir
searches the home directory.

process_dir
searches the process directory.

site-defined keywords
expand into one or more directory pathnames. See the get_system_search_rules command for an explanation of the values of these keywords. The "default" keyword can be used to obtain the site-defined default rules.

set_search_rules (ssr)

NOTES: A maximum of 21 rules is allowed. Leading and trailing blanks are allowed, but embedded blanks are not allowed.

If the user decides not to include the system libraries in the search rules, many standard commands cannot be found.

See also the descriptions of the print_search_rules, get_system_search_rules, add_search_rules, and delete_search_rules commands.

SYNTAX AS A COMMAND:

severity command_name

SYNTAX AS AN ACTIVE FUNCTION:

[severity command_name]

FUNCTION: returns a number representing the severity of the most recent translation/invocation of the specified command. The meaning of the returned value is explained under "Severity" in the description of the specified command.

ARGUMENTS:

command_name
is either pl1, map355, or fortran (see "Notes" below).

NOTES: The fortran command only supports the severity active function if the site is using the new FORTRAN compiler as its "standard" FORTRAN compiler.

SYNTAX AS A COMMAND:

slave

FUNCTION: changes the service type of the channel from login to slave for the duration of the connection.

NOTES: The slave command enables a privileged process to request the answering service to assign the channel to it, and then attach it. Refer to the description of the dial_manager_ subroutine in the MPM Subsystem Writers' Guide for an explanation of the mechanism for requesting channels from the answering service.

SYNTAX AS A COMMAND:

sort

FUNCTION: provides a generalized file sorting capability, which is specialized for execution by user-supplied parameters.

NOTES: The basic function of the sort is to read one or more input files of unordered records, sort these records according to the values of one or more key fields, and write a single file of ordered (or "ranked") records.

For a detailed description of the sort command, refer to the Multics Sort/Merge Reference Manual, Order No. AW32.

SYNTAX AS A COMMAND:

ss path {-control_args}

FUNCTION: orders the contents of a segment according to the ASCII collating sequence.

ARGUMENTS:

path
specifies the pathname of an input segment. The star convention is NOT allowed.

CONTROL ARGUMENTS:

-all, -a
makes the primary (and only) sort field the entire sort unit; i.e., the entire sort unit is considered when sorting. This is the default mode of operation.

-ascending, -asc
makes the sort in ascending order, according to the ASCII collating sequence. This is the default mode of operation.

-block N, -bk N
makes the sort unit a block of N strings where N must be a positive integer. The default for N is 1 (see "Examples" below).

-delimiter STR, -dm STR
uses STR concatenated with a newline character as the string delimiter. The character STR can be any sequence of ASCII characters. The default is a single newline character (see "Examples" below).

-descending, -dsc
makes the sort in descending order, according to the ASCII collating sequence. The use of this control argument is incompatible with the use of the -ascending control argument.

-field field_spec, -fl field_spec
specifies the field (or fields) when sorting within a sort unit. The field_spec string consists of positive integers separated by spaces:
S1 L1 S2 L2 ... Sn Ln
notice that the arguments must be specified in pairs. The first argument of the pair (represented by "S") is the start

position of the field in the sort unit (e.g., 1 if the field begins at the first character). The second argument (represented by "L") is the length of the field, in characters. The first pair of field specifications defines the primary sort field; the second pair defines the secondary sort field; and so forth. The use of this control argument is incompatible with the -all or -ordered_field control arguments (see "Notes" below).

-ordered_field field_spec, -ofl field_spec
specifies a sort with independent ordering of the fields, i.e., mixed ascending and descending fields. The field_spec arguments must be specified in threes:

S1 L1 O1 S2 L2 O2 ... Sn Ln On

The first and second arguments are identical to those given with the -field control argument (i.e., positive integers specifying the start position and length of the sort field). The third argument (represented by "O") is either the string "asc" to indicate an ascending field or "dsc" to indicate a descending field. Use of this control argument is incompatible with the -ascending, -descending, or -field control arguments.

-replace, -rp
replaces the original contents of the input segment with the sorted units. This is the default.

-segment path, -sm path
places the sorted units in a segment whose pathname is path. The use of this control argument is incompatible with the use of the -replace control argument.

-unique, -uq
deletes duplicate sort units from the sorted results. The default is to retain any duplicated units (see "Notes" below).

NOTES: Using the control arguments, the segment is broken down into separate sort units, which are strings or blocks of strings. A string can comprise one or more lines. These sort units are then sorted, and the ordered units either replace the original segment or are placed in a new segment.

If the sort_seg command is invoked without any control_args, the -replace, -ascending, -all, and -delimiter control arguments are assumed, and the default delimiter of a newline character is used. That is, the sort_seg command, when

invoked with path as the only argument, sorts the lines of that segment in ascending ASCII collating sequence, replacing the original segment with the sorted result.

The start position of a sort field is calculated relative to the beginning of a sort unit. If the blocking factor is N = 1, the start position is calculated relative to the beginning of a string. If the blocking factor is N > 1, the start position is calculated relative to the beginning of the first string of a block. When calculating field specifications within a sort unit of N > 1 strings (blocking factor N > 1), string delimiters internal to the sort unit should not be considered (see "Examples" below).

Sort fields/units of unequal length are compared by assuming the shorter field/unit to be padded on the right with blanks, immediately following the rightmost character. The string delimiter is never considered when padding (see "Examples" below).

If characters are detected in the input segment following the final delimited sort unit, they are ignored for the purposes of sorting, but appear in the sorted output immediately following the final delimited sort unit. An error message specifies the location of the first nondelimited character.

A maximum of 262,143 units can be sorted. The sort is stable, i.e., duplicate units appear in the same order in the sorted segment as in the original segment.

The input segment is sorted using temporary segments in the process directory. If the -segment control argument is specified, and path is the pathname of an already existing segment, its contents are destroyed upon beginning the sort. If the sorted results are to replace the original contents of the input file, that replacement does not occur until the last possible moment.

The -unique control argument deletes duplicate sort units from the sorted results. The determination of whether or not a sort unit is to be deleted is independent of sort field specifications; i.e., given a number of nonidentical sort units that contain identical sort fields, all the units do appear in the sorted results.

EXAMPLES: Suppose a segment contains the following lines (where nl represents the ASCII newline character):

```
ABCDEFGHXYnl
ABCDEFXYnl
ABCDEFGHIJXYnl
ABCXYnl
```

The display below shows how the sort_seg command sorts the contents of this segment, according to the arguments specified in the first column (nl stands for the ASCII newline character and ß stands for the ASCII space character).

these arguments	define these sort units	sorted on these fields	giving these results
-dm XY	ABCDEFGH	ABCDEFGHßß	ABCXYnl
	ABCDEF	ABCDEFßßßß	ABCDEFXYnl
	ABCDEFGHIJ	ABCDEFGHIJ	ABCDEFGHXYnl
	ABC	ABCßßßßßßßß	ABCDEFGHIJXYnl
-bk 2	ABCDEFGHABCDEF	ABCDEFGHABCDEF	ABCDEFGHXYnl
-dm XY	ABCDEFGHIJABC	ABCDEFGHIJABCß	ABCDEFXYnl
			ABCDEFGHIJXYnl
			ABCXYnl
-fl 6 4	ABCDEFGHXY	FGHX	ABCXYnl
	ABCDEFXY	FXYß	ABCDEFGHIJXYnl
	ABCDEFGHIJXY	FGHI	ABCDEFGHXYnl
	ABCXY	ßßßß	ABCDEFXYnl
		first second	
-fl 1 4 7 2	ABCDEFGHXY	ABCD GH	ABCDEFGHXYnl
	ABCDEFXY	ABCD XY	ABCDEFGHIJX
	ABCDEFGHIJXY	ABCD GH	ABCDEFXYnl
	ABCXY	ABCX ßß	ABCXYnl
-dm Y	ABCDEFGHXABCDEFX	FGHX DE	ABCDEFGHIJXYnl
-bk 2	ABCDEFGHIJXABCX	FGHI DE	ABCXYnl
-fl 6 4 4 2			
			ABCDEFGHIJXABCX
			ABCDEFXYnl
		first second	
-ofl 6 4	ABCDEFGHXY	FGHX CDE	ABCDEFXYnl
dsc 3	ABCDEFXY	FXYß CDE	ABCDEFGHXYnl
3 asc	ABCDEFGHIJXY	FGHI CDE	ABCDEFGHIJXYnl
	ABCXY	ßßßß CXY	ABCXYnl

SYNTAX AS A COMMAND:

start {-control_arg}

FUNCTION: is employed after the quit signal has been issued in order to resume execution of the user's process from the point of interruption.

CONTROL ARGUMENTS:

-no_restore, -nr
indicates that the standard I/O attachments should not be restored. See "Notes" below.

NOTES: The start command can also be used to resume execution after an unclaimed signal, provided that the condition that caused the unclaimed signal either is innocuous or has been corrected. It restores the attachments of the user_input, user_output, and error_output I/O switches, and the mode of user_i/o to their values at the time of the interruption, unless the -no_restore control argument is given.

The start command can be issued at any time after a quit signal as long as a release command has not been given.

If there is no suspended computation to restart, the command prints the message "start ignored."

SYNTAX AS A COMMAND:

st paths {-control_args}

FUNCTION: prints selected detailed status information about specified storage system entries.

ARGUMENTS:

paths
are the pathnames of segments, directories, multisegment files, and links for which status information is desired. The default pathname is the working directory, which can also be specified by -wd or -working_directory. The star convention can be used.

CONTROL ARGUMENTS:
The following control arguments can be used with any type of entry, and can appear anywhere on the line after the command name and are in effect for the whole line.

-type, -tp
prints the type of entry: segment, directory, multisegment file, or link.

-author, -at
prints the author of the entry.

-date, -dt
prints all the relevant dates on the entry.

-date_time_entry_modified, -dtem
prints the date-time-entry-modified.

-date_time_dumped, -dtd
prints the date-time-dumped by the hierarchy dumper.

-date_time_used, -dtu
prints the date-time-used.

-directory, -dr
selects directories when using the star convention.

-segment, -sm
selects segments when using the star convention.

-link, -lk
selects links when using the star convention.

-date_time_volume_dumped, -dtvd
prints the date-time-dumped by the volume dumper.

-name, -nm
prints all the names on the entry.

-primary, -pri
prints the primary name on the entry.

LIST OF TYPE SPECIFIC CONTROL ARGUMENTS:
The following control_args can only be used for segments, multisegment files, and directories.

-all, -a
prints all relevant information about the object i.e., the type of entry, names, unique identifier, date used, date modified, date branch modified, date dumped by hierarchy and volume dumpers, author, bit count author (if different from author), device, bit count, records used, current blocks (for segments, if different from records used), maximum length in words (if type is segment), safety switch (if it is on), damaged switch (if it is on), user's mode, ring brackets, access class (if it is not null), copy switch (if it is on), and the volume dumper control switches (if on).

-date, -dt
prints all the dates on the entry: i.e., date used, date contents modified, date branch modified, date dumped.

-access, -ac
prints the user's effective mode, ring brackets, access class (if different from the default), and safety switch (if it is on).

-length, -ln
for segments: prints the bit count, the number of records used, the current blocks (if different from records used), and the maximum length in words;

for multisegment files: prints the number of records used by the whole file, the sum of the bit counts of all components, and the number of components;

for directories: prints the number of records used and the bit count.

-unique_id, -uid
   prints the entry's unique identifier.

-date_time_contents_modified, -dtcm
   prints the date-time-contents-modified.

-bc_author, -bca
   prints the bit count author of the entry.

-copy_switch, -csw
   prints whether the copy switch is on or off.

-safety_switch, -ssw
   prints whether the safety switch is on or off.

-damaged_switch, -dsw
   prints whether the damaged switch is on or off.

-mode, -md
   prints the user's effective mode.

-access_class
   prints the access class.

-ring_brackets, -rb
   prints the ring brackets.

-bit_count, -bc
   prints the bit count.

-max_length, -ml
   prints the maximum length of a segment.

-current_length, -cl
   prints the current length in pages.

-records_used, -ru
   prints the records used.

-device, -dv
   prints the logical volume on which the entry resides.

-logical_volume, -lv
   prints the logical volume on which the entry resides. This control argument is the same as the -device control argument.

LIST OF CONTROL ARGUMENTS FOR SEGMENTS:

-comp_volume_dump_switch, -cvds
prints whether the complete volume dump switch is on or off.

-incr_volume_dump_switch, -ivds
prints whether the incremental volume dump switch is on or off.

-use_count, -use
prints the number of page faults taken on the segment since creation.

LIST OF CONTROL ARGUMENTS FOR LINKS:

-all, -a
prints all relevant information about the link, i.e., the pathname of the entry being linked to, names, unique identifier, date link modified, date dumped, and the author of the link.

-chase
prints status information for the targets of links rather than for the links themselves. If a link has no target, link information is printed.

-link_path, -lp
prints the target pathname.

NOTES: If no control argument is specified, the following information is printed for segments, multisegment files, and directories: names, type, date used, date modified, date branch modified, bit count, records used, user's mode, access class.

If no control argument is specified, the following information is printed for links: the pathname of the entry linked to, names, date link modified, date dumped. The -mode, -device, and -length control arguments are ignored for links.

Zero-valued dates (i.e., dates that have never been set) are not printed. In addition, attributes in the default state are not printed.

attribute	default
bit count author	same as author
current blocks	same as records used
access class	null
safety switch	off
copy switch	off
damaged switch	off
complete volume dump switch	on
incremental volume dump switch	on

Directories that have been used to implement multisegment files are labeled as such.

For a description of the attributes listed, see "Entry Attributes" in the MPM Reference Guide.

EXAMPLES: In the first example, the user requests all the status information on the segment named >user_dir_dir>Demo>Jones>working_file.

```
! st >user_dir_dir>Demo>Jones>working_file -all

   names:    test_segment
             working_file
   type:                          segment
   unique id:                     764576046673
   date used:                     01/27/77 1459.0 est Thu
   date modified:                 01/27/77 1459.0 est Thu
   branch modified:               11/19/76 1542.6 est Fri
   date branch dumped:            01/29/77 0305.4 est Sat
   date volume dumped:            01/31/77 0305.4 est Mon
   author:                        Hamilton.Demo.a
   bit count author:              Jones.Demo.m
   volume:                        public
   bit count:                     292968
   records used:                  8
   max length:                    261120
   mode:                          rw
   access class:                  confidential
   ring brackets:                 4, 4, 4
   safety sw:                     on
   ivds switch:                   off
   use count:                     869221
```

(The current blocks, copy switch, damaged switch, and incremental volume dump switch attributes are not printed because they have the default state values.)

In the next example, the user asks for specific status information on entrynames with the first component of newtest in the current working directory.

```
! status -type -mode -date newtest.*

     >user_dir_dir>Demo>Smith>newtest.pl1

type:                       segment
date used:                  01/26/77 2145.0 est Wed
date modified:              01/13/77 1630.0 est Thu
branch modified:            01/13/77 1626.7 est Thu
date branch dumped:         01/14/77 0305.4 est Fri
date volume dumped:         01/16/77 0305.4 est Sun
mode:                       rew
ring brackets:              4, 4, 4


     >user_dir_dir>Demo>Smith>newtest.list

names:                      newtest.list
type:                       link
links to:                   user_dir_dir>Demo>Smith>sub_dir>
                               newtest.list
date link modified:         01/26/74 2139.3 est Sat
```

In the following example, the user asks for status information about the directory named >user_dir_dir>Demo>Black>test.

```
status >user_dir_dir>Demo>Black>test

names:                      test
type:                       directory
date used:                  12/05/77    606.6 est Mon
date modified:              12/05/77    606.6 est Mon
branch modified:            11/29/77    957.2 est Tue
bit count:                  0
records used:               1
mode:                       sma
access class:               Sensitive,Research
```

stop_cobol_run (scr)

SYNTAX AS A COMMAND:

scr {-control_arg}

FUNCTION: causes the termination of the current COBOL run unit.

CONTROL ARGUMENTS:

-retain_data, -retd
leaves the data segments associated with the programs of the run unit intact for debugging purposes. See "Notes" below.

NOTES: The results of the stop_cobol_run command and the execution of the STOP RUN statement from within a COBOL program are identical. Stopping the run unit consists of cleaning up all files that have been opened during the execution of the current run unit, and ensuring that the next time a program that was a component of this run unit is invoked, its data is in its initial state.

To maintain the value of all data referenced in the run unit in its last used state, the -retain_data control argument should be used.

Refer to the run_cobol command for information concerning the run unit and the COBOL runtime environment.

See also the description of display_cobol_run_unit (dcr) and cancel_cobol_program (ccp) in this manual.

SYNTAX AS A COMMAND:

stop_run

FUNCTION: is used in conjunction with the run command to effect an abnormal termination of the run-unit created by the run command.

NOTES: Issuing the stop_run command when a run-unit is not active on the stack causes the command_abort condition to be raised. If no special handler for this condition has been established, the command has no effect. If a run-unit is active on the stack, issuing the stop_run command signals the finish condition, executes the epilogue handlers, and forces a return from the run command to its caller.

For a description of run units see the writeup of run in this manual, and the description available in the New Programmer's Introduction, Order No. AL40.

SYNTAX AS A COMMAND:

```
string {strs}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[string {strs}]
```

FUNCTION: returns a single character string formed by concatenating all of the strings together, separated by single spaces. If no strings are specified, a null character string is returned. If one or more strings are specified, any quotes in these are returned as single quotes.

EXAMPLES: The following interactions illustrate the string command.

```
! string He said, "Hi."
  He said, Hi.
! string He said, """Hi."""
  He said, "Hi."
```

The following interaction illustrates the active function.

```
! string [string This is "food".]
  This is food.
```

SYNTAX AS A COMMAND:

```
strip path {str}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[strip path {str}]
```

FUNCTION: returns the absolute pathname of the specified entry with the last component removed, if the entryname portion has more than one component. If str is specified, the last component is removed only if it matches str.

EXAMPLES: The following interactions illustrate the strip active function. Assume the working directory is >udd>Demo>Jones.

```
! string [strip FW25.report.runoff runoff]
>udd>Demo>Jones>FW25.report
! string [strip FW25.report.runoff]
>udd>Demo>Jones>FW25.report
! string [strip FW25.report.runoff xyz]
>udd>Demo>Jones>FW25.report.runoff
```

strip_entry (spe)

SYNTAX AS A COMMAND:

spe path {str}

SYNTAX AS AN ACTIVE FUNCTION:

[spe path {str}]

FUNCTION: returns the entryname portion of the absolute pathname returned by the strip active function. If str is not specified, the last component of the entryname portion of path is removed (if the entryname has more than one component). If str is specified, the last component is removed only if it matches str.

SYNTAX AS A COMMAND:

```
substr str I {N}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[substr str I {N}]
```

FUNCTION: returns the portion of str starting with the character in position I and continuing for N characters (where I and N are decimal integers; I must be greater than zero and N must be greater than or equal to zero). If N is omitted, the remainder of str is returned. If I is greater than the length of str, the null string is returned. If N is greater than the remainder of str, the remainder is returned.

EXAMPLES: The following interaction illustrates the substr active function.

```
! string [substr programmers 4 4]
gram

! string [substr trounce 3]
ounce
```

SYNTAX AS A COMMAND:

```
suffix path
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[suffix path]
```

FUNCTION: returns the last component of the entryname portion of the specified segment. If that entryname has only one component, the null string is returned.

EXAMPLES:

```
! string [suffix a.pl1]
pl1
```

SYNTAX AS A COMMAND:

system key

SYNTAX AS AN ACTIVE FUNCTION:

[system key]

FUNCTION: returns various installation-dependent system parameters.

LIST OF KEYS:

ARPANET_host_number
ARPA network address of the installation or -1 if the installation is not attached to the ARPA network.

company
company name.

date_up
date that the system was brought up, in the form "mm/dd/yy".

department
computer center department name.

down_until_date
date that the system will next be brought up, if specified by the operator, in the form "mm/dd/yy".

down_until_time
time that the system will next be brought up, if specified by the operator, in the form "hhmm.t".

ds_company
company name, with the characters of the name double spaced.

ds_department
computer center department name, with the characters of the name double spaced.

installation_id
installation identification.

last_down_date
   date that service was last interrupted, whether by shutdown or by crash.

last_down_reason
   reason for the last system service interruption, if known. The reason can be:

shutdown	normal system shutdown
crash	system crash (no number assigned)
n	number of system crash

last_down_time
   time that service was last interrupted.

max_units
   current maximum number of load units, in the form "nnn.n".

max_users
   current maximum number of users.

n_units
   current number of logged-in load units including daemon and absentee, in the form "nnn.n".

n_users
   current number of logged-in users including daemon and absentee.

next_down_date
   date that system will next be shut down, if specified by the operator.

next_down_time
   time that system will next be shut down, if specified by the operator.

next_shift
   next shift number.

reason_down
   reason for next shutdown, if specified by the operator.

shift
   current shift number.

shift_change_date
   date on which current shift number will change to next_shift.

shift_change_time
   time at which current shift number will change to next_shift.

sysid
   version number of the hardcore system tape currently running.

time_up
   time that the system was brought up, in the form "hhmm.t".

SYNTAX AS A COMMAND:

ta key table_path {args}

FUNCTION: performs a variety of operations to create and maintain a set of files on magnetic tape.

ARGUMENTS:

key
is one of the key functions described below.

table_path
is the pathname of a segment created and maintained by the tape_archive command to serve as a table of contents for the archive.

args
are additional arguments or control_arguments as required by the particular key chosen. These arguments are described within the descriptions of each key, below.

Extract Operations:

x tape_archive x table_path {components} {-control_arg}

extracts from the archive those components named by the path arguments, placing them in segments in the storage system. The star convention is allowed for components. The directory where a segment is placed is the directory portion of the component argument. The ACL, names, and other settable segment attributes that were in effect when the component was archived are placed onto the new segment. If a segment of the same name already exists, it observes the duplicate name convention in a manner similar to the copy command. If no component names are specified, all components of the archive are extracted and placed in the working directory.

xf tape_archive xf table_path {components} {-control_arg}

extracts forcibly; operates like x, but forcibly deletes existing segments in the storage system if all their names conflict with names of components being extracted.

Control_args for the extract operation can be the following:

-single_name, -snm
specifies that the name of the component, as it appears in the table, is to be the only name placed on the newly created file in the storage system. The default is to place all the names on the file.

Append Operations:

a tape_archive a table_path {paths} {-control_args}

appends named files to the archive. The star convention is allowed for paths. The files that are appended to the archive are not otherwise affected. If the named file is already in the archive, a diagnostic is issued and the component is not replaced. At least one file must be explicitly named by the path arguments. If the tape archive does not exist, it is created.

ad tape_archive ad table_path {paths} {-control_args}

appends and deletes; operates like a, but then deletes each file that was appended to the archive. Deletion takes place after the tape is processed and the file has been successfully appended to the tape. If the safety switch is on for any named file, the user is queried as to whether the file should be deleted.

adf tape_archive adf table_path {paths} {-control_args}

appends and deletes forcibly; operates like ad, except that the safety switch is disregarded.

Control_args for the append operation can be chosen from the following:

-mode ascii
-mode binary
-mode ebcdic
specify that the file is to be replaced on or appended to the tape archive using the specified encoding mode. If ascii or ebcdic encoding mode is specified, the file is verified to ensure that it can be encoded in the specified mode without loss of information. If it cannot, a warning message is printed, and the encoding mode for that file is changed to

binary. If no explicit mode specifications are specified, the file is encoded in the mode determined by the criteria described under "Default Encoding Modes".

-single_name, -snm
specifies that the name of the component, as specified in the command line, is to be the only name recorded for the file on the volume set.

Replace Operations:

r tape_archive r table_path {paths} {-control_args}

replaces components in, or adds components to the tape archive. The star convention is allowed for paths. If no files are named in the command line, all files of the archive for which files by the same name are found in the user's working directory are replaced. If a file is explicitly named and does not already exist in the tape archive, it is appended, and an advisory is printed.

rd tape_archive rd table_path {paths} {-control_args}

replaces and deletes; operates like r, and then deletes each file that was replaced in or appended to the archive. Deletion takes place after the tape is processed and the file has been successfully replaced on or appended to the tape. If the safety switch is on for any named file, the user is queried as to whether the file should be deleted.

rdf tape_archive rdf table_path {paths} {-control_args}

replaces and deletes forcibly; operates like rd, except that the safety switch is disregarded.

Control_args for the replace operation can be chosen from the following:

-mode ascii
-mode binary
-mode ebcdic
specify that the file is to be replaced on or appended to the tape archive using the specified encoding mode. If ascii or ebcdic encoding mode is specified, the file is verified to ensure that it can be encoded in the specified mode without loss of information. If it cannot, a warning message is

printed, and the encoding mode for that file is changed to binary. If no explicit mode specifications are specified, the file is encoded in the mode determined by the criteria described under "Default Encoding Modes".

-single_name, -snm
specifies that the name of the component, as specified in the command line, is to be the only name recorded for the file on the volume set.

Update Operations:

u tape_archive u table_path {paths}

update operates like r except it replaces only those components for which the corresponding file has a date-time modified later than the date-time associated with the component in the archive. If the file is not found in the archive, it is not added.

ud tape_archive ud table_path {paths}

updates and deletes; operates like u, and then deletes each file that was updated in the archive. Deletion takes place after the tape is processed and the file has been successfully updated on the tape. If the safety switch is on for any named file, the user is queried as to whether the file should be deleted.

udf tape_archive udf table_path {paths}

updates and deletes forcibly; operates like ud, except that the safety switch is disregarded.

Delete Operation:

d tape_archive d table_path components

deletes named components from the archive. The star convention is allowed for components.

df tape_archive df table_path components

deletes forcibly; operates like d, except that the safety switch is disregarded.

Cancel Operation:

cancel tape_archive cancel table_path {components}

cancels any pending requests for the components named. The star convention is allowed for components. This operation removes any requests scheduled to be performed on the named components. If no components are named, the user is queried as to whether all pending requests are to be cancelled.

Table of Contents Operation:

t tape_archive t table_path {components} {-control_args}

prints table of contents and associated information for each named component of the archive (including files scheduled to be placed into the archive), as well as information about the archive itself. The star convention is allowed for components.

Control_args for the table of contents operation can be chosen from the following:

-brief, -bf
prints only the component name.

-long, -lg
prints all of the information shown below.

-no_header, -nhe
suppresses the header information (blocks [1] and [2]), even if -long is specified.

-header, -he
prints the header information (blocks [1] and [2]), even is no components are specified.

-all, -a
causes printing of dead components (components that have been logically deleted or replaced, but still exist on the volume set.)

-pending
prints only those components for which requests are pending.

If no control arguments are specified, a short header, pending operations for the named components, and the component names are printed.

Sample Table of Contents

```
[1] 11 entries in sample.ta;  2 pending requests.
    Mount of volume set for read pending.

[2] Auto compaction limit    = 0.75
    Compaction warning limit = 0.50
    Waste factor: 561/2441 pages = 0.23

    Date-time tape modified:   01/20/77  1231.5 est Fri
    Date-time tape compacted:  01/03/77  0800.3 est Mon

    Current volume set contains 1 tape:
         22304

    Alternate volume set contains no tapes.

[3] REQ  COMPONENT      FILENAME (MODE)   LEN   BC AUTHOR
           DATE ARCHIVED     DATE MODIFIED    DATE DUMPED

    [A]  [B]
    xf bound_simulation_pack.archive  BOUND-SIMULA/0003 (b) 12
       Doe.SIM.a
             12/23/76  1322.6  12/12/76  1430.2  12/12/76  1512.0
                   (in >user_dir_dir>SIM>Doe>library)
```

The information printed in the table of contents for each component includes:

- The type of request pending for the component if any (e.g., "xf");
- The entry name of the component ("bound_simulation_pack.archive");
- The filename of the file on the tape ("BOUND-SIMULA/0003");
- A one-letter indication of the recording mode of the file ("b" for binary, "a" for ascii, and "e" for ebcdic);
- The length of the file in storage-system records;
- The bitcount author of the file ("Doe.SIM.a");
- The date the file was archived to tape;
- The date the file was last modified while still in the storage system previous to its archival;
- The date the file was dumped by the Multics backup facility; and

- If a request is pending, the pathname of the directory in which the file exists or is to be created.

Processing Operation:

go tape_archive go table_path {-control_args}

performs the actual tape mounting and processing of the queued file transferral requests. The current volume set is mounted. Those components scheduled for extraction are processed. Next, additions and replacements are performed. When all tape processing has been completed, requests to delete files in the storage system that have been appended or replaced are processed. Finally, if the processing involves writing to tape, the table is modified to reflect the new state of the tape archive and appended to the tape.

Control arguments for the processing operation can be any of the following:

-retain all
specifies that the volume set is to remain mounted after processing is complete. In cases where several successive tape processing operations are planned, this control argument speeds up processing of requests by reducing the physical handling of the tapes. The volume set remains mounted until the processing operation is invoked with the -retain none control argument.

-retain none
reverts the effects of -retain all as described above.

-long, -lg
causes a message to be printed for each file search, extraction, or appending, as they are perfomed on the volume set.

Compaction Operation:

compact tape_archive compact table_path

schedules the tape archive for compaction. The compaction process causes the active components on the current volume set to be copied onto the alternate volume set. This process removes cumulative tape waste attributable to inactive tape files (components that have been logically deleted, updated, or replaced, but never physically removed.) Other file

transferral requests can be processed at the same time that the archive is being compacted. After the compaction operation, the alternate volume set becomes the current volume set, and vice versa.

Parameter Alteration Operations:

alter tape_archive alter table_path alter_spec

changes global attributes of the tape archive that can be set by the user. The specific attribute modified depends on the alter_spec arguments. Arguments recognized in this position are:

warning_limit floatnum
specifies that an advisory message is to be printed whenever the number of wasted tape records on the volume set reaches or exceeds a certain fraction of the total tape records. The floatnum argument must be from 0.0 to 1.0. The initial default for warning_limit is 0.5.

auto_limit floatnum
specifies that the tape archive should be automatically scheduled for compaction at the next mounting whenever the number of wasted tape records on the volume set exceed a certain fraction of the total tape records used. When compaction is automatically scheduled in this manner, an advisory message is printed. The floatnum argument must be between 0.0 and 1.0. The initial default for auto_limit is 1.0 (never automatically compact.)

volume old_volume_spec new_volume_spec {-alternate}
specifies that the volume (reel) with label new_volume_spec is to take the place of the volume old_volume_spec. If old_volume_set is the null string, and the -alternate control argument is specified, new_volume_set is appended to the alternate volume set; otherwise, it is appended to the primary volume set. If new_volume_spec is the null string, old_volume_spec is deleted from the appropriate volume set.

volume -number N new_volume_spec {-alternate}
specifies that the volume with label new_volume_spec is to take the place of the Nth volume in the primary volume set (the alternate volume set if the -alternate control argument appears.) If new_volume_spec is the null string, the volume is deleted. If N is one greater than the number of volumes currently contained in the volume set, the volume is appended to the volume set.

compaction off
unschedules any pending compaction of the tape archive.

Load Table Operation:

load_table tape_archive load_table table_path volume_id

causes the copy of the online table kept on a volume set to be loaded into the segment specified by table_path. If the segment already exists, the user is asked whether it should be overwritten. Only the volume id of the initial volume in the set need be specified.

Table Reconstruction Operation:

reconstruct tape_archive reconstruct table_path volume_id

causes an entire volume set to be scanned, and an online table to be reconstructed strictly from the apparent contents of the tape. This operation should only be used when tape errors or another catastrophic failure makes the load table operation impossible. The table created may include files that were previously deleted but never physically compacted out.

Interactive Mode:

direct tape_archive direct table_path {-control_arg}

allows a user to direct the actions of tape_archive using an interactive mode in which each line typed is interpreted as a key followed by the arguments (with the exception of table_path) accepted by that key. This mode of operation is exited by typing "go". If any requests are outstanding when the mode is exited, the tapes are automatically mounted and the requests performed (except as noted below).

While in the interactive mode, all requests are maintained in a temporary copy of the online table, allowing the user to abort processing if desired without recording any requests in the actual online table.

All keys are accepted in this mode of operation with the exception of the following keys:

load_table
reconstruct

In addition, the following special commands are accepted in this mode of operation:

quit
specifies that the interactive mode is to be exited without performing the actual processing of the requests. Unless preceded by the "save" command, all requests made in this invocation of tape_archive are discarded. If unsaved requests exist, the user is asked to confirm the command.

save
specifies that all requests made during this invocation of the command are to be recorded permanently in the table.

go
specifies that all preceding requests are to be recorded into the table, and that the volume set is to be mounted and processed.

..{command_line}
passes the specified command line to the command processor.

causes tape_archive to identify itself.

Allowable control arguments to the direct operation are:

-retain all
specifies that the volume set is not to be dismounted when the "go" request is complete. If this control argument is specified, the "go" request does not cause termination of the command invocation, but returns the user to the interactive mode of tape_archive so that more requests may be entered. The "quit" request must be used to exit this mode and dismount the volume sets.

Default Encoding Modes

If no particular encoding mode is specified for files being appended to or replaced in the archive, the following criteria are applied to determine the most appropriate mode.

When performing file replacement, the default encoding mode remains unchanged if it is determined that the file has not

been altered in any way that precludes encoding it in the same mode; otherwise, a diagnostic is printed, and the replacement is performed in binary mode.

Otherwise, the file is recorded in binary mode for reasons of efficiency.

Tape File Naming Conventions

Tape files of a tape_archive volume set follow certain conventions with respect to ordering and naming.

Each user file is preceded by an attribute file, containing the information necessary to re-create the file faithfully in the storage system (e.g. names, ACL, and so on.) Attributes files are named "ATTRIBUTEFILENNNN", where NNNN is the physical file number (by order of occurrence on the tape) of the attributes file, e.g., "ATTRIBUTEFILE0023".

Each user file is named with a unique name constructed of all or part of the Multics entry name of the file, one or more slash characters, and the physical file number of the file. The Multics file name is truncated or padded with slashes to twelve characters. For example, the name of the tape file containing the Multics file named "alpha" might be "ALPHA////////0024", and the name of the tape file containing the Multics file named "source.archive" might be "SOURCE.ARCHI/0037".

Copies of the online table describing the tape are named "ONLINE-TABLE-NNNN" where NNNN is a number representing the serial number of the online tables on this volume set. This number starts from 1 and increases by 1 each time a new table is written to the tape.

NOTES: A tape archive can be used to temporarily hold files that will be needed at some future time, but that meanwhile take up large amounts of expensive storage space. Additionally, tape archives can be used to transfer files between Multics systems, and in a limited fashion, from Multics to other operating systems.

A tape archive consists of one or more reels of tape, known as the "volume set", on which files are stored in ANSI standard tape format. The constituent files that compose the tape archive are called components of the archive. Associated with each tape archive is a Multics segment known as the table. This segment is created and maintained by the tape_archive command, and contains information about each component in the archive.

The tape_archive command provides the user with the ability to append components to the archive, replace components of the archive with new versions, extract components from the archive, and delete components from the archive. In addition, other facilities include listing the contents of the archive and re-creating the online table in the event of catastrophe or for file transfer purposes.

Requests to move components between the tape and the storage system are specified via invocations of the tape_archive command before any reels are actually mounted and processed. When all desired transferral requests have been specified, the user then invokes the tape_archive command to mount and process the tape.

An interactive mode of operation is supplied that allows the user to specify multiple requests to a single invocation of the command, and causes the requests to be automatically performed after all requests have been satisfactorily entered.

SYNTAX AS A COMMAND:

```
tm paths
tms segnos
tmr ref_names
tmsr ref_names
```

FUNCTION: allows the user to remove a segment from his address space and resets links to the terminated segment. It is most useful when recompiling procedures so that the new version can be invoked with no linkage complications. Therefore the same operation is done automatically by the Multics system compilers. The user can also call this command directly in order to test various versions of a procedure.

ARGUMENTS:

paths
are pathnames of segments to be terminated.

ref_names
are the reference names of segments to be terminated.

segnos
are segment numbers (in octal) to be terminated.

NOTES: The tms command allows termination by segment number rather than by pathname.

The tmr command allows termination by reference name rather than by pathname. The segment itself is terminated, not merely the particular reference name specified.

The tmsr command allows termination of a single reference name. Unless the specified reference name is the only one by which the segment is known, the segment is not terminated.

Caution must be exercised when using these commands as the user can unintentionally terminate (within the user's process only) a segment of the command language interpreter or another critical piece of the environment. The usual result is termination of the user's process.

The terminate commands, with their short names, are as follows:

terminate_segno, tms
terminate_single_refname, tmsr
terminate_refname, tmr

The star convention is not recognized in any of the above commands.

SYNTAX AS A COMMAND:

time {dt}

SYNTAX AS AN ACTIVE FUNCTION:

[time {dt}]

FUNCTION: returns a four-digit time of day in the form "hh:mm" where 00 ≤ hh ≤ 23 and 00 ≤ mm ≤ 59.

ARGUMENTS:

dt
is a date-time in a form acceptable to convert_date_to_binary_. If no argument is specified, the current time is used.

SYNTAX AS A COMMAND:

```
times {num_args}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[times {num_args}]
```

FUNCTION: returns the product of the num_args. If no num_args are specified, 1 (the multiplicative identity) is returned.

EXAMPLES:

```
! string [times 6 7.3]
  43.8
```

SYNTAX AS A COMMAND:

trace {-control_args} names

ARGUMENTS:

names
is a pathname or reference name. The reference name or entry portion of a pathname is used in the trace table. (See "Notes" below.)

CONTROL ARGUMENTS:
apply to the names arguments that follow, and, if applicable, change the current value in the trace control template (TCT). (See "Notes" below.)

-after N
calls the command processor after calling the traced procedure every N times. The default is to NOT call.

-argument N, -ag N
prints the arguments every Nth time the procedure is entered. The default is to NOT print.

-before N
calls the command processor before calling the traced procedure every N times. The default is to NOT call.

-brief, -bf
prints a short form of the monitoring information.

-depth N, -dh N
monitors to the maximum recursion depth of N. The default is unlimited.

-every N, -ev N
monitors every Nth call. The default is 1.

-execute STR, -ex STR
executes the Multics command line specified by the string STR whenever the procedure is monitored.

-first N, -ft N
starts monitoring on the Nth call. The default is 1.

-govern STR, -gv STR
limits/does not limit the recursion level for a procedure, where STR can be the string on or off. The default is off.

See "Recursion Limiting" below.

-in
prints the arguments only on entry. This is the default.

-inout
prints the arguments on both entry and exit. The default is on entry only.

-io_switch STR, -is STR
changes the switch for output to the switch specified by STR. (See "Notes on Changing Output Switch" below.)

-last N, -lt N
stops monitoring after the Nth call. The default is to continue for the life of the process.

-long, -lg
prints the long form of the monitoring information. This is the default.

-meter STR, -mt STR
meters/does not meter the time spent in the procedure, where STR can be the string on or off. The default is off. See "Notes on Metering" below.

-out
prints the arguments only on exit. The default is only on entry.

-off entryname
stops monitoring the procedure specified by entryname. The procedure remains in the trace table, and calls continue to be counted.

-on entryname
resumes monitoring the specified procedure when it has been turned off previously by -off.

-remove entryname, -rm entryname
removes the specified procedure from the trace table. Tracing can be removed at any time.

-reset entryname, -rs entryname
sets the number of calls and recursion depth of the specified procedure to zero.

-return_value STR, -rv STR
prints/does not print the return value on exit, where STR can be the string on or off. The default is off. This control

argument assumes the entry is a function.

-status *, -st *
prints the procedures being monitored and their counters. (See "Notes" below.)

-status entryname, -st entryname
prints the trace parameters and counters for the procedure specified by entryname. (See "Notes" below.)

-stop_proc path, -sp path
changes the procedure that is called for stop requests from the command processor to the procedure specified by path. To reset the stop procedure, issue this control argument with no path argument.

-subtotal, -stt
prints and does not clear the metering statistics.

-template, -tp
lists the trace control template.

-watch STR, -wt STR
watches all procedures being traced for a change in the current contents of the memory word(s) specified by the string STR. See "Notes on Watch Facility" below.

NOTES: The trace command is a debugging tool that lets the user monitor all calls to a specified set of external procedures. The trace command modifies the standard Multics procedure call mechanism so that whenever control enters or leaves one of the specified procedures, a debugging procedure is invoked. The user can request the following:

1. Print the arguments at entry, exit, or both.

2. Stop (by calling the command processor) at entry, exit, or both.

3. Change the frequency that tracing messages are printed (e.g., every 100 calls, after the 2000th call, only if the recursion depth is less than five, etc.).

4. Execute a Multics command line at entry, exit, or both.

5. Meter the time spent in the various procedures being monitored.

Use of the trace command is subject to the following restrictions:

1. Only external procedures compiled by PL/I, FORTRAN, alm, or COBOL can be traced.

2. Ring 0 or gate entries cannot be traced.

3. Incorrect execution results if the traced procedure looks back a fixed number of stack frames, e.g., cu_$arg_ptr cannot be traced.

4. Only 1000 procedures can be traced at one time. Up to 16 locations can be watched at one time.

5. The procedure being traced and the trace package itself must share the same combined linkage segment.

6. A procedure in a bound segment can only be traced if its entry point is externally available.

7. An alm procedure can only be traced if it uses the entry operator.

The procedure whose pathname is specified in the command line is added to the trace table with the tracing parameters from the trace control template (TCT). If the procedure is already in the table, the counters are reset and the current parameters in TCT are used.

For control arguments that affect procedures being traced, the argument is an entryname or an asterisk (*). If an entryname is used, the control argument applies to that procedure. If an asterisk is used, the control argument is applied to all entries in the trace table. All control arguments that affect the TCT must have a number argument (indicated by N above).

The trace command processes all arguments from left to right and they take effect immediately when read. Any parameters that are set keep their values across invocations of the command. For example:

```
! trace -every 2 alpha -every 3 beta
```

traces the program alpha on every second call and the program beta on every third call.

If the user then types:

! trace gamma

it traces the program gamma on every third call because the value for -every was set during the preceding invocation of the command.

EXAMPLES:

The command line:

! trace -ag 1 -inout test

prints the arguments for test on entry and exit.

The command line:

! trace -ag 2 -in -depth 6 test

prints the arguments for test every second time test is entered up to a recursion depth of six, i.e., 2, 4, 6.

The command line:

! trace -govern on test

prints the arguments of test each time test is called with a new maximum recursion depth. The trace procedure calls the command processor every time the recursion depth is a multiple of 10.

! The command line:

! trace -st * -tp

lists the procedures in the trace table and prints the values of the trace control template.

NOTES ON MESSAGE FORMAT: The message printed when control enters a procedure can appear in any one of several formats, depending on the setting of the brief switch and the status of the calling procedure. If the calling procedure is unbound or occurs in a bound segment containing a bindmap, the message takes the form:

Call 4.1 of alpha from beta|127, ap = 204|10746.

This is the fourth call of procedure alpha, at recursion level 1. The call comes from location 127 in component beta, and the argument list is at 204|10746. If the procedure making the call is in a bound segment that does not contain a bindmap, the message takes the form:

Call 4.1 of alpha from bound_gamma|437 (beta), ap = 204|10746.

The name in parentheses may not always be available and may be omitted in some cases. If the user has requested the brief output mode, the message is shortened to:

Call 4.1 of alpha.

When tracing is requested for a procedure, the parameters for that entry are taken from the trace control template (TCT). If the user does not alter the values in the TCT, the initial default values are used (see below). The initial values in the TCT specify that every call is monitored.

NOTES ON TRACE CONTROL TEMPLATE: As mentioned earlier, the trace table entry holds a number of parameters for each procedure to be traced. The values of the parameters are determined by the contents of the TCT at the time the table entry is filled in. These parameters are used in conjunction with N (the number of calls to the traced procedure in this process) and R (the current recursion depth) to control when and how the procedure should be monitored. The execution count (N) is set to 0 when tracing is first started and is incremented by 1 every time the traced procedure is called. The recursion depth (R) is set to 0 when tracing is first started and is incremented by 1 every time control enters the traced procedure and is decremented by 1 every time control leaves the traced procedure.

Let:

D =
the maximum recursion depth to be monitored (-depth).

F =
the number of the first call to be monitored (-first).

L =
the number of the last call to be monitored (-last).

E =
how often monitoring should occur (-every).

B =
the number of times the procedure is called before trace stops at entry to the traced procedure (-before).

A =
the number of times the procedure is called before trace stops at exit from the traced procedure (-after).

AG =
the number of times the procedure is called before trace prints the arguments of the traced procedure (-argument).

I =
a bit that is "1"b if the tracing procedure should print the arguments of the traced procedure when control goes into the traced procedure (-in).

O =
a bit that is "1"b if the tracing procedure should print the arguments of the traced procedure when control goes out of the traced procedure (-out).

A call is monitored and the tracing procedure is called if, and only if:

F <= N <= L

R <= D

mod(N,E) = 0

If AG ^= 0, mod(N,abs(AG)) = 0, and I = "1"b, trace prints the values of the arguments (if any) being passed to the traced procedure. All of the arguments are listed when AG < 0. If AG < 0, the procedure is assumed to be a function and the value of the last argument is printed after the procedure returns.

If B ^= 0 and mod(N,B) = 0, the monitoring procedure prints "Stop" and calls the command processor (or a user-set procedure if the -stop_proc control argument was used). This call occurs before the procedure being traced has created its stack frame.

After control leaves the traced procedure, trace prints a line of the form:

Return N.R from alpha.

If AG ^= 0 and mod(N,abs(AG)) = 0, then all of the arguments of the traced procedure are printed if 0 = "1"b; otherwise, if AG < 0, the value of the last argument (assumed to be the value of the function) is printed.

Finally, trace calls the command processor. If the -stop_proc control argument was specified, a procedure set by the user is called. This call occurs after the stack frame of the procedure being traced has been destroyed.

NOTES ON METERING: The trace command can be used to meter the execution of a specified set of procedures. If the metering feature is being used, trace does not call the debugging procedure when control enters a procedure being traced; instead, it determines the current time and the virtual CPU time used, and the number of page faults taken by the user's process before control enters and after control leaves the traced procedure. This information is used to compute the real time and CPU time used, and the number of page faults taken by the traced procedure on a local and global basis. The global CPU time is the time spent in the procedure including the time spent in any procedures that it calls. The local CPU time does not include the time spent in any traced procedure called by the procedure, but it does include time spent in called procedures that are not being traced. The local and global versions of real time and page faults are calculated in a similar manner. Metering is only done when the first, last, every, and depth tracing conditions are satisfied.

The control argument:

-meter on, -mt on

sets the metering switch in the TCT; any procedures added to the trace table or that have their table entries updated after this argument is used are metered.

The control argument:

-meter off, -mt off

turns off the metering switch in the TCT; any procedures currently being metered continue to be metered.

The control argument:

-total

causes trace to print the metering statistics of all procedures in the trace table. The output specifies the number of calls (#CALLS), global CPU time (GCPU), global real time (GREAL), global page waits (GPWS), local CPU time (LCPU), local real time (LREAL), local page waits (LPWS), and the usage percentage (%USAGE) based on local CPU time, of all the procedures being metered. The metering statistics are set to 0 after they are printed.

The control argument:

-subtotal, -stt

prints the same information as the -total control argument, but does not clear the statistics.

NOTES ON RECURSION LIMITING: The control argument:

-govern on, -gv on

sets a bit in the TCT that causes recursion limiting to be in effect for any procedure subsequently added to the trace table. When the governing feature is used, the depth control parameter is ignored and trace prints the call message only when the recursion depth of the traced procedure reaches a new, maximum depth. Each call message has a recursion depth one greater than the previous call message. In addition, trace calls the command processor (or a user-defined procedure if the -stop_proc control argument was used) whenever the recursion depth is a multiple of 10. Return messages are not printed. This feature enables the user to find and limit uncontrolled recursion; it can be very useful in finding the procedure(s) responsible for fatal process error.

The control argument:

-govern off, -gv off

turns off the governing switch in the TCT; any procedure currently being governed continues to be governed.

NOTES ON WATCH FACILITY: The trace command has an optional watch facility in that trace watches the contents of a set of previously specified memory cells. The cells are checked at every entry to and every exit from every traced procedure. As long as the values in the locations being watched remain the same, no action is taken and no tracing messages are printed. The tracing message is printed as soon as trace finds that any of the locations being watched has had its value changed. This can be found either at entry to or exit from the traced procedure. When any value changes, the tracing message is preceded by lines that give the new values of all of the locations that have changed, and the command processor (or a user-set procedure if the -stop_proc control argument was used) is called even if the A or B conditions are not met. When execution continues, the locations that have changed are watched with the new value being used in subsequent checks. This feature can be very useful in determining the user procedures that have incorrectly modified a word of storage.

The control argument:

-watch STR, -wt STR

causes all procedures being traced to watch for a change in the current contents of the memory word(s) specified by the string STR. This string, specifying the location, can consist of a single address specification or a series of address specifications separated by blanks and surrounded by quotes. If an address specification does not contain a vertical bar (|), it is taken to be an octal number giving a location in the stack; otherwise, it is taken to be a segment number and offset in octal in the standard form, e.g., segment_number|offset.

The control argument:

-watch off, -wt off

turns off the watch facility.

The watch facility differs from other trace facilities in that there is a single table of locations being watched that is used by all procedures being traced. When the -watch control argument is processed, the new location(s) specified replace any locations currently in the watch table. There is no provision made for removing a single location from the watch table; the user must reissue a watch request that omits the location to be removed from the table.

NOTES ON COMMAND EXECUTION:

The command execution facility of trace allows the user to specify a Multics command line to be executed whenever the trace debugging procedure is called. The trace procedure calls the command processor with the specified string after printing the tracing message, but before the stop request causes the command processor to be called.

The control argument:

-execute string

sets the execution string parameter in the TCT. Since string is a single argument, it must be enclosed in quotes if it contains any spaces. The execution parameter in the TCT is

turned off if string has zero length (-execute ""). The following command line:

```
! trace -ex time test
```

causes trace to execute the time command before and after test is called.

NOTES ON CHANGING OUTPUT SWITCH: All of the messages from the trace command that may be generated while actually monitoring procedures are normally written on the user_i/o switch so that trace can conveniently be used with procedures that change the attachment of the normal switch, user_output. The control argument:

-io_switch STR

causes trace to write further monitoring output on the switch specified by STR. The switch must already be attached and opened for stream_output.

SYNTAX AS A COMMAND:

ts {-control_args}

FUNCTION: prints a detailed explanation of the current process stack history in reverse order (most recent frame first). For each stack frame, all available information about the procedure that established the frame (including, if possible, the source statement last executed), the arguments to that (the owning) procedure, and the condition handlers established in the frame are printed. For a description of stack frames, see "Multics Stack Segments" in the MPM Subsystem Writers' Guide.

CONTROL ARGUMENTS:

-brief, -bf
suppresses listing of arguments and handlers. This control argument cannot be specified if -long is also specified as a control argument.

-long, -lg
prints octal dump of each stack frame.

-depth N, -dh N
dumps only N frames.

NOTES: The trace_stack command is most useful after a fault or other error condition. If the command is invoked after such an error, the machine registers at the time of the fault are also printed, as well as an explanation of the fault. The source line in which it occurred can be given if the object segment is compiled with the -table option.

NOTES ON OUTPUT FORMAT: The trace_stack command is most useful after a fault or other error condition. If the command is invoked after such an error, the machine registers at the time of the fault are also printed, as well as an explanation of the fault. The source line in which it occurred can be given if the object segment is compiled with the -table option. When trace_stack is invoked, it first searches backward through the stack for a stack frame containing saved machine conditions as the result of a signalled condition. If such a frame is found, tracing proceeds backward from that point; otherwise, a comment is printed and tracing begins with the stack frame preceding trace_stack.

If a machine-conditions frame is found, trace_stack repeats the system error message describing the fault. Unless the -brief control argument is specified, trace_stack also prints the source line and faulting instruction and a listing of the machine registers at the time the error occurred.

The command then performs a backward trace of the stack, for N frames if the -depth N argument was specified, or else until the beginning of the stack is reached.

For each stack frame, trace_stack prints the offset of the frame, the condition name if an error occurred in the frame, and the identification of the procedure that established the frame. If the procedure is a component of a bound segment, the bound segment name and the offset of the procedure within the bound segment are also printed.

The trace_stack command then attempts to locate and print the source line associated with the last instruction executed in the procedure that owns the frame (that is, either a call forward or a line that encountered an error). The source line can be printed only if the procedure has a symbol table (that is, if it was compiled with the -table option) and if the source for the procedure is available in the user's working directory. If the source line cannot be printed, trace_stack prints a comment explaining why.

Next, trace_stack prints the machine instruction last executed by the procedure that owns the current frame. If the machine instruction is a call to a PL/I operator, trace_stack also prints the name of the operator. If the instruction is a procedure call, trace_stack suppresses the octal printout of the machine instruction and prints the name of the procedure being called.

Unless the -brief control argument is specified, trace_stack lists the arguments supplied to the procedure that owns the current frame and also lists any enabled condition, default, and clean-up handlers established in the frame.

If the -long control argument is specified, trace_stack then prints an octal dump of the stack frame, with eight words per line.

EXAMPLES: After a fault that reenters the user environment and reaches command level, the user invokes the trace_stack command.

For example, after quitting out of the list command, the following process history might appear:

```
  ! list

Segments=8, Records=3

rew  0  mailbox
r w
QUIT

  ! trace_stack
quit in ipc_$block|156
(>system_library_1>bound_command_loop_|156)
  No symbol table for ipc_
   156   400010116100      cmpq    pr4|10

Machine registers at time of fault

pr0 (ap)  263|4656          pl1_operators_$operator_table|162
                               (external symbol in separate
                                nonstandard text section)
pr1 (ab)  103|264           scs|264
pr2 (bp)  14|12200          as_linkage|12200
pr3 (bb)  113|0             tc_data|0
pr4 (lp)  253|2250          !BBBJGjFkPBWcNZ.area.linker|2250
                               (internal static|0 for ipc_)
pr5 (lb)  244|3614          stack_4|3614
pr6 (sp)  244|3500          stack_4|3500
                               ( -> "kcpMbLH +0000000")
pr7 (sb)  244|0             stack_4|0

x0     73   x1       0   x2       0   x3 600000
x4      0   x5      32   x6    3033   x7       4
a 000000000000  q 000000000004  e 0
Timer reg - 1746005, Ring alarm reg - 0
```

trace_stack (ts)

```
SCU Data:

  4030    400270250011 000000000021 400270000000 000000672000
          000156000200 000154000700 002250370000 600044370120

Connect Fault (21)
At: 270|156 ipc_|156  (bound_command_loop_|156)
On: cpu a (#0)
Indicators:  ^bar
APU Status:  xsf, sd-on, pt-on, fabs
CU Status:   rfi, its, fif
Instructions:
  4036    002250 3700 00      epp4      2250
  4037   6 00044 3701 20      epp4      pr6|44,*

Time stored: 08/02/77  1635.5 edt Tue (1045416743612266O2)
Ring:          4

Backward trace of stack from 244|3500


3500 quit ipc_$block|156 (bound_command_loop_|156)
  No symbol table for ipc_
   156   400010116100     cmpq    pr4|10
         ARG  1: 253|5704 !BBBJGjFkPBWcNZ.area.linker|5704
         ARG  2: 244|3152 stack_4|3152
         ARG  3: 0


2720      tty_$tty_get_line|2442 (bound_iox_|11546)
  No symbol table for tty_
  call_ext_out to ipc_$block
         ARG  1: 253|4320 !BBBJGjFkPBWcNZ.area.linker|4320
                          (internal static|154
                             for find_iocb)
         ARG  2: 244|2660 stack_4|2660 ( -> "fo stuff")
         ARG  3: 128
         ARG  4: 0
         ARG  5: 0


2400      listen_$listen_|461 (bound_command_loop_|1325)
  No symbol table for listen_
  call_ext_out to iox_$get_line
         ARG  1: ""
         on "cleanup"  call listen_|256
            (bound_command_loop_|1122)
```

trace_stack (ts)

```
2100      process_overseer_$process_overseer_|473
            (bound_command_loop_|21433)
  No symbol table for process_overseer_
  call_ext_out_desc to listen_$listen_
          Argument list header invalid.
          on "any_other"
            call standard_default_handler_
               $standard_default_handler_3
            (external symbol in separate nonstandard
               text section)


2000      user_init_admin_$user_init_admin_|36
             (bound_command_loop_|21676)
  No symbol table for user_init_admin_
 21676   700036670120     tsp4    pr7|36,* alm_call
          No arguments.

End of trace.

r 1635 1.756 40.790 207 level 2, 9
```

SYNTAX AS A COMMAND:

```
translate strA strB {strC}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[translate strA strB {strC}]
```

FUNCTION: returns translation in which all the characters of a string strA that appear in a string strC are translated to the corresponding characters in a string strB. If strC is omitted, a default string containing all possible 9-bit bit patterns is used, as returned by collate9.

EXAMPLES:

```
! string [translate abcdefgh BDFH bdfh]
  aBcDeFgH
! string [translate "My work" KLMNOPQRSTUVWXYZ klmnortvwxyz]
  MY WORK
```

SYNTAX AS A COMMAND:

```
trunc num
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[trunc num]
```

FUNCTION: returns the largest decimal integer whose absolute value is less than or equal to the absolute value of num.

EXAMPLES:

```
! string [trunc 7.6]
7
! string [trunc -7.6]
-7
```

SYNTAX AS A COMMAND:

```
   tc {-control_arg} path {length}
or:
   tc segno {length}
```

FUNCTION: truncates a segment to an optionally specified length and resets the bit count accordingly, setting the bit count author to be the user who invoked the command. The segment can be specified by pathname or segment number.

ARGUMENTS:

path
is the pathname of a segment. The star convention is NOT allowed.

segno
is an octal segment number.

length
is an octal integer indicating the length of the segment in words after truncation. If no length argument is provided, zero is assumed.

CONTROL ARGUMENTS:

-name, -nm
specifies that the octal number following it is a pathname.

ACCESS REQUIRED: The user must have write access on the segment to be truncated.

NOTES: If the segment is already shorter than the specified length, its length is unchanged, but the bit count is set to the specified length.

This command should not be used on segments that are (or are components of) structured files.

EXAMPLES:

The command line:

```
! tc alpha 50
```

truncates segment alpha to 50 words; i.e., all words from word 50 (octal) on are zero. The bit count of the segment is set to the truncated length.

SYNTAX AS A COMMAND:

ur resources {-control_args}

FUNCTION: unassigns one or more resources that have been assigned to the user's process by the Resource Control Package (RCP).

ARGUMENTS:

resources
specify the resources to be unassigned from the user's process. Currently, the only resources managed by RCP are devices. If a device is attached, it is automatically detached. A user can unassign all devices assigned to the process by specifying the keyword "all", or unassign one device by specifying its name.

CONTROL ARGUMENTS:

-comment STR, -com STR
is a comment string that is displayed to the operator when the resource is unassigned. This comment is displayed only once, even if several resources are being unassigned. (See the assign_resource command for details about comment strings.)

-admin, -am
forces an unassignment. This control argument should be specified by highly privileged users who want to unassign a resource that is assigned to some other process.

EXAMPLES: In the example that follows, the user unassigns a tape previously assigned by the assign_resource command by typing the command line:

```
! ur tape_03
```

SYNTAX AS A COMMAND:

```
underline str_args
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[underline str_args]
```

FUNCTION: underlines str_args. Each str_arg is underlined separately in the return value, and this value is canonicalized. Those str_args containing quotes or spaces are quoted in the return value.

EXAMPLES: The following interactions illustrate the underline active function.

```
! string [underline She said, "be quiet!"]
She said, be quiet!
! string [underline She said, """be quiet!"""]
She said, "be quiet!"
! string [underline "Now is the time for all ""folks...."""]
Now is the time for all "folks...."
! string [underline "Now is" the time "for all horses."""]
Now is the time for all horses."
```

SYNTAX AS A COMMAND:

```
unique {arg}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[unique {arg}]
```

FUNCTION: returns a unique character string as generated by the unique_chars_ subroutine (described in the MPM Subroutines). Unique character strings are 15 characters long and begin with an exclamation mark (!).

ARGUMENTS:

arg
is an octal number from which the unique character string is to be generated. If arg is omitted, the current clock value is assumed.

EXAMPLES:

```
! string [unique]
!BBBJHwHMzmmxMF
! string [unique [user process_id]]
!BPGBzBBBBBBB
```

SYNTAX:

ul {paths} {-control_args}

FUNCTION: deletes link entries.

ARGUMENTS:

paths
specify storage system link entries to be deleted.

CONTROL ARGUMENTS:

-brief, -bf
inhibits the printing of an error message if a link to be deleted is not found.

-force
suppresses the query "Do you want to unlink ** in <dir_path>?" when appropriate.

-long, -lg
prints a message of the form "Deleted link <path>" for each link deleted.

-name STR, -nm STR
specifies a nonstandard entry name STR (e.g., an invalid starname such as *.compout or a name containing <.)

ACCESS REQUIRED: The user must have modify permission on the directory containing the link.

NOTES: Use delete to delete segments and multisegment files. Use delete_dir to delete entries.

For a discussion of links see "Entry Attributes" in the MPM Reference Guide.

SYNTAX AS A COMMAND:

```
upper_case {strs}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[upper_case {strs}]
```

FUNCTION: returns all strings with lowercase alphabetic characters translated to uppercase characters. Returned strings are separated from each other by single spaces. Each input string is returned as a separate output string, enclosed in quotes if necessary.

EXAMPLES:

```
! string [upper_case Now is the time.]
  NOW IS THE TIME.
! string [upper_case "Now is the time"]
  NOW IS THE TIME
```

SYNTAX AS A COMMAND:

```
user key
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[user key]
```

FUNCTION: returns various user parameters.

LIST OF KEYS:

abs_queue
queue number in which user's absentee process is running; "interactive" if user has no absentee process.

absentee
true if the user is an absentee user; otherwise false.

absin
absolute pathname of absentee user's absentee input segment including the absin suffix; otherwise a null string.

absout
absolute pathname of absentee user's absentee output segment; otherwise a null string.

anonymous
true if the user is an anonymous user; otherwise false.

attributes
user's attributes determined at login time. The attributes are separated by a comma and a blank and end with a semicolon. The attributes are chosen from the following:

administrator	multip	nopreempt
anonymous	no_eo	nostartup
brief	no_prime	preempting
daemon	no_secondary	primary_line
dialok	no_warning	v_outer_module
guaranteed_login	nobump	vhomedir
igroup	nolist	vinitproc

auth
short string for the authorization of the user's process or system_low.

auth_long
long string (in quotes) for the authorization of the user's process or "system_low".

brief_bit
true if the user specified the -brief control argument in the login line; otherwise false.

charge_type
device charge type associated with the user's terminal.

cpu_secs
user's CPU usage (in seconds) since login, in the form "sss.t" with leading zeros suppressed.

device_channel
I/O device channel associated with user's terminal.

group
user's load control group.

initial_term_id
user's terminal identifier code at login.

initial_term_type
user's terminal type at login.

line_type
line type of the user's terminal. It can have one of the following values:

MC	Sync	SYNC1
TELNET	G115	SYNC2
none	BSC	SYNC3
ASCII	202ETX	POLLED_VIP
1050	ASYNC1	VIP
2741	ASYNC2	
ARDS	ASYNC3	

log_time
user connect time (in minutes) since login, the form "mmm.t".

login_date
date at login time, in the form "mm/dd/yy".

login_time
time of login, in the form "hhmm.t".

login_word
word used to log in, i.e., login, enter, enterp.

max_auth
   short string for the maximum authorization of the user's process or "system_low".
max_auth_long
   long string (in quotes) for the maximum authorization of the user's process or "system_low".

n_processes
   number of processes created for the user since login; this number equals 1 plus the number of new_proc commands plus the number of fatal process errors.

name
   user's User_id at login time.

outer_module
   initial outer module for the terminal channel.

preemption_time
   time at which the primary user becomes eligible for group preemption, in the form "hhmm.t".

process_id
   user's process identification in octal.

process_type
   user's process type. It can have one of the following values:
     interactive
     absentee
     daemon

process_overseer
   name of user's process overseer.

project
   user's Project_id.

protected
   true if the user is currently a primary user and protected from preemption; otherwise false.

secondary
   true if the user is currently subject to preemption; otherwise false.

service_type
   service type of the user's terminal. It can have one of the following values:
     login
     FTP

term_id
user's terminal identifier code. It is "none" if the user's terminal does not have the answerback feature.

term_type
user's terminal type, which can be any terminal type name defined in the terminal type file described in MPM Communications I/O.

weight
loading factor system assumes for user's process.

SYNTAX AS A COMMAND:

```
verify strA strB
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[verify strA strB]
```

FUNCTION: returns an integer representing the first character position in strA that contains a character that does not occur anywhere in strB. If every character of strA occurs in strB, 0 is returned.

EXAMPLES: The following interactions illustrate the verify active function.

```
! string [verify chart chapter]
0
! string [verify chapter chart]
4
! string [verify 31 0123456789]
0
! string [verify 31q22 0123456789]
3
```

SYNTAX AS A COMMAND:

vfa path {-control_arg}

FUNCTION: adjusts structured files left in an inconsistent state by an interrupted opening, or unstructured files in any state.

ARGUMENTS:

path
is the pathname of a file to be adjusted.

CONTROL ARGUMENTS:
must be specified only for unstructured files and can be selected from the following:

-set_nl
if the last nonzero byte in the file is not a newline character, a newline character is appended. The bit count of the file's last nonempty segment is then set to the file's last nonzero byte (which is now sure to be a newline character).

-use_nl
the file is truncated after the last newline character.

-set_bc
the bit count of the file's last nonempty segment is set to the last nonzero byte in that segment. Any components beyond it are deleted.

-use_bc {N}
the file is truncated to the byte specified by the bit count of multisegment file component N. If N is not specified, it is taken to be the last nonempty component.

NOTES: For unstructured files a control argument must specify the desired adjustment. Otherwise, no control arguments are allowed. A sequential or blocked file is adjusted by truncation after the last complete record. An indexed file is adjusted by finishing the interrupted operation.

vfile_adjust (vfa)

See the description of the vfile_ I/O module (described in the MPM Subroutines) for further details. The adjust_bit_count command used with the character control argument is equivalent to vfile_adjust used with the -set_bc control argument, except that the latter only operates on a file that appears to be unstructured.

SYNTAX AS A COMMAND:

vfs path

FUNCTION: prints the apparent type (unstructured, sequential, blocked, or indexed) and length of files. For structured files, information about the state of the file (if busy) and the file version (unless current) is printed. The maximum record length is printed for blocked files. For indexed files, the following statistics are printed:

1. The number of records in the file, including zero length records.
2. The number of nonnull records in the file, if different from the above.
3. The total length of the records (bytes).
4. The number of blocks in the free space list for records.
5. The height of the index tree (equal to zero for empty files).
6. The number of nodes (each 1K words, page aligned) in the index tree.
7. The total length of all keys (bytes).
8. The number of keys (if different from record count).
9. The number of duplicate keys (if nonzero).
10. The total length of duplicate keys (if any).

ARGUMENTS:

path
is the pathname of a segment or multisegment file. If the entryname portion of a pathname denotes a directory, it is ignored. If no files are found for the given pathname, a message to that effect is printed. If the entry is a link, the information returned pertains to the entry to which the link points. The star convention is permitted.

NOTES: Additional information can be obtained through the status command.

Examples

Assume that the file foo is in the user's working directory. The command line:

```
! vfile_status foo
```

might produce the following output:

```
type:   unstructured
bytes:   4993
```

if the file is unstructured, or:

```
type:   sequential
records:   603
```

if the file is sequential, or:

```
type:   blocked
records:   1200
max recl:   7 bytes
```

if the file is blocked, or:

```
type:   indexed
records:   397
state:   locked by this process
action:   write in progress
record bytes:   3970
free blocks:   1
index height:   2
nodes:   3
key bytes:   3176
```

if the file is indexed and a write operation has been interrupted in the user's process.

volume_dump_switch_off (vdsf)

SYNTAX AS A COMMAND:

vdsf paths {-control_arg}

FUNCTION: turns off the specified volume dump switch of a segment.

ARGUMENTS:

paths
are the pathnames of segments whose dump switches are to be turned off. The star convention is accepted.

control_arg
can be one of the following:

CONTROL ARGUMENTS:

-incremental, -incr
turns off only the incremental dump switch.

-complete, -comp
turns off only the complete dump switch.

NOTES: If a switch is off, the segment is not dumped during that mode of volume dumping and thus may not be available for reloading and retrieval. Both volume dump switches (incremental and complete) are on by default for all segments when they are created. The user is cautioned not to turn off both volume dump switches unless the segment is easily re-createable, is a temporary segment that exists outside the process directory, or is privately backed up by some other means.

For information on turning a volume dump switch back on, see the volume_dump_switch_on command.

The two control arguments are mutually exclusive. If neither control argument is specified, both volume dump switches are turned off.

The states of the volume dump switches can be displayed using the status command.

volume_dump_switch_on (vdsn)

SYNTAX AS A COMMAND:

vdsn paths {-control_arg}

FUNCTION: turns on the specified volume dump switch of a segment.

ARGUMENTS:

paths
are the pathnames of segments whose dump switches are to be turned on. The star convention is accepted.

CONTROL ARGUMENTS:

-incremental, -incr
turns on only the incremental dump switch.

-complete, -comp
turns on only the complete dump switch.

NOTES: It a switch is not on, the segment is not dumped during that mode of volume dumping, and thus cannot be reloaded or retrieved. Both volume dump switches (incremental and complete) are on by default for all segments when they are created. Therefore, this command is only needed if the user had previously turned off one or both switches with the volume_dump_switch_off command.

The two control arguments are mutually exclusive. If neither control argument is specified, both volume dump switches are turned on.

The states of the volume dump switches can be displayed by the status command.

SYNTAX AS A COMMAND:

ws path command_line {-control_args}

FUNCTION: executes a specified command line in a specified directory (called the starting node) and in directories inferior to the starting node. The command prints the pathname of every directory in which the command line is executed.

ARGUMENTS:

path
is the starting node. This must be the first argument. A path of -wd or -working_directory specifies the working directory.

command_line
is the command line to be executed. The entire command line is taken to be a single argument. A multiple-word command line should be typed as a quoted string.

CONTROL ARGUMENTS:

-first N, -ft N
makes N the first level in the storage system hierarchy at which the command line is to be executed where, by definition, the starting node is level 1. The default is -ft 1.

-last N, -lt N
makes N the last level in the storage system hierarchy at which the command line is to be executed. The default is -lt 99999, i.e., all levels.

-brief, -bf
suppresses printing of the names of the directories in which the command line is executed.

-bottom_up, -bu
causes execution of the command line to commence at the last level and to proceed upward through the storage system hierarchy until the first level is reached. In the default mode, execution begins at the highest (first) level and proceeds downward to the lowest (last) level.

-priv
invokes a highly privileged primitive to list directories. Use of this control argument requires access to the hphcs_gate.

-msf
treats multisegment files as directories. Normally, multisegment files are not treated as directories.

NOTES: The walk_subtree command has a cleanup handler. If the user quits out of the command and immediately types release (rl) the user's directory is changed back to what it was prior to the invocation of walk_subtree.

EXAMPLES: To list all segments in the current working directory having a two-component name with a second component of pl1 the user types:

```
! ws -wd "list *.pl1"
```

To list two-component names with a second component of pl1 in directories subordinate to the working directory named George, the user types:

```
! ws >udd>m>George "list *.pl1" -all
```

SYNTAX AS A COMMAND:

wh names {-control_args}

FUNCTION: uses the standard search rules to search for a given segment or entry point.

names
are segment and entry point names. The star convention is NOT allowed.

CONTROL ARGUMENTS:

-all, -a
lists the pathnames of all segments and entry points with the specified names that can be found using the current search rules, the user's effective access to each segment or entry point, and the name of the search rule used to find each segment or entry point. (See "Examples" below.)

-entry_point, -ep
searches for entry points. If a name argument does not contain a dollar sign ($), the where command searches for the entry point name$name.

-segment, -sm
searches for segments. This is the default, unless name contains a dollar sign. (See "Notes" below.)

NOTES: The command prints out the full pathname of the segment, using its primary name and the entry point name if one is requested. If the segment or entry point is not in the search path, an error message is printed.

The primary name of a storage system entry is the name that is first in the list of names on that entry.

If the -all control argument is not specified, the where command prints information only about the first matching segment or entry point encountered (using the standard search rules).

The -entry_point and -segment control arguments are mutually exclusive. If one of these control arguments is used, all the name arguments are assumed to be of the type specified.

If neither the -entry_point nor -segment control argument is specified, the where command scans the name arguments. Any name arguments that contain a dollar sign are assumed to be names of entry points; all others are assumed to be names of segments.

For a discussion of search rules, see "Search Rules" in the MPM Reference Guide.

EXAMPLES: If a user has a private copy of the cwd command in the working directory, and that copy has been initiated, the command line:

```
! wh cwd -all
```

prints three lines:

```
>udd>Project_id>Person_id>wd>cwd
   (re) search rule "initiated_segments"
>udd>Project_id>Person_id>wd>cwd (re) search rule "wd"
>sss>cwd (re) search rule "system_library_standard"
```

where_search_paths (wsp)

SYNTAX AS A COMMAND:

wsp search_list entryname {-control_arg}

FUNCTION: returns the absolute pathname(s) of entryname when search list name and entryname are specified. The search for the entryname is made using the current search paths contained in the specified search list.

ARGUMENTS:

search_list
is the name of the search list searched.

entryname
is the entryname sought.

CONTROL ARGUMENTS:

-all, -a
specifies that all occurrences of this entryname found by searching this search list should be returned.

NOTES: For a complete list of the search facility commands, see the add_search_paths command description in this manual.

EXAMPLES: In the examples below, the exclamation mark is used to indicate the lines typed by the user. To find the include file struct.incl.pl1 using the translator search list, type:

```
! wsp translator struct.incl.pl1
  >user_dir_dir>Project_id>Person_id>struct.incl.pl1
```

If the -all control argument is specified, and there is more than one occurrence of the specified entryname:

```
! wsp translator struct.incl.pl1 -all
  >user_dir_dir>Project_id>Person_id>struct.incl.pl1
  >user_dir_dir>Project_id>include>struct.incl.pl1
  >library_dir_dir>include>struct.incl.pl1
```

SYNTAX AS A COMMAND:

```
who {-control_args} {-optional_args}
```

FUNCTION: lists the number, identification, and status of all users of the system; it prints out a header and lists the name and project of each user. The header consists of the system name, the total number of users, the current system load, the maximum load, the current number of absentee users, and the maximum number of absentee users. (See the description of the how_many_users command to print only the header.)

CONTROL ARGUMENTS:

-long, -lg
prints the date and time logged in, the terminal identification and the load units of each user, in addition to the user's name and project. The header includes installation identification and the time the system was brought up. If available, the time of the next scheduled shutdown, the time when service will resume after the shutdown, and the time of the previous shutdown are printed.

-project, -pj
sorts the output by the Project_id of each user.

-name, -nm
sorts the output by the name (Person_id) of each user.

-interactive, -ia
lists interactive users. See Notes.

-absentee, -as
lists absentee users. See Notes.

-daemon, -dmn
lists daemon users. See Notes.

-brief, -bf
suppresses the printing of the header.

LIST OF OPTIONAL ARGUMENTS:

Person_id
lists only users with the name Person_id.

.Project_id
lists only users with the project name Project_id.

Person_id.Project_id
lists only users with the name Person_id and the project name Project_id.

NOTES: If the control_args -interactive, -absentee, or -daemon are not specified, the default is to list all three types of users. If one or more of these control_args is specified, only users of the specified type(s) are listed.

Absentee users are denoted in the list by an asterisk (*) following Person_id.Project_id.

If the who command is specified with no arguments, the system responds with a two-line header followed by a list of interactive users sorted according to login time. (See "Examples" below.)

If the -project and -name control arguments are omitted, the output is sorted on login time. Both arguments cannot be used together, because the sort is performed on one key at a time.

If an optional_arg is specified, the header is suppressed even if the -long control argument is specified.

It is possible to prevent the user's own name from being listed; to do this, the user should first contact the project administrator.

EXAMPLES: To print default information, type:

```
! who
Multics 2.0, load 4.0/100.00; 4 users,.0, load 4.0/100.00;
   4 users, 2 interactive, 1 daemons.
Absentee users  1/2

IO.SysDaemon
Jones.Faculty
Doe.Work
Smith.Student*
```

To print long information for absentee users on the Student project (with no header), type:

```
! who -absentee -long .Student
  10/21/74  0050.2 none 1.0  Smith.Student*
```

To print brief information for all users, type:

```
! who -brief
  IO.SysDaemon
  Jones.Faculty
  Doe.Work
  Smith.Student*
```

working_dir (wd)

SYNTAX AS A COMMAND:

wd

SYNTAX AS AN ACTIVE FUNCTION:

[wd]

FUNCTION: returns the pathname of the working directory of the process in which it is invoked.

SYNTAX AS A COMMAND:

```
year {dt}
```

SYNTAX AS AN ACTIVE FUNCTION:

```
[year {dt}]
```

FUNCTION: returns the two-digit number of a year of the century from 01 to 99.

ARGUMENTS:

dt
is a date-time in a form acceptable to convert_date_to_binary_. If no argument is specified, the current year is used.

zero_segments (zsegs)

SYNTAX AS A COMMAND:

zsegs star_names {-control_arg}

SYNTAX AS AN ACTIVE FUNCTION:

[zsegs star_names {-control_arg}]

FUNCTION: returns the entrynames or absolute pathnames of segments with a zero bit count that match one or more star names.

ARGUMENTS:

star_name
is a star name to be used in selecting the names to be returned.

CONTROL ARGUMENTS:

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

NOTES: Only one name per segment is returned; i.e., if a segment has more than one name that matches star_name, only the first match found is returned.

Since each entryname (or pathname) returned by zero_segments is enclosed in quotes, the command processor treats each name as a single argument regardless of the presence of special characters in the name.

EXAMPLES: The following interaction illustrates the use of the zero_segments active function.

```
! pwd
  >udd>Apple>Jones
! ls -a

  Segments = 7, Lengths = 6.

  r w     0  empty_seg
  re      1  test
  r w     1  test.list
  r w     1  test.pl1
  re      1  prog
  r w     1  prog.list
  r w     1  prog.pl1

  Multisegment-files = 2, Lengths = 770.

  r w   513  prog.output
  r w   257  prog.data

  Directories = 2.

  sma  prog_stuff
  sma  documents

  Links = 3.

  prog.temp2                  >udd>Apple>Jones>temp_seg_2
  prog.temp1                  >udd>Apple>Jones>temp_seg_1
  junk                        >udd>Apple>Jones>empty_seg

! string [zsegs *]
  empty_seg
```

SECTION 4

ACCESS TO THE SYSTEM

This section describes the requests interpreted by the answering service. These requests can only be issued from a terminal connected to the answering service; that is, one that has just dialed up or one that has been returned to the answering service after a session terminated with a "logout -hold" command. (For more information on gaining access to the system, see "How to Access the Multics System" in the Multics Users' Guide, Order No. AL40.

For clarity, this section identifies two categories of answering service requests: preaccess and access. The preaccess requests are necessary because certain terminals do not have an answerback. By convention, Multics uses a terminal answerback to identify the particular type of device being used. The device type is used by the system to interpret all input/output. Therefore, for input to be understood by Multics and output understood by the user, these requests must be specified before the access requests. The access requests connect the terminal to a process. This process may exist already (e.g., dial) or be created in response to the request (e.g., login).

SYNTAX AS A COMMAND:

d dial_id {Person_id.Project_id}

FUNCTION: used to connect an additional terminal to an existing process. It is a request to the answering service to perform the connection and to notify the user's process of the new terminal connection.

ARGUMENTS:

dial_id
is the identifying keyword that uniquely specifies a logged-in process that is accepting dial connections. This keyword is supplied by that process when it informs the answering service that it is accepting dialed terminals.

Person_id.Project_id
is the Person_id and Project_id of the process that the user wishes to connect to. This argument is required only if the dial_id is not registered with the system.

NOTES: When the dial request is invoked, the answering service searches for a logged-in process that is accepting dial connections using the dial_id specified by the user. If no such process is found, the message "Dial line not active." is printed, and the user can try again, with a different dial_id. If a process is found, a one-line message verifying the connection is printed. All further messages printed on the terminal are from the user process itself.

This request is administratively restricted.

The Person_id.Project_id argument is optional if the dial_id is registered. All arguments must be supplied in the correct order.

If the user process terminates or logs out, a message is printed on the terminal, and control of the terminal is returned to the answering service.

Users who wish to accept dialed terminals must be registered with the dialok attribute, and so must their project. The

dialok attribute is normally assigned by the project administrator. Users who wish to accept dialed terminals without specification of the Person_id.Project_id must register the dial_id with the system administrator.

SYNTAX AS A COMMAND:

echo

FUNCTION: used to set the terminal into echoplex mode before login.

NOTES: This command is equivilent to:

modes echoplex

enter (e)
enterp (ep)

SYNTAX AS A COMMAND:

e {anonymous_name} Project_id {-control_args}
ep {anonymous_name} Project_id {-control_args}

FUNCTION: used by anonymous users to gain access to Multics. Either one is actually a request to the answering service to create a process for the anonymous user. Anonymous users who are not to supply a password use the enter (e) request. Anonymous users who are to supply a password use the enterp (ep) request.

ARGUMENTS:

anonymous_name
is an optional identifier that is not checked by the system, but is passed to the user's process overseer as if it were a person identifier. If anonymous_name is not specified, it is assumed to be the same as the project identifier.

Project_id
is the identification of the user's project.

CONTROL ARGUMENTS:

-brief, -bf
suppresses messages associated with a successful login. If the standard process overseer is being used, the message of the day is not printed.

-force
logs the user in if at all possible, provided the user has the guaranteed login attribute. Only system users who perform emergency repair functions have the necessary attribute.

-home_dir path, -hd path
sets the user's home directory to the path specified, if the user's project administrator allows that user specify a home directory.

-modes STR, -mode STR, -md STR
sets the I/O modes associated with the user's terminal to STR, where the string STR consists of modes acceptable to the tty_ I/O module. (See the tty_ I/O module description in the MPM

enter (e)
enterp (ep)

Subroutines for a complete explanation of possible modes.) The STR string is usually a list of modes separated by commas; the STR string must not contain blanks. (See "Examples" below.)

-no_preempt, -np
refuses to log the user in if login can be achieved only by preempting some other user in the load control group.

-no_print_off, -npf
causes the system to overtype a string of characters to provide a black area for typing the password.

-no_start_up, -ns
instructs the standard process overseer not to execute the user's start_up.ec segment, if one exists, and if the project administrator allows the user to avoid it.

-no_warning, -nw
suppresses even urgent system warning and emergency messages from the operator, both at login and during the user's session. Use of this argument is recommended only for users who are using a remote computer to simulate a terminal, or are typing out long memoranda, when the process output should not be interrupted by even the most serious messages.

-outer_module p, -om p
attaches the user's terminal via the outer module named p rather than the user's registered outer module, if the user has the privilege of specifying an outer module.

-print_off, -pf
suppresses overtyping for the password. The default is determined by the terminal type.

-process_overseer path, -po path
sets the user's process overseer to the procedure given by the path specified, if the user's project administrator allows that user to specify a process overseer. If path ends in the characters ",direct", the specified procedure is called directly during process initialization rather than by the init_admin procedure provided by the system. This means that the program specified by path must perform the tasks that would have been performed by the init_admin procedure.

-ring N, -rg N
sets the user's initial ring to be ring N, if this ring number

enter (e)
enterp (ep)

is greater than or equal to the user's registered initial ring and less than the user's registered maximum ring.

-subsystem path, -ss path
creates the user's process using the prelinked subsystem in the directory specified by path. The permission to specify a process overseer, which may be given by the user's project administrator, also governs the use of the -subsystem argument. To override a default subsystem by the project administrator, type -ss "".

-terminal_type STR, -ttp STR
sets the user's terminal type to STR, where STR is any terminal type name defined in the standard terminal type table. (To obtain a list of terminal types, refer to the print_terminal_types command.) This control argument overrides the default terminal type.

NOTES: If neither the -print_off nor -no_print_off control argument is specified at log-in, the system attempts to choose the option most appropriate for the user's terminal type.

If the project administrator does not allow the user to specify the -subsystem, -outer_module, -home_dir, -process_overseer, or -ring control arguments or if the administrator does allow one or more of these control arguments and they are incorrectly specified by the user, a message is printed and the login is refused.

SYNTAX AS A COMMAND:

hangup

FUNCTION: terminates communication between the terminal and Multics system. If the communication is via a dial-up phone line, the line is hung up. A user who is unable to log in can issue the hangup request as an alternative to manually hanging up the phone.

SYNTAX AS A COMMAND:

hello

FUNCTION: repeats the greeting message that is printed whenever a terminal is first connected to the system. The request is particularly useful after a 963 or 029 request since the greeting message is then printed in the proper code.

SYNTAX AS A COMMAND:

```
l Person_id {Project_id} {-control_args}
```

FUNCTION: used to gain access to the system. It is a request to the answering service to start the user identification procedure, and then either create a proces for the user, or connect the terminal to an existing disconnected process belonging to the user.

ARGUMENTS:

Person_id
is the user's registered personal identifier. This argument must be supplied. The personal identifier can be replaced by a registered "login alias" if the user has one. Aliases, like personal identifiers, are registered by the system administrator and are unique at the site. The login alias is translated into the user's personal identifier during the login process, and there is no difference between a user process created by supplying a personal identifier and one created by supplying an alias.

Project_id
is the identification of the user's project. If this argument is not supplied, the default project associated with the Person_id is used. See the -change_default_project control argument below for changing the default project to the Project_id specified by this argument.

CONTROL ARGUMENTS:
The following is an alphabetized listing of control_arg names. Complete description of these control arguments is provided in one of the three functional lists that appear below.

-authorization
-brief
-change_default_auth
-change_default_project
-change_password
-connect
-create
-destroy
-force
-generate_password
-home_dir
-list

-modes
-new_proc
-no_preempt
-no_print_off
-no_save_on_disconnect
-no_start_up
-no_warning
-outer_module
-print_off
-process_overseer
-ring
-save_on_disconnect
-subsystem
-terminal_type

LIST OF GENERAL CONTROL ARGUMENTS:
The following are permitted in any use of the login command:

-brief, -bf
suppresses messages associated with a successful login. If the standard process overseer is being used, the message of the day is not printed.

-change_default_auth, -cda
changes the user's registered default login authorization to the authorization specified by the -authorization control argument. If the authorization given by the user is valid, the default authorization is changed for subsequent logins, and the message "default authorization changed" is printed at the terminal. If the -cda control argument is given without the -auth argument, an error message is printed.

-change_default_project, -cdp
changes the user's default project to be the Project_id specified in this login request line (see the description of the Project_id argument above). The default Project_id is changed for subsequent logins, and the message "default project changed" is printed at the user's terminal. If the -cdp control argument is specified without a Project_id argument, an error message is printed.

-change_password, -cpw
changes the user's password to a newly given password. The login request asks for the old password before it requests the new one. It requests the new one twice, to verify the spelling. If it is not typed the same both times, the login and the password change are refused. If the old password is correct, the new password replaces the old for subsequent logins, and the message "password changed" is printed at the

user's terminal. The user should not type the new password as part of the control argument. Passwords can be up to eight characters long and can not contain imbedded blanks.

-generate_password, -gpw
changes the user's password to a new password, generated for the user by the system. The login request asks for the old password first. Then, a new password is generated and typed on the user's terminal. The user is asked to retype the new password, to verify having seen it. If the user types the new password correctly, it replaces the old password for subsequent logins, and the message "password changed" is printed at the user's terminal. If the user mistypes the new password, the login and password change are refused.

-modes STR, -mode STR, -md STR
sets the I/O modes associated with the user's terminal to STR, where the string STR consists of modes acceptable to the tty_ I/O module. (See the tty_ I/O module description in the MPM Subroutines for a complete explanation of possible modes.) The STR string is usually a list of modes separated by commas; the STR string must not contain blanks. (See "Examples" below.)

-no_print_off, -npf
causes the system to overtype a string of characters to provide a black area for the user to type the password.

-no_warning, -nw
suppresses even urgent system warning and emergency messages from the operator, both at login and during the user's session. Use of this argument is recommended only for users who are using a remote computer to simulate a terminal, or are typing out long memoranda, when the process output should not be interrupted by even the most serious messages.

-print_off, -pf
suppresses overtyping for the password. The default for this control argument depends on the terminal type.

-terminal_type STR, -ttp STR
sets the user's terminal type to STR, where STR is any terminal type name defined in the standard terminal type table. This control argument overrides the default terminal type.

LIST OF CONTROL ARGUMENTS FOR PROCESS CREATION:
The following arguments are to used when requesting the creation of a new process.

-authorization STR, -auth STR
sets the authorization of the process to that specified by STR, where STR is a character string composed of level and category names for the desired authorization, separated by commas. The STR character string cannot contain any embedded blank or tab characters. (The short names for each level and category are guaranteed to not contain any blanks or tabs, and can be used whenever the corresponding long names do contain blanks or tabs.) The STR character string must represent an authorization that is less than or equal to the maximum authorization of Person_id on the project Project_id. If this control argument is omitted, the user's registered default login authorization is used. (See "Access Control" in the MPM Reference Guide for more information about process authorizations.)

-force
logs the user in if at all possible, provided the user has the guaranteed login attribute. Only system users who perform emergency repair functions have the necessary attribute.

-home_dir path, -hd path
sets the user's home directory to the path specified, if the user's project administrator allows this choice.

-no_save_on_disconnect, -nosave
causes the user's process to be logged out instead of being saved, if it becomes disconnected from its login terminal. This argument is used to override a default of -save_on_disconnect, if that default has been set by the user's project administrator.

-no_preempt, -np
refuses to log the user in if this can only be done by preempting some other user in this user's load control group.

-no_start_up, -ns
instructs the standard process overseer not to execute the user's start_up.ec segment, if one exists, and if the project administrator allows this choice.

-outer_module path, -om path
attaches the user's terminal via the outer module named path rather than the user's registered outer module, if the user is allowed this choice.

-process_overseer path, -po path
sets the user's process overseer to the procedure given by the path specified, if the user's project administrator allows this choice. If path ends in the characters ",direct", the

specified procedure is called directly during process initialization rather than by the standard procedure provided by the system. This means that the program specified by path must perform the tasks that would have been performed by the standard procedure.

-ring N, -rg N
sets the user's initial ring to be ring N, if this ring number is greater than or equal to the user's registered initial ring and less than the user's registered maximum ring.

-save_on_disconnect, -save
saves the user's process if it becomes disconnected from its login terminal because of a communications line hangup or FNP crash. Permission to use the process-saving facility, and the setting of whether or not the facility is enabled by default, are both under the control of the user's project administrator. See the description of the no_save_on_disconnect command elsewhere in this manual.

-subsystem path, -ss path
creates the user's process using the prelinked subsystem in the directory specified by path. The permission to specify a process overseer, which can be given by the user's project administrator, also governs the use of the -subsystem argument. To override a default subsystem specified by the project administrator, type -ss "".

LIST OF CONTROL ARGUMENTS FOR DISCONNECTED PROCESSES: The following are used to specify the disposition of disconnected processes. See "Notes on Disconnected Processes" below.

-connect {N}
connects the terminal to the user's disconnected process. If more than one such process exists, the process number N must be specified.

-new_proc {N}
destroys the user's disconnected process and creates a new one. If more than one such process exists, the process number N must be specified.

-destroy {N}
destroys the user's disconnected process and logs out. If more than one such process exists, the process number N must be specified.

-create
creates a new process without destroying any disconnected

processes. This is permitted only for users who are allowed to have multiple interactive processes.

-list
lists the user's disconnected process, showing the process number, the time of the original login, and the ID of the channel and terminal that were last connected to the process.

NOTES: The login request asks for a password from the user (and attempts to ensure either that the password does not appear at all on the user's terminal or that it is thoroughly hidden in a string of cover-up characters). The password is a string of one to eight letters and/or digits associated with the Person_id.

After the user responds with the password, the answering service looks up the Person_id, the Project_id, and the password in its tables and verifies that the Person_id is valid, that the Project_id is valid, that the user is a legal user of the project, and that the password given matches the registered password. If these tests succeed, the user is said to be logged in.

Unless the user already has one or more processes, the normal action taken at login is to attempt to create a process. First, the load control mechanism is consulted to determine if creation of a process for this user would overload either the system or the user's load control group.

If permitted by load control, a process is created for the user and the terminal is connected to it (i.e., the terminal is placed under the control of that process).

The control arguments described above under "List of Process Creation Control Arguments" can be used to specify some of the attributes of a newly-created process.

The user might have a disconnected process (one that became disconnected from its terminal because of a phone line hangup or an FNP crash). In that case, the user can choose among the following alternatives: connection of the terminal to the process; destruction of the disconnected process, with or without the creation of a new one; or logging out without affecting the disconnected process. These alternatives are

described below under "Notes on Disconnected Processes". The arguments used to select them are described above under "List of Disconnected Process Control Arguments".

If neither the -print_off nor -no_print_off control argument is specified at log-in, the system attempts to choose the option most appropriate for the user terminal type.

Several parameters of the user's process, as noted above, can be controlled by the user's project administrator. The project administrator can allow the user to override some of these attributes by specifying control arguments in the login line.

If the project administrator does not allow the user to specify the -subsystem, -outer_module, -home_dir, -process_overseer, -save_on_disconnect, or -ring control arguments or if the administrator does allow one or more of these control arguments and they are incorrectly specified by the user, a message is printed and the login is refused.

NOTES ON DISCONNECTED PROCESSES: If a user's project administrator allows it, a user's process can be preserved when it becomes disconnected from its terminal because of a phone hangup or FNP crash. The user can call back any time before the (installation-defined) maximum inactive time and ask to be connected to this disconnected process. This feature is controlled by the -save_on_disconnect and -no_save_on_disconnect control arguments; the default is set by the user's project administrator.

Some users are permitted by their project administrators to have several interactive processes simultaneously. These users can have more than one disconnected process. Multiple disconnected processes are numbered consecutively starting with 1, in the order of their login times. These process numbers must be used as arguments when referring to one of a set of multiple disconnected processes. The number and login time of each is printed by the -list argument or the list preaccess request. The user can, however, anticipate the process numbering and use a number in an argument to the login command. The time listed and sorted on is the time of the original login from which the process is descended; this time is not affected by new_proc or reconnection.

A user with disconnected processes who does not specify, on the login line, the action to be taken with respect to the disconnected processes, is told of the existence of the disconnected processes and given a choice of the following actions:

1) list the user's disconnected processes;
2) create an additional process;
3) connect the terminal to a disconnected process;
4) destroy a disconnected process, create a new one with the same attributes, and connect the terminal to it;
5) destroy a disconnected process and log out;
6) log out without affecting any process.

These are specified by means of the list, create, connect, new_proc, destroy, and logout preaccess requests. The connect, new_proc, and destroy requests take an optional process number as an argument. The help request, when issued from a logged in but disconnected terminal, explains these options rather than explaining how to log in.

EXAMPLES: In the examples below, the lines typed by the user are preceded by an exclamation mark (!) and the user's password is shown even though in most cases the system either prints a string of cover-up characters to "hide" the password or temporarily turns off the printing mechanism of the user's terminal.

Probably the most common form of the login request is to specify just the Person_id and the Project_id (and then the password) as:

```
! login Jones Demo
  Password:
! mypass
```

To set (or change) the default project to Demo, type:

```
! login Jones Demo -cdp
  Password: ! mypass
  Default project changed.
```

To set the tabs and crecho I/O modes so the terminal uses tabs rather than spaces where appropriate on output and echoes a

carriage return when a line feed is typed (assuming the user has a default project), type:

```
! login Jones -modes tabs,crecho
  Password:
! mypass
```

To change the password from mypass to newpass (assuming the user has a default project), type:

```
! login Jones -cpw
  Password:
! mypass
  New Password:
! newpass
  New Password Again:
! newpass
  Password changed.
```

The following example illustrates a login involving a disconnected process.

```
! login Jones.Demo
  Password:
! mypass

  You have 1 disconnected process.
  Jones.Demo logged in 11/16/79  1435.9 est Fri
     from ROSY terminal "none"
  Last login 11/16/79  1435.1 est Fri
     from ROSY terminal "none"
  Please give instructions regarding your
     disconnected process(es).
  Please type list, create, connect, new_proc, destroy,
     logout, or help.

! list
  1)  logged in 11/16/79  1435.1 est Fri over channel a.h001,
        terminal "none"
  Please type list, create, connect, new_proc, destroy,
     logout, or help.

! connect
  Your disconnected process will be connected to this terminal
  Wait for QUIT.
  QUIT
  r 1503:03 .47 12 Level 2
```

SYNTAX AS A COMMAND:

```
logout {-control_args}
```

FUNCTION: terminates a user session and ends communication with the Multics system. It is used from a terminal that is logged in but not connected to a process. It informs the answering service that the user who gave a correct Person_id password combination is no longer using the terminal.

CONTROL ARGUMENTS:

-hold, -hd
the user's session is terminated. However, communication with the Multics system is not terminated, and a user can immediately log in without redialing.

-brief, -bf
no logout message is printed, and if the -hold control argument has been specified, no login message is printed either.

SYNTAX AS A COMMAND:

```
modes mode_string
```

FUNCTION: used to set terminal modes before login.

ARGUMENTS:

mode_string
is a list of modes to be set.

SYNTAX AS A COMMAND:

slave

FUNCTION: changes the service type of the channel from login to slave for the duration of the connection.

NOTES: The slave command enables a privileged process to request the answering service to assign the channel to it, and then attach it. Refer to the description of the dial_manager_ subroutine in the MPM Subsystem Writers' Guide for an explanation of the mechanism for requesting channels from the answering service.

terminal_type (ttp)

SYNTAX AS A COMMAND:

ttp terminal_type_name

FUNCTION: used to set the terminal type prior to login.

ARGUMENTS:

terminal_type_name
is the name of a system defined terminal type.

SYNTAX AS A COMMAND:

MAP

FUNCTION: tells the system that the user is attempting to gain access from a terminal whose keyboard generates only uppercase characters. This request must be invoked before the access requests (e.g., login) can be successfully issued.

NOTES: Once the request has been issued, the system changes the translation tables used by the terminal control software so that all uppercase alphabetic characters are translated to lowercase. The user still needs to use the special escape conventions to represent the ASCII graphics that are not on the uppercase-only terminal keyboard. Uppercase alphabetic characters also require the escape conventions. (See "Escape Characters" in the MPM Reference Guide.) After the map request is given, the user may log in normally.

This request must be used for 150-, 300-, and 1200-baud terminals if their keyboards can transmit only uppercase characters; for any other terminal type, it is ignored.

EXAMPLES: The following example shows a user invoking the MAP request. The lines typed by the user are preceded by an exclamation mark (!).

```
! MAP ! LOGIN \JONES \DEMO
  PASSWORD: ! MYPASS
```

029 and 963

SYNTAX AS A COMMAND:

029

or:

963

FUNCTION: requests tell the system whether the user is attempting to gain access from a device similar to an EBCDIC or Correspondence code IBM Model 2741. These requests must be invoked before the access requests (e.g., login) can be successfully issued.

NOTES: If the user attempts to log in from a device similar to an EBCDIC or Correspondence code IBM Model 2741, the system returns a "Type 'help' for instructions" message accompanied by a partially readable line. For example,

cidu #63 cqn U:XVOXK Type 029 for Correspondence code.

or

Type 963 for EBCDIC. Ula; z17 qis Fiss;nairp;rf; fip;-

The user should respond to this message by typing the specified request.

Once the request has been issued, the system changes the translation tables used by the terminal control software so that all input/output is readable. The user can then log in normally.

These requests are valid for 134-baud devices similar to an IBM Model 2741 only; for any other terminal type, they are ignored.

The names (963, 029) of the requests are actually the standard part numbers of the usual typeballs for EBCDIC and Correspondence code IBM Model 2741s, respectively.

APPENDIX A

OBSOLETE COMMANDS

This appendix contains selected Multics system commands that are obsolete. They appear in alphabetical order for user convenience.

SYNTAX AS A COMMAND:

fs_chname dir_name entryname oldname newname

FUNCTION: The fs_chname command is an interface to the storage system subroutine hcs_$chname_file (described in the MPM Subroutines). It causes an entryname of a specified segment to be replaced, deleted, or added.

ARGUMENTS:

dir_name
is the directory name portion of the pathname of the segment in question.

entryname
is the entryname portion of the pathname of the segment in question.

oldname
is an old entryname to be deleted. See "Notes" below.

newname
is a new entryname to be added. See "Notes" below.

ACCESS REQUIRED: The user must have modify permission on the directory containing the entry in order to make any name changes.

NOTES: This command interprets none of the special command system symbols (e.g., *, >) and thus allows the user to bypass the star convention or to manipulate strangely named segments. For segments with ordinary names, the rename, add_name, and delete_name commands perform the same function.

Since the -name control argument of the rename command allows the user to ignore special command system symbols and thus give a strangely named entry an ordinary name, the fs_chname command is now obsolete. It is documented here for compatibility and will be removed at the next update.

When both an old entryname and a new entryname appear in the command line, the new entryname replaces the old entryname. This is equivalent to using the rename command.

If the old entryname is a null character string (""), then the new entryname is added to the segment. This is equivalent to using the add_name command.

If the new entryname is a null character string (""), then the old entryname is deleted from the segment. This is equivalent to using the delete_name command.

SYNTAX AS A COMMAND:

```
ml path Person_id1.Project_id1 ...  {Person_idn.Project_idn}
     {-control_args}
```

or:

```
ml {destination} {-control_args}
```

FUNCTION: sends a message to another user or prints messages in any mailbox to which the user has sufficient access.

ARGUMENTS:

path
is the pathname of a segment to be sent or is an asterisk (*) to indicate that the user wishes to type a message to be sent (see "Notes on Composing Mail" below).

Person_idi
is the person name of a person to whom mail is to be sent.

Project_idi
is the name of a project on which Person_idi is registered.

Mail is sent to the mailbox >udd>p3Project_idi>Person_idi>Person_idi.mbx for each Person_idi.Project_idi argument in the command line.

CONTROL ARGUMENTS:

-acknowledge, -ack
requests acknowledgement of the pieces of mail. The acknowledgement consists of the string:

"Acknowledge message of <date-time sent>"

and is sent as an interactive message by the mail command when the mail command is invoked to print mail.

-no_notify, -nnt
suppresses the sending of an interactive "You have mail" notification.

-pathname path, -pn path
specifies a mailbox by pathname. The mbx suffix is assumed.

To print messages sent by the mail and send_message commands:

Syntax: ml {destination} {-control_args}

ARGUMENTS:

destination
can be of the form Person_id.Project_id to specify a mailbox. The default is the user's default mailbox. If destination contains a < or >, it is the pathname of a mailbox. The mbx suffix is assumed in this case.

CONTROL ARGUMENTS:

-header, -he
prints only the header line for each message. No messages are deleted.

-match STR
prints messages sent by users whose Person_id.Project_id matches the Person_id.Project_id specified in STR. If the -exclude control argument has been specified, exclusion is performed before matching. The star convention is allowed.

-exclude STR, -ex STR
ignores messages sent by users whose Person_id.Project_id matches the Person_id.Project_id specified in STR. The star convention is allowed. If the -match control argument has been specified, exclusion is performed before matching.

-brief, -bf
prints the total number of messages in the mailbox. If the mailbox is empty, nothing is printed.

-pathname path, -pn path
specifies a mailbox by pathname. The mbx suffix is assumed. This control argument cannot be used with the destination argument.

NOTES ON PRINTING MAIL: When the contents of the mailbox named by path are printed, they are preceded by a line of the form:

N messages.

Each message is preceded by a line of the form:

i) From: Person_id.Project_id (sent_from) date time (N lines)

where:

i
is the incremental number of the message. The oldest message is numbered 1, the next oldest is 2, and so on. The messages are printed in ascending numerical order.

Person_id
is the registered person identifier of the user who sent the message.

Project_id
is the name of the project on which the sender was logged in when he sent the message.

sent_from
is an optional field that further identifies the sender, e.g., the log-in name of an anonymous user, if he has one.

date
is the date the message was sent, of the form mm/dd/yy to indicate the month, day, and year.

time
is the time the message was sent, of the form hhmm.m zzz www to indicate the hours, minutes, and tenths of minutes in 24-hour time followed by the time zone and day of the week.

N lines
is the number of lines in the message.

After printing all messages, the mail command asks whether the user wants the messages deleted. If the answer is yes, all messages in the mailbox are deleted. If the answer is no, no messages are deleted. In either case, the user returns to command level.

If the user issues a quit signal while the messages in the mailbox are being printed and then issues the program_interrupt command, the mail command stops printing and asks whether to delete all messages in the mailbox, including those that were not printed.

NOTES: The extended access used on mailboxes, which are ring 1 segments, permits the creator of a mailbox to control other users' access to it. Adding, reading, and deleting messages are independent privileges under extended access. For example, one user can be given access to only add messages, another user to add messages and to read and delete only the messages he or she has added. For more information on extended access, see "Notes on Extended Access" below. Mail and interactive messages sent to a user are placed in the mailbox: >udd>Project_id>Person_id>Person_id.mbx. For more information on the mail facility, see the print_mail, read_mail, and send_mail command descriptions in this manual.

If the recipient of mail is accepting interactive messages (see accept_messages in this manual), the recipient receives an immediate notification of the form "You have mail from Person_id.Project_id."

Segments to be mailed have a maximum length of one record (4096 ASCII characters).

NOTES ON COMPOSING MAIL: If path is *, mail responds with "Input:" and accepts lines from the terminal until a line consisting only of a period (.) is typed. The typed lines are then sent to the specified user(s).

NOTES ON CREATING A MAILBOX: A default mailbox is created automatically the first time a user prints mail or issues the accept_messages or print_messages commands. The default mailbox is:

>user_dir_dir>Project_id>Person_id>Person_id.mbx

NOTES ON EXTENDED ACCESS: Access on a newly created mailbox is automatically set to adrosw for the user who created it, aow for *.SysDaemon.*, and aow for *.*.*. The types of extended access for mailboxes are:

add	a	add a message.
delete	d	delete any message.
read	r	read any message.
own	o	read or delete only your own messages, i.e., those sent by you.
status	s	find out how many messages are in the mailbox.
wakeup	w	send a wakeup when adding a message (used by the send_message command).

The modes "n", "null", and "" specify null access.

LIST OF RELATED COMMANDS: Special commands exist to create additional mailboxes and to change the attributes of mailboxes. These commands, described in the MPM Subsystem Writers' Guide, are:

mbx_create	create a mailbox.
mbx_delete	delete a mailbox.
mbx_add_name	add a name to a mailbox.
mbx_delete_name	delete a name from a mailbox.
mbx_rename	rename a mailbox.
mbx_list_acl	list the access control list (ACL) of a mailbox.
mbx_set_acl	change or add entries to the ACL of a mailbox.
mbx_delete_acl	delete entries from the ACL of a mailbox.
mbx_set_max_length	set the maximum length of a mailbox.
mbx_safety_switch_on	turn on the safety switch of a mailbox.
mbx_safety_switch_off	turn off the safety switch of a mailbox.

print_translator_search_rules
(ptsr)

SYNTAX AS A COMMAND:

ptsr

FUNCTION: prints the current translator search list used by language translators (pl1, fortran, basic, etc.) to find include files.

NOTES: This command has the same effect as "print_search_paths translator". It is recommended that the print_search_paths command be used. The synonym for translator search list is trans.

set_translator_search_rules
(stsr)

SYNTAX AS A COMMAND:

stsr {paths}

FUNCTION: manipulates the translator search list used by languages to find include files.

ARGUMENTS:

paths
are the pathnames of directories to be searched, in the order given, when searching for an include file.

NOTES: The set_translator_search_rules command is identical to "set_search_paths translator". It is recommended that the add_search_paths or set_search_paths commands be used. The synonym for translator is trans.