HONEYWELL

MULTICS COMMON COMMANDS

SOFTWARE

MULTICS COMMON COMMANDS

SUBJECT

Description of Commonly Used Multics Commands

SPECIAL INSTRUCTIONS

This document describes selected Multics commands for use in performing commonly employed procedures. The commands in this manual represent a subset of the complete command list that is described in the Multics Commands and Active Functions manual Order No. AG92.

SOFTWARE SUPPORTED

Multics Software Release 10.1

ORDER NUMBER

GB58-00

February 1983

Honeywell

PREFACE

This document contains selected Multics commands that implement frequently used procedures. This document is designed for users who require concise reference information for basic programming tasks such as system access, file/directory creation and management, I/O operations, absentee usage, use of display and reporting capabilities, and protection implementation. Also provided are descriptions of tools for sending messages and mail, obtaining accounting information, and using file/directory search facilities. In certain instances, the information associated with a particular command has been edited to delete descriptions of options or services not commonly used. In those cases, the user is referred to the appropriate manual (or on-line information segment) where the complete reference text can be found.

The Multics Commands and Active Functions manual (Order No. AG92) contains an exhaustive list of programming commands and constitutes the central reference work for standard Multics programming functions. Programmers should refer to the Commands and Active Functions manual for information on the complete set of features and tools available for use. The Common Commands manual represents an alternative source of information for programmers who desire basic information only.

File No.: 1L13, 1U13
GB58-00

CONTENTS

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 commands described in this manual. Section 3 contains an alphabetized listing of the standard Multics system commands and active functions.

Each command 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

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, an "s" is added to the argument name (e.g., `paths`, `{paths}`, `{-control_args}`).

GENERAL DEFINITION OF A COMMAND

A command performs some action for a user, such as displaying information on the user's terminal, formatting a report, or compiling a program. 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. Commands are invoked at the beginning of a command line at command level, or multiple commands can appear on a single line, with an unquoted semicolon (;) as a delimiter between each one.

GENERAL DEFINITION OF AN ACTIVE FUNCTION

An active function is most frequently used to shorten the amount of typing required to invoke a command. 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 multiple commands are specified on a line, active functions in each are expanded before execution. This means that the first command is executed before active functions in the second command invocation are expanded. Therefore, the execution of a command can affect the values of active functions which appear later in the line.

EXAMPLES OF COMMAND VS. ACTIVE FUNCTION USE

Many programs can be invoked as either a command or active function. The format of the active function return string is 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
```

or:

```
! format_line "^(^a ^)" [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 Programmers' Reference Manual.

SECTION 2

FUNCTIONAL GROUPING OF COMMANDS AND ACTIVE FUNCTIONS

The Multics commands and active functions are presented in this section, referenced by functional use. Descriptions appear in sections 3 in alphabetical order.

FUNCTIONAL HEADINGS OF COMMANDS

ACCESS TO THE SYSTEM

login
logout

USE OF FILES/DIRECTORIES

add_name
add_search_paths
change_wdir
copy
copy_dir
create
create_dir
delete
delete_acl
delete_dir
delete_name
delete_search_paths
delete_search_rules
link
list
list_acl
move
print_search_paths
print_search_rules
rename
set_acl
set_search_paths
set_search_rules
status
switch_on
switch_off
unlink
working_dir

ABSENTEE PROCESSING

enter_abs_request
list_abs_request
cancel_abs_request

MESSAGES/MAIL

accept_messages
defer_messages
delete_message
immediate_messages
last_message
last_message_sender
print_messages
read_mail
send_mail
send_message

INPUT/OUTPUT

cancel_output_request
discard_output

enter_output_request
file_output
list_output_requests
move_output_request

TEXT EDITORS

emacs
qedx

ON-LINE MEETING SUBSYSTEM

forum

ONLINE HELP FACILITY

help

LISTING INFORMATION

check_info_segs
date
get_quota
how_many_users
list
list_abs_request
list_output_requests
print
print_motd
print_request_types
print_search_paths
print_search_rules
print_terminal_types
ready
reprint_error
status
who
working_dir

CREATING FORMATTED DOCUMENTS

fdoc

USER ENVIRONMENT CONTROL

abbrev
do
exec_com

TERMINAL SUPPORT

set_tty

PROCESS TERMINATION/RESTART

new_proc
program_interrupt
release
start

MISCELLANEOUS TOOLS

adjust_bit_count
archive
calc
memo
program_interrupt
query
rady_off
ready_on

SECTION 3

COMMANDS AND ACTIVE FUNCTIONS

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

Name: abbrev, ab

SYNTAX AS A COMMAND

```
ab
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ab]
```

FUNCTION

provides the user with a mechanism for abbreviating parts of (or whole) command lines in the normal command environment. As an active function, returns "true" if abbreviation expansion of command lines is currently enabled and "false" otherwise. Once the abbrev command is invoked, you are placed in the abbrev subsystem, where you must use abbrev requests.

NOTES

The abbrev command uses 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
```

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 Programmers' Reference Manual 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 represent a subset of the total requests; see the Commands and Active Functions manual for the complete list of requests.

`.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.

.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.

.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.

.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.

print "abbrev" followed by the current version number of the abbrev processor.

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 wishes to abbreviate the pathname of a directory in which he does a lot of his work. Instead of having to type the entire pathname every time he needs to reference it, it can be called up easily with much fewer keystrokes as in the following examples:

Invoke the abbrev command:

```
ab
```

Define the abbreviation:

```
.a myinfo >udd>States>Washington>info
```

Now that "myinfo" is defined, the user can change to that directory simply by:

```
cwd myinfo
```

Change to the inferior directory called data_dir by:

```
cwd myinfo>data_dir
```

Another useful abbreviation is for the enter_output_request command, when the user frequently uses a certain printer queue and a special request type (see the enter_output_request command). For example:

```
.ab printx eor -q 2 -rqt x1200 -nt -he "By George"
```

Now to request a printout of a segment contained in "myinfo", simply type:

```
printx myinfo>data.list
```

Name: accept_messages, am

SYNTAX AS A COMMAND

```
am {address}
```

FUNCTION

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

ARGUMENTS

address
is the address of a mailbox. If no address is specified, the user's default mailbox is assumed. The mailbox must be specified in one of the following forms:

STR
is any argument that does not begin with a minus sign (-). If it contains either of the characters > or < it is interpreted as a mailbox pathname (the .mbx suffix is added if not present); otherwise it is interpreted as a User_id.

-pathname PATH, -pn PATH
specifies the pathname of the mailbox. The .mbx suffix is assumed if it is not present.

CONTROL ARGUMENTS

A complete list of control arguments can be found in the Commands and Active Functions manual.

NOTES

A default mailbox is created automatically the first time a user issues either print_mail, read_mail, accept_messages, or print_messages. The default mailbox is:

```
>udd>Project_id>Person_id>Person_id.mbx
```

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.

It is not advisable for several users to share the same mailbox.

At any time, only one process can be accepting messages from a given mailbox. If a user creates two processes and both processes accept messages from the same mailbox, the second process (i.e., the one issuing an am command most recently) will

automatically take over the accept_messages function. The first process will receive no indication that messages are being routed to the second process.

If the second process logs out or is destroyed, the messages do not revert to an earlier process. Thus, a user who sends a message to that mailbox will be informed that the addressee is currently not accepting messages or is not logged in.

A user who is registered on multiple projects using a common mailbox should be aware that this behavior will affect his processes.

Users should generally not accept messages in absentee processes. For example, the start_up.ec should distinguish between interactive and absentee process, and only issue the accept_messages command in the former case.

Name: add_name, an

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. The star convention is allowed.

names
: are additional names to be added. The equal convention is allowed.

ACCESS REQUIRED

modify on the parent directory.

NOTES

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.

Name: add_search_paths, asp

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_path1 is a relative or absolute pathname or a keyword. (For a list of acceptable keywords see "List of Keywords" below.) Each search_path argument 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 after the search_path argument. Only one is allowed for each search_path.

-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 (psp) command is invoked.

-before STR, -be STR
specifies that the new search path is positioned before the STR search path.

-first, -ft
specifies that the new search path is positioned as the first search path in the search list.

-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.

LIST OF RELATED SEARCH FACILITY 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.

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.

Name: adjust_bit_count, abc

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).

ARGUMENTS

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 chase links only for non-starred pathnames.

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

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.

Name: archive, ac

SYNTAX AS A COMMAND

```
ac operation archive_path paths
```

FUNCTION

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

ARGUMENTS

operation
is one of the functions listed below under "List of Operations."

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. 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. The star and equal conventions cannot be used. For append, replace, update and extract operations, each path specifies the pathname of a segment corresponding to a component whose name 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 maintiain existing ones. The operations are:

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. 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. 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.

ad
append and delete; operates like a and then deletes all segments that have been appended to the archive.

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.

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.

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 Operation

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 no component names are given, all components are extracted and placed in segments in the working directory. The archive segment is not modified.

xd
extract and delete; operates like x but deletes the component from the archive if it is extracted successfully.

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

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.

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 during extract and table of contents operations; it cannot be used during append, replace, update, and delete operations.

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.

Each component of an archive segment retains certain attributes of the segment from which it was copied. These consist of one name, the effective mode of the user who placed the component in the archive, the date-time last modified, the bit count, and the date-time placed in the archive. 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 use exec_coms to update archives should be aware of this limitation.

Date-time values are stored in ASCII without a time zone. The time is expressed relative to the time zone set for the process that placed the component in the archive. If the time zone set during the archive update operation differs from the zone set when the component was first archived, the update will not be performed correctly. This can cause a component to be updated needlessly, or prevent a component from being updated even though changes were made to its corresponding segment. The time zone of a process can be changed via the set_time_zone command.

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 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.

EXAMPLES

Assume that the user has several short segments and wants 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 operation) 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 operation) 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_id>dir_one>greek.archive

archive d greek gamma
```

A table of contents new 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_id>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

nl 3
```

Name: calc

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 must be quoted if it contains spaces or other command language characters. Variables are not allowed.

LIST OF REQUESTS

print "calc".

`..STR`
: execute the Multics command line STR.

`<expression>`
: type value of expression.

`<variable>=<expression>`
: assign value of expression to variable.

`list`
: list variables.

`q`
: return to command level.

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.

OTHER REQUESTS

Typing "." on a line by itself causes calc to identify itself by printing "calc".

Typing a line beginning with two periods ".." causes the remainder of the line to be passed to Multics as a command line, and executed.

Typing "q" causes calc to return to the calling program, i.e., to command level.

Name: cancel_abs_request, car

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 requests to be cancelled. The star convention is allowed.

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

-id ID
identifies one or more requests to be cancelled by request identifier. This identifier can be used to further define any path or -entry identifier (see "Notes").

CONTROL ARGUMENTS

-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.

-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 defined 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. If the -queue, -fg, and -all control arguments are omitted, only the default priority queue is searched.

-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 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

If the -id control argument is specified, only one path or -entry control argument is allowed. If the -id control argument is given in addition to a path or -entry control argument, they must match the same request. If any path or -entry STR request identifiers are given, only one -id ID request identifier will be accepted and it must match any requests selected by path or entryname.

Multiple -id ID identifiers can be specified in a single command invocation only if NO path or entry request identifiers are given.

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. When a running absentee process is cancelled by a user or an operator, the message "Process terminated by the system. The reason will be sent by Multics mail." will appear in the absentee output segment.

EXAMPLES

The command line:

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

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

Name: cancel_output_request, cor

SYNTAX AS A COMMAND

```
cor 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
identifies a request to be cancelled by the full or relative pathname of the input data segment. The star convention is allowed.

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

-id ID
identifies one or more requests to be cancelled by request identifier. This identifier may be used to further define any path or -entry identifier (see "Notes").

CONTROL ARGUMENTS

-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.

-queue N, -q N
specifies that queue N of the request type contains the requests 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.

-print, -pr
specifies that the requests to be cancelled are found in the queue(s) associated with the default printer request type. See Notes below.

-punch, -pch
specifies that the requests to be cancelled are found in the queue(s) associated with the default punch request type. See Notes below.

-plot
specifies that the requests to be cancelled are found in the queue(s) associated with the default plotter request type. See Notes below.

-request_type STR, -rqt STR
indicates that the requests to be to be cancelled is to be found in the queue for the request type identified by the string STR. See Notes below.

-user User_id
specifies the name of the submitter of the requests 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

Users 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

Multiple -id ID identifiers can be specified in a single command invocation only if NO path or entry request identifiers are given.

If any path or -entry STR request identifiers are given, only one -id ID request identifier will be accepted and it must match any requests selected by path or entryname.

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.

If the request is already running, the entry is still removed from the queue but the running request is not stopped. However, the user is given a message stating that the request is running.

When a request has been removed from the queue after it has started running and before it has finished, any user requested deletion of the segment (done with the -delete control argument to the enter_output_request command) will be ignored by the system.

The -print, -punch, -plot and -request_type control arguments are mutually exclusive. Only one may be used in a given command. If none are given, then cor searches the

default request type used by eor -print (as displayed by the print_request_types command).

See also the description of the enter_output_request command.

Name: 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.

Name: check_info_segs, cis

SYNTAX AS A COMMAND

```
cis {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[cis {-control_args}]
```

FUNCTION

prints a list of info segments modified since a given time.

CONTROL ARGUMENTS

-absolute_pathname, -absp
prints or returns absolute pathnames of segments rather than entrynames.

-brief, -bf
does not print the names of changed info segs or the "No change" message. This control argument is intended for use with -call, and cannot be used with the cis active function.

-call cmdline
calls the command processor with "cmdline path" for each changed segment; path is the absolute pathname of a changed segment. If cmdline contains blanks, it must be enclosed in quotes. This control argument cannot be used with the cis active function.

-date DT, -dt DT
uses the date DT instead of the date in the user's default value segment. The date in the value segment is not updated. DT must be acceptable to convert_date_to_binary_.

-long, -lg
prints the date-time-entry-modified as well as the segment name. This control argument cannot be used with the cis active function.

-no_update, -nud
does not update the date in the user's value segment.

-pathname star_path, -pn star_path
star_path is a pathname with a star name in the entryname portion. All segments that match star_path are checked. More than one -pathname control argument can be given. If none are given, the directories in the "info_segments" search list, which has synonyms "info_segs" and "info" are used; the starname **.info is used as the entryname.

-time_checked, -tmck
prints the date_time that is stored in the user's default value segment indicating from when checking of modified info segments would occur if the -date control argument were not used. This control argument is incompatible with all others when used with the cis active function. It does not update the time in the user's value segment when used as the only control argument.

NOTES

The first time cis is invoked by a user, it just sets the date in the user's default value segment. The value segment is created if one doesn't exist and is normally:

```
>udd>Project_id>Person_id>Person_id.value
```

but can be changed by the value_set_path command.

For links which match the star names, the date-time-entry-modified of the target of the link is checked rather than that of the link itself.

The cis active function returns entrynames of selected info segments separated by spaces. If -absp is specified, it returns full pathnames of info segments separated by spaces.

WARNING

Since the cis active function also sets the date in the user's default value segment, a command line using `[cis]` sets this date before processing any of the returned info seg names. As a result, segments can be unintentionally skipped and not seen a second time if a command line containing `[cis]` is interrupted.

Name: copy, cp

SYNTAX AS A COMMAND

```
cp path1 {path2 ...  path1N path2N} {-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
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
is the pathname of a copy to be created from path1. If the last path2 argument is not given, the copy is placed in the working directory with the entryname of path1. 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 path1. Status permission is required for the directory containing path1 if the -name, -acl or -all control argument is specified. Append permission is required for the directory containing path2. Modify permission is required for the directory containing path2 if the -name, -acl, or -all control argument is specified.

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. The AIM access class of a segment is not copied by -acl.

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 path1) already exists in the directory specified by path2. 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 path1 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.

Name: copy_dir, cpd

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. The user can also specify that portions of the subtree be copied and can control the processing of links.

ARGUMENTS

source_dir
is the pathname of a directory to be copied. The star convention is allowed to match directory names. Matching names associated with other storage types are ignored. The source_dir can not be contained in target_dir.

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.

CONTROL ARGUMENTS

-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.

-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".

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

-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.

-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.

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

-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.

LIST OF ENTRY TYPE KEYS

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.

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.

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

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 source_dir, an appropriate error message is printed and control is returned to command level.

If name duplication occurs while appending the source_dir to the target_dir and the name duplication is between directories, the user is queried whether processing should continue. If the user answers yes, the contents of the directory are copied (appended) but none of the attributes of that directory are copied. If the answer is no, the directory and its subtree is skipped. If name duplication should occur between segments, the user is asked whether to delete the existing one in target_dir. (See the copy command)

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

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.

See also the copy, move and move_dir commands.

EXAMPLES

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).

Name: create, cr

SYNTAX AS A COMMAND

```
cr paths {-control_arg}
```

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.

CONTROL ARGUMENTS

-name STR, -nm STR
: specifies an entryname STR that begins with a minus sign, to distinguish it from a control argument.

-ring_brackets N1 {N2 {N3}}, -rb N1 {N2 {N3}}
: specifies the desired ring brackets for the created segment. N3 defaults to N2, which defaults to N1, which defaults to the user's validation level.

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.

See the description of the create_dir and link commands for an explanation of the creation of directories and links, respectively.

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.

Name: create_dir, cd

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 ARGUMENTS

-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.

-name STR, -nm STR
specifies an entryname STR that begins with a minus sign, to distinguish it from a control argument, or consists solely of white space.

-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.

-ring_brackets N1 {N2}, -rb N1 {N2}
specifies the ring brackets of the created directory. N2 defaults to N1, which defaults to 7.

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 entry has only one name, the user is aksed whether to delete the old entry. If the old entry 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.

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.

Name: date

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 the convert_date_to_binary_ subroutine. If no argument is specified, the current date is returned. The DT argument is concatenated to form a single string even if it contains spaces, and need not be quoted.

NOTES

See the Subroutines manual for a complete description of convert_date_to_binary_.

EXAMPLES

The command line:

```
date May 5, 1980
```

prints:

```
05/05/80
```

The command line:

```
date Monday
```

prints the next occurrence of a Monday.

Name: 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 and messages sent by the send_message_express command are not saved.

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 < 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.

Name: delete, dl

SYNTAX AS A COMMAND

```
dl {paths} {-control_args}
```

FUNCTION

causes the specified segments and/or multisegment files to be deleted. Use delete_dir to delete directories. Use unlink to delete links.

ARGUMENTS

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

CONTROL ARGUMENTS

-absolute_pathname
causes the entries listed by -long, -query_all and -query_each to have the entire pathname printed. The default is to print entrynames.

-brief, -bf
inhibits the printing of an error message if a segment or multisegment file to be deleted is not found.

-chase
deletes targets of links specified by paths, as well as segments.

-entryname, -etnm
causes the entries listed by -long, -query_all and -query_each to have only the entrynames printed, rather than the entire pathname. This is the default.

-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 <)

-no_chase
does not delete targets of links. (Default)

-query_all, -qya
lists all segments to be deleted, and issues a query as to whether they should all be deleted or not. Unless -force is given, an individual query will be given for protected segments.

-query_each, -qye
issues a query for every entry to be deleted, whether or not it is protected. Protected segments will be noted in the query.

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.

Name: delete_acl, da

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. For a description of ACLs, see the Programmers' Reference Manual.

ARGUMENTS

path
: pathname of a segment, multisegment file, or directory. If it is -wd or -working_directory, the working directory is assumed. 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. If User_id is omitted, the user's Person_id and current Project_id are assumed.

CONTROL ARGUMENTS

-all, -a
: deletes all ACL entries except for *.SysDaemon.*.

-brief, -bf
: suppresses the messages "User name not on ACL." and "Empty ACL."

-chase
: chases links when using the star convention. Links are always chased when path is not a starname.

-directory, -dr
specifies that only directories are affected. The default is segments, multisegment files, and directories.

-no_chase
does not chase links when using the star convention. (Default)

-segment, -sm
specifies that only segments and multisegment files are affected.

ACCESS REQUIRED

The user must have modify permission on the containing directory.

Name: delete_dir, dd

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. Use the delete command to delete segments; use the unlink command to delete link entries.

ARGUMENTS

paths
are pathnames of directories. The star convention is allowed.

CONTROL ARGUMENTS

-absolute_pathname
causes the entries listed by -long, -query_all and -query_each to have the entire pathname printed. The default is to print entrynames.

-brief, -bf
inhibits the printing of an error message if the directory to be deleted is not found.

-entryname, -etnm
causes the entries listed by -long, -query_all and -query_each to have only the entrynames printed, rather than the entire pathname. This is the default.

-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 <.)

-query_all, -qya
lists all directories to be deleted, and issues one query for all of them.

-query_each, -qye
issues a query for each directory being deleted. This is the default.

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.

If the -force control argument is not specified, delete_dir asks the user whether to delete the specified directory. It is then 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 subtrees of that master directory are deleted, even if they are mounted.

Name: 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.)

ARGUMENTS

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.

Name: delete_name, dn

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 error messages when entries are not found. This is the default.

-name STR, -nm STR
specifies a nonstandard entry name STR (e.g., a name which looks like a 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.

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.

Name: delete_search_paths, dsp

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 the add_search_paths command.

Name: 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 "Notes" 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.

Name: 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.

ARGUMENTS

command_line
is a command line. It need not be quoted.

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. If the control argument is specified, it must appear before command_line.

NOTES

If the command specified in command_line cannot be executed, an error message is printed.

Name: do

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

do

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. To access more than 9 arguments, see "Notes on accessing more than nine arguments" below. 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:

-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.

-brief, -bf
: suppresses printing of the expanded command line. This is the default.

-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.

-interactive
: does not catch any signals. This is the default. (See "Notes on Modes" below.)

-long, -lg
: prints the expanded command line on error_output before it is executed or passed back.

do

-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.

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 the Programmers' Reference Manual.)

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 i'th 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

do

of &ri, and of &qi written between an additional level of quotes, are identical. The string &rfi is replaced by a list of the i'th 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 ".1"
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.*
```

do

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.

Name: emacs

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.

For a basic introduction to the Emacs Text Editor, and descriptions of the most generally used editing requests of emacs fundamental mode, see the *Introduction to Emacs*, Order No. CP31. A tutorial introduction to the Emacs Text Editor, fully describing the editing requests available, and containing instructions for using special features of emacs can be found in the *Emacs Text Editor Users' Guide*, Order No. CH27. A guide for programmers writing extensions and terminal control modules (CTL) in the Lisp programming language is provided in the *Emacs Extension Writers' Guide*, Order No. CJ52.

ARGUMENTS

paths
: are pathname(s) of segments to be read in. Each is put into its own appropriately named buffer. Star and archive-component pathnames are accepted.

CONTROL ARGUMENTS

-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.

-line_length N, -ll N
: sets the line length to be different from the terminal's default line length.

-line_speed N, -ls N
: indicates linespeed to obtain proper padding (for ARPANet users), where N is the output line baud rate in bits/second.

-macros path, -macro path, -mc path
: loads the segment, specified by path, as Lisp, so that features therein are available.

-no_start_up, -no_startup, -ns
: prevents use of the user's startup (start_up.emacs).

-page_length N, -pl N
sets the page length to be different from the terminal's default page length.

-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 cannot be used with -ttp or -reset.

-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 cannot be used with -ttp or -query.

-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 cannot be used with -reset or -query.

None of the terminal-type control arguments (-ttp, -reset, -query, -line_speed) are generally necessary; they are only used for solving various communications problems.

NOTES

A complete list of available requests in emacs can be gotten via the make-wall-chart request while in emacs. Type the following:

```
emacs
ESC-X make-wall-chart
```

where ESC is the escape button on the terminal.

In addition, emacs provides its own on-line, interactive tutorial which can be invoked by typing the following:

```
emacs
^
 _
```

where "^" stands for the CONTROL button which is to be held down while pressing the underscore character.

See also the list_emacs_ctls command which produces a list of all known Emacs terminal types.

Name: 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 ARGUMENTS

-arguments STRs, -argument STRs, -ag STRs
: 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.

-brief, -bf
: suppresses the message "ID: HHMMSS.f; N already requested."

-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.

-defer_indefinitely, -dfi
: does not run the absentee process until the operator starts it.

-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.

-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.

-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.

-output_file path, -of path
specifies the pathname of the output segment. (See "Notes" below.)

-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.

-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 specified by the site administrator. 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 -foreground.

-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.

-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.

-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.

-sender STR
specifies that only requests from sender STR should be entered. In most cases, the sender is an RJE station identifier.

-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 Subroutines manual). If the DT string contains blanks, it must be enclosed in quotes.

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. 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. Users should be aware of differing time zones when deferring absentee jobs. If there is a possibility of overlapping times (i.e., when est changes to edt, etc.), specify the time zone in the value given for -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.

The absentee login and logout messages are generated by the absentee process itself. The messages are written to the user_i/o switch. If a fatal process error occurs, or certain types of process damage occur, the messages may not appear in the absout segment. Also note that if output is being diverted to another file (by the file_output

command, for example) when such an error occurs it may be necessary to issue the adjust_bit_count command for that file. This is because the revert_output command was never executed, and the bit count of the file being written was not updated.

EXAMPLES

Suppose that a user wants to request an offline compilation. This can be done with a control segment called absentee_pl1.absin containing:

```
change_wdir current_dir
pl1 x -table -map
dprint -delete x.list
logout
```

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 the default queue 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 edt" -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. Eastern Daylight Time, 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.

Name: enter_output_request, eor

SYNTAX AS A COMMAND

```
eor {paths} {-control_args}
```

FUNCTION

submits requests to printer, punch or plotter queues. Control arguments are available to modify processing of the requests. All control arguments are nonpositional. Paired control arguments override one another if both are used in a single command. The user can also establish personalized default settings for these control arguments.

ARGUMENTS

paths
are pathnames of segments or multisegment files to be printed, punched or plotted. The star convention is accepted. Null links and directories matching a starname are ignored without error.

BASIC OPERATION: When the eor command is invoked with a pathname argument, it submits a request to print the file(s) identified by path. Each printed listing is identified by a destination string, which tells the operator how to route the listing to the submitter, and by a heading string, which further identifies the submitter or the listing. When no control arguments are given in the command line, eor uses default values for the header, the destination, the request type and priority at which the request is queued, the number of copies to be printed, and so on.

BASIC CONTROL ARGUMENTS
The following requests represent a subset of the total requests. See the Commands and Active Functions manual for the complete list of requests.

-print, -pr
submits requests for printing. (default)

-punch, -pch
submits requests for punching.

-plot
submits requests for plotting on an installation-defined plotting device.

-request_type STR, -rqt STR
submits requests to the STR printer, punch or plotter request type. STR must be one of the request types listed by the print_request_types (prt) command. (default: printer when printing, punch when punching, plotter when plotting)

-queue N, -q N
submits requests to queue N of the request type. (default: varies depending upon the request type)

-header {-control_args} STR, -he {-control_args} STR
identifies output with a heading of STR. The STR is limited to 59 characters, and must be quoted if it contains spaces. If STR contains any equal signs (=) or percent characters (%), then the heading is constructed by applying the equals convention to STR and the entryname of the file being processed. (default: submitter's person_id)

-destination STR, -ds STR
labels output with the STR string, which is used to determine where to deliver the output. The STR is limited to 24 characters, and must be quoted if it contains spaces. If STR contains any equal signs (=) or percent characters (%), then the destination string is constructed by applying the equals convention to

STR and the entryname of the file being processed. (default: submitter's project_id)

-copy N, -cp N
produces N copies of the printed, punched or plotted output. N may be any number from 1 to 30. (default: 1)

-delete, -dl
deletes files after they are printed, punched or plotted.

-no_delete, -ndl
does not delete files after they are printed, punched or plotted. (default)

-notify, -nt
sends a confirming message to the submitter when the request has been processed, showing the pathname and charge.

-no_notify, -nnt
does not send the confirmation. (default)

Name: exec_com, ec (version 2)

SYNTAX AS A COMMAND

```
ec path {ec_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ec path {ec_args}]
```

FUNCTION

Executes programs written in the exec_com language, used to pass command lines to the Multics command processor and pass input lines to commands reading input. The syntax described here is known as Version 2, for which the first line of the exec_com program must be the line consisting of "&version 2". For a description of Version 1 syntax, see the Commands and Active Functions manual.

ARGUMENTS

path
is the pathname of an exec_com program, written using the constructs described below. The ec suffix is assumed if not specified. The star convention is NOT allowed.

exec_com (version 2)

ec_args
are optional arguments to the exec_com program, and are substituted for parameter references such as &1. See "List of parameters".

LIST OF PARAMETERS

&1 - &9
expand to the 1st through 9th ec_args, or to defaults defined by a &default statement or to null string if there is no corresponding ec_arg.

&(1) - &(9)
are synonyms for &1 - &9.

&(11), &(12), etc.
expands to the corresponding ec_arg, or to a default defined by &default or to null string if there is no corresponding ec_arg. The parentheses are required when there are two or more digits.

&q1 - &q9
&q(1), &q(11), etc.
expands to the corresponding argument with quotes doubled according to the quote depth of the surrounding context. See "Notes on quoting". This parameter ensures that quotes in the argument to exec_com are handled correctly under the quote-stripping action of the command processor.

&r1 - &r9
&r(1), &r(11), etc.
expands to the corresponding argument enclosed in an added layer of quotes, and internal quotes doubled accordingly. See "Notes on quoting". This parameter keeps the value of the argument as a single unit after one layer of quote-stripping by the command processor.

&n
expands to the number of ec_args specified to exec_com.

&f1 - &f9
&f(1), &f(11), etc.
expands to a list of the Nth through last ec_args separated by spaces. If N is greater than the value of &n, expands to null string.

&qf1 - &qf9
&qf(1), &qf(11), etc.
expands to a list of the Nth through last ec_args, with quotes doubled, separated by spaces. If N is greater than the value of &n, expands to null string. This parameter is equivalent to: &qN &qN+1 &qN+2 ....

&rf1 - &rf9
&rf(1), &rf(11), etc.
expands to a list of the Nth through last ec_args, individually requoted, separated

exec_com (version 2)

by spaces. If N is greater than the value of &n, expands to null string. This parameter is equivalent to: &rN &rN+1 &rN+2 ....

&f&n, &qf&n, &rf&n
expands to the last ec_arg specified to exec_com, either as is, with quotes doubled, or requoted.

&ec_dir
expands to the pathname of the directory containing the exec_com currently running. It can be used to call other exec_com's in the same directory.

&ec_name
expands to the entryname of the exec_com currently running, with any ec or absin suffix removed (the absin suffix is for an exec_com invoked by the absentee facility). This parameter can be used to simulate entrypoints in an exec_com segment, by adding multiple names to the segment and transferring to a different &label depending on the name invoked: "&goto &ec_name".

&ec_path
expands to the expanded, suffixed pathname of the current exec_com.

&ec_switch
expands to the name of the I/O switch over which the exec_com interpreter is reading the exec_com.

LIST OF VALUE EXPRESSIONS
(All of these constructs can be nested arbitrarily inside each other.)

&(NAME)
expands to the value assigned to the variable NAME by a previous &set statement in the same exec_com. If NAME contains &'s, it is first expanded. Therefore, &() constructs can be nested. However, &'s in the expansion are not re-expanded. A second level of expansion must be specified, therefore, by &(&()). If NAME has not been assigned a value by &set, an error occurs. Variable names are allowed to contain any characters except & and cannot consist solely of digits or white space.

&(N)
where N is a positive integer, expands to the value of the Nth ec_arg to exec_com, or if there is no Nth ec_arg, to the last default value assigned to argument N by a &default statement, or if no default value was assigned, to null string.

&q(NAME), &q(N)
expands to the same thing as &(NAME) or &(N), but with quotes inside the value doubled according to the quote depth of the surrounding context.

&r(NAME), &r(n)
expands to the same thing as &(NAME) or &(N), but requoted and with internal quotes doubled.

exec_com (version 2)

&[ACTIVE STRING]
expands to the return value of an active string by calling the command processor. This construct ends with the matching right bracket.

LIST OF LITERALS
Also see "Notes on white space".

&"..."
encloses an arbitrary character string to be taken literally. Quotes inside the string must be doubled, and the closing undoubled quote ends the literal string.

&&
expands to a single & character, not further expanded.

&AMP, &AMP(N)
expands to a single ampersand character (ASCII 046), in which case it is identical to &&, or to N ampersands where N is a positive integer.

&SP, &SP(N)
expands to a single space character (ASCII 040) or to N spaces.

&BS, &BS(N)
expands to a single backspace character (ASCII 010) or to N backspaces.

&HT, &HT(N)
expands to a single horizontal tab character (ASCII 011) or to N horizontal tabs.

&VT, &VT(N)
expands to a single vertical tab character (ASCII 013) or to N vertical tabs.

&FF, &FF(N), &NP, &NP(N)
expands to a single form-feed character (ASCII 014) or to N form-feeds.

&NL, &NL(N), &LF, &LF(N)
expands to a single newline character (ASCII 012) or to N newlines.

&QT, &QT(N)
expands to a single double-quote character (") or to N of them.

&!
expands to a Multics 15-character unique name, for example "!BBBhjBnWQpGbbc". Multiple occurrences of & within the same exec_com expand to the same string.

LIST OF PREDICATES

&is_defined(NAME)
expands to "true" if the variable named NAME has been assigned a value by an &set statement in the current exec_com, "false" otherwise (See "Notes on variables"). This construct expands to "true" if &(NAME) can be expanded, "false" if &(NAME) is an error.

exec_com (version 2)

&is_defined(N)
where N is a positive integer, expands to "true" if an Nth ec_arg was specified to exec_com or an Nth default was defined via the &default statement (see "List of assignment statements"), "false" otherwise.

&is_absin
expands to "true" if the exec_com is being executed by the absentee facility, "false" if it is being executed by the exec_com command or active function.

&is_active_function, &is_af
expands to "true" if the exec_com is being executed by the exec_com active function, "false" otherwise.

&is_attached
expands to "true" if input is currently attached via an &attach statement, "false" otherwise. See "Notes on input attachment". Input is always attached when running as an absentee.

&is_input_line
expands to "true" if the line in which it appears is being read as an input line by some command, "false" otherwise.

LIST OF CONTROL STATEMENTS

&attach
causes any commands subsequently invoked in command lines to read their input from the exec_com rather than from the terminal. See "Notes on input attachment".

&detach
causes any commands subsequently invoked in command lines to read their input from the terminal. This is the default. See "Notes on input attachment".

&if EXPRESSION
expands EXPRESSION to get a true or false value. EXPRESSION can contain any exec_com-expandable constructs, such as &[...] (See "List of value expressions"). If the expanded value of EXPRESSION is "true", the following &then statement (if any) is executed next. If the value is "false", the following &else statement (if any) is executed next. If the value is neither "true" nor "false", an error occurs. See "Examples of if statements".

&then LINE
&then &do LINES &end
&else LINE
&else &do LINES &end
where LINE is any exec_com line, including another &if statement. LINE is executed or not depending on the value of the preceding &if clause. The &then and &else statements, unlike other exec_com statements, are allowed to appear on the same line with one another and with &if. However, the &then or &else cannot be on a separate line from the LINE or &do that it executes. See

exec_com (version 2)

"Examples of if statements". The contents of a &do-&end block reference the same variables as the containing exec_com. No &goto's are allowed into a &do-&end block from outside it.

&goto LABEL
causes the next statement to be executed to be the statement following the first occurrence of "&label LABEL" in the exec_com.

&label LABEL
specifies a target for "&goto LABEL" and is otherwise ignored. The string LABEL can contain any characters except &.

&quit
terminates execution of the exec_com. If the program was invoked by the exec_com active function, the active function return value is a quoted null string ("").

&return LINE
terminates execution of the exec_com. If the program was invoked by the exec_com active function, the active function value is the (expanded) value of LINE, the rest of the line. If the program was invoked by the exec_com command, the expanded value of LINE is printed on the terminal.

LIST OF ASSIGNMENT STATEMENTS

&set NAME1 VALUE1 ... NAMEn VALUEn
assigns values to the variables NAME1 through NAMEn, which are created if no assignments for them already exist. All NAMEj and VALUEj arguments are fully expanded before any values are set. Therefore, the statement:

```
&set a &(b) b &(a)
```

exchanges the values of the variables a and b. Arguments to &set are delimited by white space. White space and literals inside them must be enclosed in &"...". There is no restriction on the lengths of NAMEj or VALUEj. VALUEj can contain any characters; NAMEj cannot be all digits. If VALUEj is the unquoted keyword &undefined, any existing value for NAMEj is deleted, and the &is_defined(NAMEj) construct will expand to "false".

&default VALUE1 ... VALUEn
assigns default values for the exec_com parameters &(1) through &(n). The default value of &(j) only matters if no jth ec_arg was specified to exec_com. The &(j) parameter reference expands to the value of the jth ec_arg, or if there is none, to the jth default value set by &default, or if there is none, to null string. VALUEj arguments are separated by white space, and each is fully expanded before default values are set. White space and literal 's in them must be enclosed in &"...". If VALUEj is the keyword &undefined or &undef, no jth default value is set. This keyword is used as a place-holder to skip the jth position.

exec_com (version 2)

LIST OF PRINTING STATEMENTS

&print LINE
prints the expanded remainder of the line, followed by a newline character. If &print appears on a line by itself, a single newline character is printed.

&print_nnl LINE
prints the expanded remainder of the line, without appending a newline character.

LIST OF TRACING STATEMENTS

&ready on
&ready off
turns ready messages on or off. Turning them on causes the system ready procedure to print a ready message when it is called. The default is off. This statement does not affect whether the ready procedure is called. The ready procedure is normally called after the execution of a command line (see the description of the ready_on command).

&ready_proc on
&ready_proc off
determines whether or not the system ready procedure is called after each command line is executed. The default is on for the exec_com command, off for the active function.

&trace {TYPES} STATE {&prefix PREFIX} {&osw SWITCHNAME}
sets tracing for one or more kinds of lines specified by TYPES. TYPES can be any combination of the following:

&command	command lines.
&comment	comments, including those sharing other lines.
&control	control lines, for example &print....
&input	lines being read as input to some command.

The default if TYPE is omitted is all four types.

STATE can be one of the following:

off, false	disables tracing entirely.
on, true	enables tracing, in whichever of the following modes was last specified. The default mode is "&expanded" for command and input lines, "&both" for control lines.
&unexpanded	prints lines as they appear in the exec_com segment. Implies "on".
&expanded	prints lines after all expansion has been done. Implies "on".
&all	prints at each stage of expansion. Implies "on".
&both	prints each line as it appears in the exec_com, and again after all expansion. Implies "on".

exec_com (version 2)

Defaults for ec's invoked by the exec_com command/active function are "&expanded" for command and input lines, "&unexpanded" for control lines, and "off" for comments. Defaults in the absentee environment are "&expanded" for command and control lines, "off" for control lines and comments.

PREFIX specifies a string to be printed at the start of each line. Default prefixes are all null string.

SWITCHNAME specifies an I/O switch on which to write the trace. The default for all types of lines in ec's invoked by the exec_com command or active function is user_output. The default in the absentee environment is user_io.

NOTES ON ABSENTEE ENVIRONMENT

An exec_com/absin runs in the absentee environment only when it has been invoked directly by the absentee facility, ie. is running an absentee process. Exec_com's called within an absentee process are said to run in the normal exec_com environment.

Input lines in an absentee process come from the absin segment running the process. These, along with output lines, are directed to an absout file. Since both input and output lines are written to the same switch, the default switch is chosen to be user_io for the absentee environment rather than user_output as for exec_com's. This default applies to all tracing, and ensures that even if user_output is redirected somewhere, the input lines driving the process still appear in the absout.

The &attach and &detach statements have no effect in the absentee environment, since input to the absentee process always comes from the absout file. The &is_attached predicate always returns true. The &ready and &ready_proc statements also have no effect in the absentee environment. Instead, the ready_on and ready_off commands should be used.

NOTES ON VERSION

The current version of exec_com is known as Version 2. In many ways similar to the old Version 1, it adds automatic variables, parameter defaults, literal character escapes, indentation, comments on lines, line continuation, expansion of active strings in control lines, and tracing of comments and control lines.

In addition, there are two incompatible changes between the versions. Whereas V1 leaves unrecognized &strings alone, V2 rejects them as syntax errors. This change makes V2 an extensible language. Secondly, V2 parses lines into control keywords and tokens (separated by whitespace) before expansion, so that expansion can only change the values of tokens but not the syntax of a line.

A Version 2 exec_com has "&version 2" as its first line. If this first line is not present, the exec_com is interpreted as Version 1. Version 1 exec_com's can optionally begin with "&version 1"; at some future time, Version 2 will be the default and "&version 1" will be required.

exec_com (version 2)

A conversion command is available to translate Version 1 exec_com's to Version 2. See the convert_ec (cvec) command.

NOTES ON WHITE SPACE

White space (SPACE, HORIZONTAL TAB, VERTICAL TAB, and FORM-FEED) is ignored at the beginning and end of each line. As a result, exec_com lines can be indented freely. Intentional white space at the beginning or end of a line (for example, an editor input line) must be specified by literal escapes such as &SP. See "List of literals".

NOTES ON COMMENTS

Comments are specified by the character sequence &- anywhere in a line. Where this sequence appears (outside of &"..."), the remainder of the line is a comment and can contain any characters. White space preceding the comment, if any, is ignored. Therefore, comments can be aligned at a particular column without affecting the executable text. White space before a comment can be specified by the literal escapes described in "List of literals".

NOTES ON CONTINUATION

Long command lines and other portions of text that must not be broken can be continued on successive lines by means of the character sequence &+ at the beginning of each continuation line. White space preceding the &+ is ignored. An example is:

```
sm Bartley.TRG This is such a long message I prefer to
     &+ stretch it onto a second line of the exec_com.
```

Note that whitespace following the &+ is part of the executable line, and in the above example is necessary to separate arguments to sm.

Continuation is not affected by intervening comments, whether at the end of executable text lines or on lines by themselves. This feature can be used to comment parts of statements, for example:

```
sa fast_print adros *.Admin.*          &-Maintainers
&-The XPer project should be added later
&+ aos *.*.*                           &-Non-maintainers
```

The complementary character sequences "&+" and "&-" can be thought of as meaning "This is part of the executable text" and "This isn't", respectively.

NOTES ON QUOTING

The exec_com interpreter strips one layer of exec_com quotes (&"...") from the text. It does not perform command processor-type stripping of regular quotes ("...").

To defeat one or more levels of command processor quote-stripping, the values of variable and parameter expansions can be quote-doubled or requoted using the "q" and

exec_com (version 2)

"r" prefixes. Quote-doubling doubles existing quote characters in a string according to the depth of quotes inside which the string is currently nested, so that one level of quote-stripping by the command processor will result in the internal quotes looking the same as they do inside the original string. Requoting goes a step further by first quote-doubling, then surrounding string with an additional layer of quotes, thus causing the entire string to remain a single argument after one level of quote stripping by the command processor. In the examples below, "Level" refers to the number of levels deep in quotes that the parameter reference appears in the exec_com text. Assume that the value of the first ec_arg to exec_com is the string a"b containing a single quote character:

```
           &1          &q1              &r1

Level 0    a"b         a"b              "a""b"
Level 1    "a"b"       "a""b"           """a"""b"""
Level 2    """a"b"""   """a"""a"""      """""""""""a""""""""""b"""""""""""
```

The exact number of quote characters is unimportant; the important thing is that &q protects internal quotes from one level of quote stripping by the command processor, and &r ensures that the value remains a single argument to the comand processor. These prefixes are very useful since, if the value of the first ec_arg (for example) contains a space, the value of &1 substituted into a command line will be parsed into more than one command line argument.

If a value is null, the &q prefix does not affect it, and the &r prefix results in a pair of quotes, doubled according to the quote depth of the context.

The "q" and "r" prefixes can be used in the following constructs:

&q1, &q(1)	&r1, &r(1)
&qf1, &qf(1)	&rf1, &rf(1)
&q&n, &qf&n	&r&n, &rf&n
&q(VAR NAME)	&r(VAR NAME)

NOTES ON INPUT ATTACHMENT

By default, commands invoked by command lines within an exec_com read their input from the terminal. By preceding a command line with an &attach statement, the command can be caused to read input lines from the text of the exec_com instead. Note that "&attach" must precede the line on which the input-reading command is invoked; otherwise, before the &attach statement is encountered, the command will already have asked to read a line from the terminal. An example of &attach usage is:

exec_com (version 2)

```
&attach
qedx
r actions.table
$a
&f3
\f
w
q
&detach
```

This example appends to the segment named actions.table a line consisting of the third through last ec_arg arguments to the exec_com. The &detach statement causes any later input-reading command to get its input from the terminal.

While &attach is in effect, the &is_attached predicate expands to "true"; after &detach, it expands to "false". In general, the answer command should be used to answer questions asked by programs via the command_query_ subroutine. Placing the answers in the text using &attach, as in:

```
&attach
read_tape -debug
50065
yes
no
&detach
```

relies on a specific number of questions being asked, and is therefore prone to fail if, for example, an error occurs while executing the command. Note that there is no inherent property of a line making it an input line rather than a command line; the distinction is a property of whether input lines are being read by a command. Use of the answer command makes this example less error-prone:

```
answer 50065 -then yes -then no read_tape -debug
```

EXAMPLES OF IF STATEMENTS

The line-placement of &then and &else statements is left up to the user. Some examples of their usage are:

```
&if EXPRESSION &then LINE1 &else LINE2

&if EXPRESSION
&then LINE1
&else LINE2
etc.
```

More examples:

exec_com (version 2)

```
&if EXPR1 &then &if EXPR2 &then LINE1 &else LINE2 &else LINE3

&if EXPR1
&then &if EXPR2 &then LINE1
      &else LINE2
&else LINE3

&if EXPR1 &then LINE1
&else &if EXPR2
     &then LINE1
     &else LINE2
&else LINE3

&if EXPR1 &then &do
     LINE1
     &if EXPR2 &then LINE2
     &else &do
          LINE3
          LINE4
     &end
&end
```

LIST OF CONSTRUCTS

This alphabetical list of exec_com constructs names the sections in which they are documented:

&"..."	List of literals
&&	List of literals
&(1), &(11), etc.	List of parameters
&(VAR_NAME)	List of value expressions
&[...]	List of value expressions
&+	Notes on continuation
&-	Notes on comments
&!	List of literals
&1, &2, etc.	List of parameters
&AMP, &BS, &FF, &HT, &&NL, &QT, &SP, &VT	List of literals
&all	List of tracing statements (&trace)
&attach	List of control statements
&both	List of tracing statements (&trace)
&command	List of tracing statements (&trace)
&comment	List of tracing statements (&trace)
&control	List of tracing statements (&trace)
&default	List of assignment statements
&detach	List of control statements
&do	List of control statements (&if)
&ec_dir	List of parameters
&ec_name	List of parameters

exec_com (version 2)

&ec_path	List of parameters
&ec_switch	List of parameters
&else	List of control statements
&end	List of control statements (&if)
&expanded	List of tracing statements (&trace)
&f1, &f(1), etc.	List of parameters
&goto	List of control statements
&if	List of control statements
&input	List of tracing statements (&trace)
&is_absin	List of predicates
&is_active_function, &is_af	List of predicates
&is_attached	List of predicates
&is_defined	List of predicates
&is_input_line	List of predicates
&label	List of control statements
&n	List of parameters
&print	List of printing statements
&print_nnl	List of printing statements
&q1, &q(1), etc.	List of parameters
&quit	List of control statements
&r1, &r(1), etc.	List of parameters
&ready	List of tracing statements
&ready_proc	List of tracing statements
&return	List of control statements
&set	List of assignment statements
&then	List of control statements (&if)
&trace	List of tracing statements
&undefined, &undef	List of assignment statements (&default)
&unexpanded	List of tracing statements (&trace)
&version	Notes on version

Name: 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

-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).

-source_switch STR, -ssw STR
: specifies the name of an I/O switch to be redirected. The default is user_output.

-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.

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.

Name: format_document, fdoc

SYNTAX AS A COMMAND

```
fdoc path {-control_args}
```

FUNCTION

Basically, the format_document command (a text formatter) takes an input file which you have created using a text editor, formats that file, and either displays it on your terminal or writes it to a new file with a unique name. To direct format_document to perform certain actions, you place special lines, called control lines, in your input file. All control lines begin with a period and must be on a line by themselves. The format_document command makes certain assumptions about how the document is to be formatted (i.e., when the format_document command executes, it defaults to certain conditions in the absence of user-specified control lines). It assumes that your output is going to be on standard-sized paper which has 66 lines per page and that you want your printed lines to be 65 characters wide. These values represent an 8 1/2 by 11 inch page with one inch margins all around. It also assumes that you wish to have both the left and right margins lined up evenly like the margins of this paragraph. When you want format_document to do something different than the standard defaults you must insert the necessary control lines in your input file to accomplish what is desired.

It is important at this time to talk about breaks. As discussed below, line filling is when words are moved from line to line to make the line size as near to the prescribed length as possible. A break is an action that temporarily stops this process (i.e., it processes the previous line, the line just ahead of the break) and prints this line as is even if it is a short one. All of the control lines, with the exception of .pdl and .pdw, cause breaks. A blank line or a line that starts with a space also causes a break.

Following is a summary of the control lines recognized by the format_document command.

.alb
: (align both) puts extra spaces into each line so that both the left and the right margins are even. This control line is effective only if fill (.fin) is also in effect. (Default)

.all
: (align left) does not put extra spaces into the lines. The left margin is even and the right margin is ragged. This control line is effective only if fill (.fin) is also in effect.

.fif
: (fill off) retains lines in the output file as they are in the input file no matter how long or short.

.fin
(fill on) restructures the input file lines to the current line length for the output file by taking a word or words from the next line in order to fill the line as close as possible to the current line length. If a line in the input file is longer than the current line length, move a word or words to the next line, etc. (See the description of the .alb and .all control lines.) (Default)

.in {n}
(indent) sets the indention level. You can have format_document indent each line a certain number of characters. If n is given with a plus or minus sign then n is added to or subtracted from the current indention level. If n is given without a sign then n becomes the indention level. An error message is displayed if an indention level is less than zero or greater than the line length. (The default indention level is 0.)

.pdl {n}
(page length) sets the page length. If n is given with a plus or minus sign then n is added to or subtracted from the current page length. If n is given without a plus or minus sign the page length is changed to n. The format_document command inserts blank lines at the top and bottom of each page, so be careful not to set the page length to a value less than 13 (or less than 14 if you are having page numbers printed.) An error message is displayed if a page length of less than the required lines is given. (The default page length is 66 lines.)

.pdw {n}
(page width) sets the page width (line length). If n is given with a plus or minus sign then n is added to or subtracted from the current line length. If n is given without a plus or minus sign the line length is changed to n. An error message is displayed if the set line length does not accommodate the input file. (The default page width is 65 characters.)

.un {n}
(undent) sets the indention level for the output of the next line only. If n has a plus sign or no sign, then indent n characters less than the current indention level. If n has a minus sign, then indent n characters more than the current indention level. If this seems backwards, just remember that undent goes in the opposite direction from indent. An error message is displayed if the indention that is caused by undenting is less than zero or more than the line length.

ARGUMENTS

path
is the pathname of an input segment or multisegment file. The suffix fdocin must be the last component of the entryname; however the suffix need not be supplied in the command line.

CONTROL ARGUMENTS

-indent {N}, -ind {N}
indents the output N spaces from the left margin. This space is in addition to any indention established by the usage of the indent control line within the text of the input file.

-output_file {PATH}, -of {PATH}
directs the output to a file instead of to the user's terminal. If PATH is not given, then the output is written to an output file whose name is formed by replacing the suffix fdocin of the input file entry name with the suffix fdocout. (The default for this feature is off.)

-page_numbers, -pgno
ends each page with two blank lines and a centered page number. (The default for this feature is off.)

EXAMPLES

What follows is an example of a business letter created using the format_document command (fdoc). Suppose you are creating a business letter that is to be printed on a standard 8-1/2 by 11 inch piece of paper and that you want the lines to be 60 characters long. You first create the input file with a text editor. In this example the input file is labeled letter.fdocin. Line numbers are shown on the example for purposes of commentary immediately following the example.

```
1   ted
2   a
3   .pdw 60
4   .fif
5   .in 35
6   9341 Millennium Lane
7   Reston, Virginia 22061
8   November 24, 1982
9   <NL>
10  <NL>
11  <NL>
12  .in
13  Zimmerman Widget Company
14  53698 Dixie Highway
15  Drayton Plains, Michigan 48999
16  <NL>
17  <NL>
18  Dear Sir,
19  <NL>
20  .fin
21  .un -5
22  I recently purchased one of your model GX-721 widgets.
23  I feel that your engineering staff deserves high
24  praise for this new model.  It is apparent
25  that a great deal of thought has gone into its
26  design.  I am particularly pleased with the optional
27  conetop replacement mechanism.
28  <NL>
29  .un -5
30  My purpose in writing this letter, however, is to
31  obtain information.  As you are well
32  aware, the filter requires a complete overhaul after
33  each 250 hours of use.  The service brochure indicates
34  that the nearest service center to my location is in
35  Chapel Hill, North Carolina, which is a six hour drive
36  from my residence.  If you can direct me to a service
37  center that is more convenient to my location, I would
38  be forever in your debt.
39  <NL>
40  <NL>
41  .fif
42  .in 35
43  Sincerely yours,
44  <NL>
45  <NL>
46  <NL>
47  <NL>
48  Michael P. Marley
```

```
| 49 \f                                                                |
| 50 w letter.fdocin                                                   |
| 51 q                                                                 |
|______________________________________________________________________|
```

line 1
Invokes the text editor.

line 2
Places the text editor in append mode.

line 3
Sets the line length (page width) to 60 characters. If this control is not present, then the line length would be set to 65 characters by default.

line 4
Turns fill mode "off". The reason for turning fill off is because the text beginning on line 6 through line 8 is an address. If fill mode was not turned "off" then the address would be reformatted by fdoc, words might be moved from line to line, or extra spaces might be filled in. You do not want this to happen, so you turn fill off. The same thing is done at line 43 just prior to the closing.

line 5
Sets the indention to character position 35. Text begins at column 1 unless you change it, and since the return address is to be on the right-hand side of the letter you must set the indention to the location desired (character position 35 in this case).

line 6-8
Return address.

line 9-11
Three blank lines are inserted by pressing the newline (NL) or carriage return (CR) key three times.

line 12
Resets the indention level to 0 (the absence of a number after the control results in a default to 0).

line 13-19
Address of the recipient, two blank lines, the salutation, and another blank line.

line 20
Turns fill mode "on" (fill was turned off by the control on line 4) as you want the body of the letter filled.

line 21
The indention level is set to 0 by the control in line 12, but you want to indent

the next line (and only the next line) by 5 characters since it begins a paragraph. To change the indention for only one line you use the undent control which works in the opposite direction of the indent control. Undent subtracts the number from the indention (i.e., if you used .un 5 it would move the indention 5 spaces to the left). You want to move 5 spaces to the right to indent the paragraph, so you use a negative number.

line 22-40
This is the body of the letter. Notice that there has been no attempt to control the entered line lengths, it is entered free-form. The fdoc command formats all of the data for you, so long as fill mode is "on". Lines can be as short or as long as you wish, even if the lines wrap around (WRAP AROUND is the situation where the user continues entering data until the line on the terminal has reached the right margin, at which point the system moves the cursor to character position 0 of the next line, and data entry is continued until a newline or a carriage return is entered).

line 41
Turns fill mode "off".

line 42
Sets the indention to character position 35 so that the letter closing, signature, and sender's name appear on the right side of the page (lines 45-50).

line 49
Terminates append mode and returns the user to edit mode.

line 50
Writes the buffer contents to permanent storage. In this case the buffer is stored in a segment identified as letter.fdocin.

line 51
Quits from the editor and returns the user to Multics command level.

Now that your input file (letter.fdocin) is ready, you can have it formatted and printed on the terminal for your perusal.

```
!    fdoc letter.fdocin



                                        9341 Millennium Lane
                                        Reston, Virginia 22061
                                        November 24, 1981



Zimmerman Widget Company
53698 Dixie Highway
Drayton Plains, Michigan 48999


Dear Sir,

     I recently purchased one of your model GX-721 widgets.
I feel that your engineering staff deserves high praise for
this new model. It is apparent that a great deal of thought
has gone into its design. I am particularly pleased with
the optional conetop replacement mechanism.

     My purpose in writing this letter, however, is to
obtain information. As you are well aware, the filter
requires a complete overhaul after each 250 hours of use.
The service brochure indicates that the nearest service
center to my location is in Chapel Hill, North Carolina,
which is a six hour drive from my residence.  If you can
direct me to a service center that is more convenient to my
location, I would be forever in your debt.


                                        Sincerely yours,



                                        Michael P. Marley

      r 1154 0.149 25
```

Assume the output looks good, and you are ready to make a final copy. Since your lines are 60 characters long and you are going to print it on a standard 8-1/2 by 11 inch piece of paper, and since most terminals and printers print 85 characters in 8-1/2 inches, you will want your letter to be centered on the paper. This is where the -indent control available within the format_document command comes into play.

Your lines are 25 characters shorter than the width of the paper, so if each line begins at character position 12 (roughly half of 25) your letter will be centered on the page. The command line:

```
! fdoc letter -indent 12
```

accomplishes this.

Let us say, for example, that you are going to save your letter in a file so that you can print it later on another terminal or on a high-speed printer. In such a situation, you would type:

```
! fdoc letter -indent 12 -output_file
```

The -output_file control argument saves the output in a file rather than printing it on your terminal. In this example, the file is named letter.fdocout. You can now use the dprint or print commands to print the letter.

Let us say that you have a high-quality printing terminal that you wish to use to print this letter on a piece of typing paper. You would type:

```
! print letter.fdocout -stop
```

After entering this command, place the typing paper in the terminal, position it so that printing begins at the top, and then enter a carriage return (newline character). The letter is then printed, stopping at the last line. At this point, you can remove the paper and put in a new sheet (in the case where the letter is more than one page). When the letter has been printed you can enter another carriage return, and you are back at Multics command level.

Name: forum

The forum command enters the Forum interactive meeting system. Once the command is invoked, you are placed in the Forum subsystem, where you must use Forum requests. Forum requests are listed below under "List of Requests".

USAGE

forum {meeting_name}

where meeting_name is the name or pathname of the meeting to be entered immediately upon invoking Forum. If a pathname is specified, it identifies the meeting to enter. Otherwise, Forum searches for meeting_name by using the forum search list.

NOTES ON REQUESTS

The notation "trans_specs" used with the requests below refers to "transaction specifiers". Transaction specifiers let you reference individual transactions or groups of transactions in a meeting. You can refer to transactions by transaction number, regular expression, or keyword, as described below.

Transaction Numbers

Forum assigns numbers to transactions as it enters them into the proceedings of a meeting. When you enter a meeting for the first time, Forum automatically takes you to the first transaction, [0001]. At this point, transaction [0001] is the current transaction. Forum keeps track of the current transaction just like it keeps track of the current meeting. As you refer to other transactions, the last one you dealt with is always the current transaction. If you print transaction [0010], when it has printed and the prompt waits for your next request, the current transaction is still [0010]. So when you type "print +5", Forum prints transaction [0015], that is, current (which Forum knows without being told its number) +5 equals [0015]. You can use a minus sign (-) in the same manner, as well as a colon (:) between numbers to specify a range. To print transactions [0001] through [0005], type "print 1:5".

Regular Expressions--Subject/Text

In its simplest form, a regular expression consists of one or more characters delimited by the right slant character (/). If the regular expression contains spaces, you must enclose it in quotes. For example, all of the following are valid regular expressions:

```
/one/
"/one or/"
/FOR/
/F/
/characters,/
/ters,/
/pl1/
/:/
```

To list all transactions that contain the character string "solution" in the subject line only, type "list -subject /solution/". If you want to list only transactions that contain "solution" within their text, type "list -text /solution/".

Keywords

keywords let you refer to transactions by their order in the proceedings without knowing their actual numbers. Following is a list of keywords:

all, a
: refers to all transactions (same as first:last).

current, c
refers to the transaction last listed, printed, written, or reset to.

first, f
refers to the first transaction in the proceedings.

last, l
refers to the last transaction in the proceedings.

new
refers to transactions that you have not yet seen (i.e., those that have been entered since you last attended the meeting).

next, n
refers to the transaction immediately after the current transaction.

previous, p
refers to the transaction immediately before the current transaction.

unprocessed, u
refers to the unprocessed transaction (i.e., a transaction that you have authored but not yet entered into a meeting).

LIST OF REQUESTS

Listed below are a subset of the available requests that you can use once you are in the Forum subsystem. See the Forum manual for the complete list of requests.

?
lists the available forum requests and active requests.

.
identifies Forum with version number; gives meeting_name if attending; gives count of new, total, last, and current transactions; and gives number of lines in the unprocessed transaction.

apply command_line
ap command_line
places the unprocessed transaction into a temporary segment, concatenates all the STRs with the pathname, and passes the result to the Multics command processor. The temporary segment is then read back in as the unprocessed transaction.

chairman {meeting_name}
cm {meeting_name}
prints the User_id (Person_id.Project_id) of the meeting's chairman.

current_meeting
cmtg
prints the name of the current meeting.

delete trans_specs
dl trans_specs
allows the chairman to delete specified transactions from the proceedings.

enter {-meeting meeting_name}
en {-meeting meeting_name}
send {-meeting meeting_name}
enters the unprocessed transaction into the proceedings of a meeting.

-meeting meeting_name, -mtg meeting_name
enters the transaction into the proceedings of the meeting_name meeting. The default is to enter the transaction into the meeting the user was attending when the transaction was created. This control argument can be the name or pathname of a meeting. This control argument cannot be used if the transaction was built using the "reply" request.

execute STRs
e STRs
executes STRs as a Multics command line after evaluating Forum active requests. As an active request, returns the result of evaluating strings as an Multics active string.

fill {-control_args}
fi {-control_args}
reformats transaction text to fit in a given line length.

-line_length N, -ll N
is the width to be used when reformatting the text. (Default--either the value specified by the -line_length argument to the forum command or 72 if this argument was not given). The line length given must be between 10 and 136.

-off
does not fill this transaction by default when printed or written.

-on
fills this transaction by default when printed or written.

forum_dir
fd
prints the pathname of the central forum directory.

goto meeting_name
g meeting_name
enters the user into the meeting_name meeting.

help {STR}
prints information about request names or topics. A list of available topics is produced by the list_help request.

`if EXPR -then request_line {-else request_line}`
conditionally execute a request. EXPR is the active string that must evaluate to either "true" or "false". If EXPR evaluates to "true" the request_line following -then is executed. If EXPR evaluates to "false", the request_line following -else is executed.

`list {trans_specs}`
`ls {trans_specs}`
prints a summary of the specified transactions.

`list_help {topics}`
`lh {topics}`
prints a list of available info segments whose names include a topic string.

`list_meetings {meeting_names}`
`lsm {meeting_names}`
prints a list of selected meetings and information about them.

list_requests
lr
prints information about forum requests.

list_users
lsu
prints information about specified participants in a meeting.

`print {trans_specs}`
`pr {trans_specs}`
prints selected transactions from a meeting.

qedx
qx
invokes the qedx editor on the unprocessed transaction.

quit
q
exits Forum.

`reply {trans_spec}`
`rp {trans_spec}`
enters/builds a new transaction in a meeting that has as its subject a reference to some other transaction in the form "Re: <some other subject>", and which will be logically linked to the transaction specified by trans_spec.

`reset {trans_spec} {-meeting meeting_name}`
`rs {trans_spec} {-meeting meeting_name}`
resets the user's current or highest-seen transaction index to the specified transaction.

-meeting meeting_name, -mtg meeting_name
enters the transaction into the proceedings of the meeting_name meeting. The default is to enter the transaction into the meeting the user was attending when the transaction was created. This control argument can be the name or pathname of a meeting. This control argument cannot be used if the transaction was built using the "reply" request.

subject {strings}
sj {strings}
prints or modifies the subject of an unprocessed transaction. If strings are supplied, they are catenated together to become the new subject. If no strings are supplied, the current subject is printed.

subsystem_name
prints the name of the subsystem ("forum").

subsystem_version
prints the current version of Forum.

talk
enters/builds a new transaction in a meeting.

ted
invokes the ted editor on the unprocessed transaction.

write {trans_specs}
w {trans_specs}
writes selected transactions to a segment.

LIST OF ACTIVE REQUESTS

chairman {meeting_name}
cm {meeting_name}
returns the Person_id.Project_id of meeting chairman.

current_meeting {-control_args}
cmtg {-control_args}
returns the name of the current meeting.

forum_dir
fd
returns absolute pathname of central forum directory.

do {request_string} {args}
returns expanded request string.

execute STRs
e STRs
invokes Multics active function within forum request line.

list_meetings
lsm
returns names of meetings that have new transactions.

```
list_users {-control_args}
lsu {-control_args}
```
return names of participants matching given conditions.

subsystem_name
returns the name of the subsystem ("forum").

subsystem_version
returns the current version of Forum.

Name: get_quota, gq

SYNTAX AS A COMMAND

```
gq {paths} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[gq {path} {-control_arg}]
```

FUNCTION

returns information about the secondary storage quota and pages used by segments.

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
includes the cumulative time-page product for the current accounting period. This control argument is not accepted by the active function.

-quota
returns the terminal quota on each directory. The default is to return both terminal quota and number of pages used.

-records_left, -rec_left
returns the number of available pages in each directory, equal to the terminal quota minus the pages used. If a directory has no terminal quota set, the available pages are computed from the terminal quota on the lowest parent with nonzero terminal quota, minus the pages used under that parent with nonzero terminal quota. The default is to return terminal quota and pages used.

-records_used, -rec_used
returns the number of pages used in each directory. The default is to return both terminal quota and number of pages used.

ACCESS REQUIRED

The user requires status permission on each directory for which quota is desired. Determining the value of -records_left may require access further up the hierarchy. If the required access is lacking, an error message is printed.

NOTES

The short form of output (the default case) prints the number of pages of quota used by the segments in that directory and in 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.

NOTES ON ACTIVE FUNCTION

At most one directory can be specified for active function use; the star convention is NOT allowed.

Either -quota, -records_left or -records_used can be specified; the default is -quota.

Name: help

SYNTAX AS A COMMAND

```
help {info_names} {-control_args}
```

FUNCTION

prints descriptions of system commands, active functions, and subroutines; as well as miscellaneous information about system status, system changes, and general information. Help selects this information from segments maintained on-line, which are in a special format, called info segments (info segs). Additional information on use of more complex features of the help command is available in the Multics Commands and Active Functions manual.

ARGUMENTS

info_names
: specify the information to be printed. The suffix ".info" is assumed. If a pathname is specified, it identifies the info seg 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). The star convention is allowed, except when an entry point name is specified or when the -entry_point control argument is used.

  If no info_names are specified, help prints the default info seg help_infos.gi.info which gives a brief introduction to the help facility.

  If the help command fails to find an info seg corresponding to a given info_name, use the list_help command to find info segs which contain the specified info_name in their entrynames.

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 entry point.

information selection
: -all, -a
  prints entire info without questions.
: -brief, -bf
  prints summary of command, active function, or subroutine info.
: -brief_header, -bfhe

prints brief heading with info.

-control_arg STRs, -ca STRs
prints only description of an argument.

-header, -he
prints only a heading line.

-title
prints section titles.

CONTROL ARGUMENTS FOR SELECTING INFO SEGS

-pathname path, -pn path
specifies the pathname of a segment containing the info seg 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>subr_$entry_pt). 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.

-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.

CONTROL ARGUMENTS FOR 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 heading line, followed by the first paragraph of info. At the end of each paragraph, help asks the user if "More help" is needed.

-all, -a
prints the entire info or subroutine entry point description without intervening questions.

-brief, -bf
prints a brief summary of a command, active function or subroutine info seg with no intervening questions. The summary includes the Syntax section, and (for commands and active functions) a list of control arguments and/or other keywords used by the command.

-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 consists of the heading and line count.

-control_arg STR, -ca STR
prints only the descriptions of the control (or other) arguments whose names contain STR. STR 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 STR. The help command prints no other information besides the argument descriptions and asks no questions of the user.

-header, -he
prints only a long heading line consisting of the pathname of the info seg, heading, and line count. No other information is printed. This control argument conflicts with all other information selection control arguments.

-title
lists the section titles used in the info seg (including section line counts), then asks if the user wishes to see the first section.

NOTES

The -all, -brief, -control_arg and -title control arguments are mutually exclusive.

LIST OF RESPONSES

The responses accepted when help questions the user are given in the list below. Those responses that search the info seg or list section titles operate from the current paragraph to the end of the info seg. No wraparound feature is employed.

brief, bf
prints a summary of a command, active function or subroutine info seg, including Syntax section and a list of control arguments, then repeats the previous question.

control_arg STR, ca STR
prints descriptions of control (or other) arguments whose names contain STR, 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_$entry_point_name. If EP_NAME is omitted, help skips to the description of the subroutine_$subroutine_ entry point, if one exists.

header, he
prints a long heading line to identify the current info seg. The line consists of the pathname of the info seg, heading, and line count.

no, n
exits from the current info seg, and begins printing the next info seg selected by info_names given in the help command. Returns from the help command if all selected info segs have been printed.

quit, q
causes the help command to return without printing the remaining info segs selected by the info_names.

`rest {-scn}, r {-scn}`
prints the rest of the info seg without intervening questions. If the -section control argument is 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.

`search {STRs} {-top}, srh {STRs} {-top}`
skips to the next paragraph containing 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 seg. If STRs is 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.

`section {STRs} {-top}, scn {STRs} {-top}`
skips to the next section whose title contains 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 is 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.

`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 seg 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.

`title {-top}`
lists titles and line counts of all sections remaining in the current info seg. If -top or -t is specified, help lists all section titles.

top, t
skips to the beginning of the info seg, prints the heading line, and asks whether the user wants to see the first section. This is useful if the user wishes to review earlier parts of the info seg.

yes, y
prints the next paragraph of information, then asks whether the user wants more help.

?
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.

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 question/answer dialogue continues until all of the information is printed, or until the user replies "no".

INFO NAMING CONVENTIONS

Info segs 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 info segs whose names end with a suffix of ".gi.info". 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 800 info segs are available on-line. To find information about a particular area of the system, use list_help, or the -header control argument with an entryname containing stars to list the names of available infos.

Name: 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

-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.

-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.

LIST OF OPTIONAL ARGUMENTS
: 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.

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 MR10.1, 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 10.1: 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 02/02/83  0908.1
  Last shutdown was at 01/31/83  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*
```

Name: immediate_messages, im

SYNTAX AS A COMMAND

```
im {address}
```

FUNCTION

restores the immediate printing of both messages that are sent to the user by the send_message command and the "You have mail" notification that is sent by the send_mail command.

ARGUMENTS

address
is the address of a mailbox. If no address is specified, the user's default mailbox is assumed. The mailbox must be specified in one of the following forms:

STR
is any argument that does not begin with a minus sign (-). If it contains either of the characters > or < it is interpreted as a mailbox pathname (the .mbx suffix is added if not present); otherwise it is interpreted as a User_id.

-pathname PATH, -pn PATH
specifies the pathname of the mailbox. The .mbx suffix is assumed if it is not present.

NOTES

This command "cancels" the defer_messages command, but does not cancel any options that may have been specified by the accept_messages command. See also the accept_messages and print_messages commands.

Name: 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
is the address of a mailbox to which the user has applied the accept_messages command. If no address is specified, the user's default mailbox is assumed. The mailbox must be specified in one of the following forms:

STR
is any argument that does not begin with a minus sign (-). If it contains either of the characters > or < it is interpreted as a mailbox pathname (the .mbx suffix is added if not present); otherwise it is interpreted as a User_id.

-pathname path, -pn path
specifies the pathname of the mailbox. The .mbx suffix is assumed if it is not present.

NOTES

Also see the description of send_message, accept_messages, last_message_sender, and last_message_time.

Name: 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
is the address of a mailbox to which the user has applied the accept_messages command. If no address is specified, the user's default mailbox is assumed. The mailbox must be specified in one of the following forms:

STR
is any argument that does not begin with a minus sign (-). If it contains either of the characters > or < it is interpreted as a mailbox pathname (the .mbx suffix is added if not present); otherwise it is interpreted as a User_id.

-pathname path, -pn path
specifies the pathname of the mailbox. The .mbx suffix is assumed if it is not present.

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_messages, last_message, and last_message_time in this appendix.

EXAMPLES

Assume that a user has just received the following message:

```
From RJones.Demo 11/19/82 1231.7 mst Fri:  Need access to xy
```

A reply can be sent as follows:

```
sm [lms] Sorry for the oversight, you have access now.
```

Name: link, lk

SYNTAX AS A COMMAND

```
lk path1A {path2A ...  path1N path2N} {-control_args}
```

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. For a discussion of links, see the Programmers' Reference Manual.

ARGUMENTS

path1A
: specifies the pathname of the storage system entry to which path2i is to point. The star convention is allowed. The pathnames must be specified in pairs.

path2A
: specifies the pathname of the link to be created. If omitted (in the final argument position of a command line only), a link to path1A is created in the working directory with the entryname portion of path1i as its entryname. The equal convention is allowed.

CONTROL ARGUMENTS

-chase
: creates a link to the ultimate target of path1A if path1A is a link. The default ia to create a link to path1A itself.

-no_chase
: creates a link directly to the target specified. (Default)

-check, -ck
: refuses to create a link if the target does not exist, or if its existence cannot be determined due to access.

-no_check, -nck
: creates a link whether or not the target exists. (Default)

-copy_names, -cpnm
: after creating the link, copies the names of the target to it.

-no_copy_names, -ncpnm
: does not copy names from the target. (Default)

-name STR, -nm STR
: specifies an entryname STR (either as a path1 or a path2, depending on position) that begins with a minus sign, to distinguish it from a control argument.

ACCESS REQUIRED

The user must have append permission for the directory in which the link is to be created.

NOTES

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 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.

EXAMPLES

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`.

Name: list, ls

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.

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.

ARGUMENTS

entrynames
: are the names of entries to be listed. The star convention is allowed. If entrynames are specified, only entries having at least one name matching an entryname argument are listed. 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, the entrynames and control_args can appear anywhere on the command line.

CONTROL ARGUMENTS

For convenience, control arguments have been arranged in categories according to the function they perform and are listed below:

directory
-pathname path, -pn path

entry type
-all, -a
-branch, -br
-directory, -dr
-file, -f
-link, -lk
-multisegment_file, -msf
-segment, -sm

columns
-count, -ct
-date_time_contents_modified, -dtcm
-date_time_entry_modified, -dtem
-date_time_used, -dtu
-length, -ln
-link_path, -lp
-mode, -md
-name, -nm
-record, -rec

totals/header line
-no_header, -nhe
-total, -tt

multiple-name entries
-match
-primary, -pri

entry order
-reverse, -rv
-sort XX, -sr XX

entry exclusion
-exclude entryname, -ex entryname
-first N, -ft N
-from D, -fm D
-to D

output format
-brief, -bf
-short, -sh

CONTROL ARGUMENTS FOR 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.

CONTROL ARGUMENTS FOR ENTRY TYPE

-all, -a
lists information about all entry types in the following order: segments, multisegment files, directories, and links.

-branch, -br
lists information about branches (i.e., segments, multisegment files, and directories, in that order).

-directory, -dr
lists information about directories.

-file, -f
lists information about files (i.e., segments and multisegment files, in that order). This is the default.

-link, -lk
lists information about links.

-multisegment_file, -msf
lists information about multisegment files.

-segment, -sm
lists information about segments.

CONTROL ARGUMENTS FOR COLUMNS

-count, -ct
prints the count column, which gives the total number of names for entries that have more than one name.

-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.

-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_used, -dtu
prints the date and time the entry was last used.

-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.

-link_path, -lp
prints the link-path column.

-mode, -md
prints the access-mode column.

-name, -nm
prints the names column, giving the primary name and any additional names of each entry. The names column is printed in every invocation of the list command except when the user explicitly requests only totals information.

-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.

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. 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.

CONTROL ARGUMENTS FOR TOTALS AND HEADER LINE

-no_header, -nhe
omits all heading lines.

-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.

If no control arguments from this category are specified, both totals and detailed information are printed.

CONTROL ARGUMENTS FOR MULTIPLE-NAME ENTRIES

-match
prints, in the names column, only those names that match one of the given entrynames.

-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.

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.

CONTROL ARGUMENTS FOR ENTRY ORDER

-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.

-sort XX, -sr XX
sorts entries, within each entry type, according to the sort column XX where XX can be one of the following:

count, ct
sort entries by number of names, most names first.

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_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_used, dtu
sort entries by the date and time used, most recent first.

length, ln
sort entries by length computed from the bit count, largest first. This argument is inconsistent with the -record control argument.

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.)

name, nm
sort entries by primary name, according to the standard ASCII collating sequence.

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.

If no control arguments from this category are specified, entries are printed in the order in which they are found in the directory.

NOTES ON SORTING

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.)

CONTROL ARGUMENTS FOR 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.

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. 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.

CONTROL ARGUMENTS FOR OUTPUT FORMAT

-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 -bf
```

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.

If no control argument from this category are specified, the output format of the list command is not changed.

NOTES

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 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.

NOTES ON THE DEFAULT

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. 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.

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.

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).

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 "7/1/82 000.0" -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 1982. Notice that the -dtu control argument is used to specify that the -to date refers to the date and time used.

Name: 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 comment of each 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.

-pathname, -pn
prints the full pathname of each selected request, rather than just the entryname, just as -absolute_pathname does.

-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 this 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 -all.

-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, can each be starnames.

The User_id arguments specified after the -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
```

Name: list_acl, la

SYNTAX AS A COMMAND

```
la {path} {User_ids} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[la {path} {User_ids} {-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 -working_directory (-wd), the working directory is used. 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

-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.

-chase
: chases links when using the star convention. Links are always chased when path is not a starname.

-directory, -dr
lists the ACLs of directories only. The default is segments, multisegment files, and directories. (See "Notes" below.)

-no_chase
does not chase links when using the star convention. (Default)

-ring_brackets, -rb
lists the ring brackets. This control argument is not valid is list_acl if invoked as an active function.

-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 list_acl command provides effective access information for segments and directories only when discretionary access control is being used (i.e., use of ACLs). If either non-discretionary access control (access control mechanism (AIM)) or intra-process access control (ring brackets) is in use, the status command should be used to determine actual access.

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 and ring brackets, see the Programmers' Reference Manual. For a description of the matching strategy, refer to the set_acl command.

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.

Name: list_output_requests, lor

SYNTAX AS A COMMAND

```
lor {request_identifier} {-control_args}
```

FUNCTION

lists requests in the I/O daemon queue.

ARGUMENTS

request_identifier
can be chosen from the following. If no request_identifier is specified, all requests are listed.

path
is the relative pathname of one or more requests to be listed. The star convention is allowed.

-entry STR, -et STR
selects only requests whose entry names match STR. The star convention is allowed. Directory portions of request pathnames are not used for selecting requests.

-id ID
selects only requests whose request_ids match ID.

CONTROL ARGUMENTS

-absolute_pathname, -absp
prints the full pathname.

-admin {User_id}, -am {User_id}
selects requests of all users or of the specified user. Default is to list the user's own requests. Requires r extended access to the queue(s) to read other users' requests.

-all, -a
searches all queues.

-brief, -bf
prevents printing of the request state in normal (not -long) mode.

-immediate, -im
selects only I/O requests that are not deferred. With -position, ignores deferred requests when computing position.

-long, -lg
prints all information about each selected request, including long request_id and full pathname. Default is to print short request_id and entryname.

-long_id, -lgid
prints the long request_id.

-position, -psn
prints queue positions of each selected request. With -total, prints a list of queue positions. Requires r extended access to the queue(s), to read other users' requests.

-queue N, -q N
searches only queue N. If this control argument is not specified, all queues are searched but nothing is printed for queues from which no requests are selected.

-print, -pr
specifies that the requests listed are found in the queue(s) associated with the default printer request type. See Notes below.

-punch, -pch
specifies that the requests listed are found in the queue(s) associated with the default punch request type. See Notes below.

-plot
specifies that the requests listed are found in the queue(s) associated with the default plotter request type. See Notes below.

-request_type STR, -rqt STR
searches the I/O daemon queues belonging to the specified request type. See Notes below.

-total, -tt
prints only the total number of selected requests and the total number in the queue. Incompatible with -long and -brief control arguments.

-user User_id
selects only requests of the specified user. Requires r extended access to the queue(s).

NOTES

Only request types belonging to the printer, punch, or plotter generic types can be specified by the -request_type control argument when the -long argument is given. A list of these request types can be obtained by invoking the print_request_types command.

The -print, -punch, -plot and -request_type control arguments are mutually exclusive. Only one may be used in a given command. If none are given, then lor lists the default request type used by eor -print (as displayed by the print_request_types command).

Name: login, l

SYNTAX

```
l Person_id {Project_id} {-control_args}
or:
    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 process for the user, or connect the terminal to an existing disconnected process belonging to the user. The login command line can be no more than 300 characters in length.

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 the Commands and Active Functions manual.

-arguments
-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

Name: logout

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.

Name: memo

SYNTAX AS A COMMAND

```
memo {-memo_options} memo_text
or:
    memo {-action_arg} {-memo_options} {-selection_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[memo memo_text]
or:
    [memo -list {-totals}]
```

FUNCTION

maintains a user-created reminder list in a memo segment, which is normally Person_ID.memo in the user's home directory.

ARGUMENTS

memo_text
: is the text of the memo being set. It may not be longer than 132 characters. It can be specified in one of the following forms:

  STR
  : the first string that does not begin with a hyphen is taken as the beginning of the memo text. It and all succeeding strings form the memo text. No further arguments are accepted.

  -memo STRs
  : treats all succeeding STRs as part of the memo text, whether or not they begin with hyphens.

LIST OF MEMO_OPTIONS

These control arguments are used to control various options of the memo being set, or to select memos being otherwise processed.

-alarm, -al
: specifies that the memo is to be an alarm. It will be printed, or executed if set with -call, when its timer goes off, if timers are enabled, rather than when memos are explicitly processed. An alarm memo is deleted immediately after it reaches maturity, unless it was set with -retain.

-call
: passes the memo text to the command processor as a command line when the memo matures, rather than printing it.

-date DT, -dt DT
identifies a date (DT) for the memo to mature in a form suitable for input to the convert_date_to_binary_ subroutine. The DT is truncated to midnight preceding the date in which DT falls.

-expires DT, -exp DT
identifies a time (DT) at which the memo is to expire; this is treated as a delta from the maturity time (which it must be greater than) so that repeating memos with expiration times will work properly. When used as a selection_arg, all expiring memos are selected, regardless of the expiration dates. See "Notes on expiring memos" below.

-invisible, -iv
specifies that the memo is never to be mature and will never be printed during a normal memo print.

-no_retain, -nret
specifies that the memo will only be processed once, and then will be automatically deleted. This is the default for alarm memos.

-repeat DT, -rpt DT
identifies the interval at which the memo is to repeat where DT must be greater than or equal to 1 minute. The repeat interval is applied repeatedly until the new maturity time is greater than the current time, and then the new memo is set. When used as a selection_arg, all repeating memos are selected, regardless of the repeat intervals given. See "Notes on repeating memos".

-repeat_when_processed, -rwp
specifies that the repeat time of a repeating memo will be applied from the time the memo is processed, rather than the maturity time. This is useful for memos which are only significant within a single process.

-retain, -ret
causes an alarm memo to be kept as an ordinary printing (or executing, if set with -call) memo after it matures, rather than being deleted automatically. This is the default for non-alarm memos.

-time DT, -tm DT
identifies a time (DT) for the memo to mature in a form suitable for input to the convert_date_to_binary_ subroutine.

LIST OF ACTION_ARGS

These control arguments control various options of the memo command. Only one may be specified, and they may not be combined with memo setting.

-brief, -bf
suppresses printing of the message "No memos." if no memos are found.

-delete {-force}, -dl {-fc}
deletes all memos selected by the optional arguments. At least one memo must be explicitly specified. Memo will query the user before deleting non-mature memos. when given with -force, causes memos to be deleted even if they are not yet mature, without querying the user.

-list, -ls
prints text and control information of selected memos; no memos are executed. If no memos are explicitly selected, all memos are listed. If -totals is also specified, only the total number of selected memos is printed.

-off
suppresses all memo alarms, until the next memo command with no explicitly specified action. The -on and -off control arguments may be combined with other actions.

-on
enables memo alarms without printing or executing nonalarm memos.

-pathname path, -pn path, -pathname -default, -pn -dft
changes the default memo segment to path if specified with no other action. Otherwise, the memo segment specified by path is used for the execution only of the current memo command. If -pathname is used along with -on or -off, the default memo segment IS changed, and alarms are turned on or off, as appropriate, for the new segment. The suffix ".memo" need not be supplied. When given as -pathname -default, the default memo segment is reset to Person.ID.memo in the user's home directory.

-postpone DT, -pp DT
reschedules the maturity of the selected memos to the time specified by DT, if DT is later than the current maturity time. At least one memo must be explicitly specified.

-print, -pr
prints text of all selected memos. No memos are executed. If no memos are explicitly selected, only mature memos are printed.

-process
causes all mature memos to be processed, and alarms to be turned on, if not otherwise specified. This is equivalent to not explicitly specifying an action.

-status, -st
prints information about the current default memo segment. If -status is specified, it must be the only argument.

-totals, -tt
can only be specified in combination with the -list control argument. When it is used, the total number of memos selected is printed, rather than listing each of the memos.

NOTES

The -delete, -list, -print, -postpone and -process actions are mutually exclusive.

LIST OF SELECTION_ARGS

These arguments are used to select memos to be listed, printed, deleted, or postponed. Some memo_options can also be used to specify types of memos to be selected (see "Notes" below). When more than one selection_args are specified, only those memos that match all of the selection criteria are selected.

memo_number
is either a positive decimal number specifying a single memo (for example 32), or two such numbers separated by a colon, specifying a range of memos (for example 12:16).

-from DT, -fm DT
selects all memos which mature on or after DT. -from can be combined with -to, each of which can only be specified once. This control argument is incompatible with -date and -time.

-match STRING
specifies a string against which memo texts are matched to select memos. STR can not be longer than 32 characters. Up to 40 strings may be specified; all memos that match at least one are selected.

-to DT
selects all memos which mature on or before DT. The -to argument can be combined with -from. This control argument is incompatible with -date and -time.

NOTES

No more than 5082 memos can be contained in a single memo segment. An individual memo can be no more than 132 characters long.

If no action is explicitly specified, and no memo is being set, all mature memos are processed (printed or executed), and the alarm timer is turned on, enabling the processing of alarm memos.

The memo_options can also be used to specify types of memos to be selected; those that take a Date/Time interval (-repeat, -expires, but not -date or -time) will cause the selection of ALL repeating or expiring memos, as the time interval (which must be specified) is ignored.

NOTES ON DEFAULT MEMO SEGMENT

The memo command operates on the default memo segment (unless -pathname is specified with one of the actions -delete, -list, -postpone, -print or -process). This default memo segment is also used when processing alarm timers, to find the memos

which should be processed for the alarm. If the default memo segment has never been explicitly specified (by using -pathname without any other actions), it is the segment Person_ID.memo in the user's home directory.

The default memo segment is created if it does not already exist. If the default memo segment is changed, alarms are turned off for the old memo segment, and then turned on for the new one (if requested). Thus, only one memo segment can have alarms active at a time.

NOTES ON REPEATING MEMOS

A repeating memo repeats by setting a new memo that is identical to the original one, and then turning off the repeat specification in the original memo. Thus the actual repeating memo, rather than its visible consequences, gets a new number each time it repeats. Since the repeat specification is turned off in the original memo, it never repeats again, but remains until deleted, unless it has an expiration date or was set with -no_retain.

An alarm memo that repeats will mature once, and then be automatically deleted, unless it was set with -retain, in which case it is turned into an ordinary, non-alarm memo and lasts until it expires or is deleted.

NOTES ON EXPIRING MEMOS

Expired memos are deleted without being reprinted or executed. However, if they are repeating memos, they are repeated before being deleted. This is useful for cases such as a reminder of a weekly meeting, where the reminder of this week's meeting should always be set, but the reminder of this week's meeting should not be printed if the current time is after the end of this week's meeting. A sequence of repeating memos must be terminated manually (by deleting the current memo); the -expires control argument is not useful for this purpose.

NOTES ON ACTIVE FUNCTION

The memo active function can only be used to set and list memos. When a memo is set, the number assigned to the newly set memo is returned. When memos are listed, a string consisting of the memo numbers selected, separated by spaces, is returned; if -totals is specified, the total count is returned.

Name: move, mv

SYNTAX AS A COMMAND

```
move path1 {path2...path1n path2n} {-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
is the pathname of a segment or multisegment file to be moved. The star convention is allowed.

path2
is the pathname to which path1 is to be moved. The equal convention is allowed. If the last path2 segment is not specified, path1 is moved to the working directory and given the entryname path1.

CONTROL ARGUMENTS

-acl
copies the ACL. (Default)

-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. (Default)

-name, -nm
copies multiple names. (Default)

-no_acl
does not copy the ACL. The segment is given the IACL of the target directory.

-no_chase
does not copy the targets of links that match path1. (See "Notes".)

-no_name, -nnm
does not copy the multiple names.

ACCESS REQUIRED

Read access is required for path1. Status and modify permission are required for the directory containing path1. Status, modify, and append permission are required for the directory containing path2.

NOTES

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.

If an entry with the entryname path1 already exists in the target directory, the user is asked whether the old (already existing) entry should be deleted. If the user answers "no", the move does not take place.

If path1 is protected by the safety switch, the user is asked whether path1 is to be deleted after it has been moved.

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.

Name: move_output_request, mor

SYNTAX AS A COMMAND

```
mor 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.

ARGUMENTS

request_identifiers
    can be chosen from the following:

path
identifies a request to be moved by the full or relative pathname of the input data segment. The star convention is allowed.

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

-id ID
identifies one or more requests to be moved by request identifier. This identifier may be used to further define any path or -entry identifier (see "Notes").

CONTROL ARGUMENTS

-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, if the source and target request types are identical.

-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.

-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.

-print, -pr
specifies that the request moved is found in the queue(s) associated with the default printer request type. See Notes below.

-punch, -pch
specifies that the request moved is found in the queue(s) associated with the default punch request type. See Notes below.

-plot
specifies that the request moved is found in the queue(s) associated with the default plotter request type. See Notes below.

-request_type STR, -rqt STR
specifies that the request moved is found in the queue(s) for the request type identified by STR. Request types can be listed by the print_request_types command.

-to_queue N, -tq N
specifies which queue to move the request to. If not given, the default queue of the target request type is used.

-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.

-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

If any path or -entry STR request identifiers are given, only one -id ID request identifier will be accepted and it must match any requests selected by path or entryname. Multiple -id ID identifiers can be specified in a single command invocation only if no path or entry request identifiers are given.

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.

If the request is already running, it is not moved and a message is printed to the user.

See the Programmers' Reference Manual for a description of request identifiers.

The control arguments -print, -punch, -plot and -request_type are mutually exclusive. Only one may be used in a given mor command. If none are given, then the default request type for eor -print (as displayed by print_request_types) is assumed.

Name: new_proc

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. 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 following command line to be automatically issued in the new process:

```
! exec_com start_up new_proc interactive
```

Name: print, pr

SYNTAX AS A COMMAND

```
print paths {-control_args}
```

FUNCTION

prints ASCII segments and multi-segment files on user_output.

ARGUMENTS

paths
are the pathnames of the segments and multisegment files to be printed. The star and archive component pathname conventions are accepted.

CONTROL ARGUMENTS

-archive, -ac
treat each archive component as a new file for heading and line numbering. If any lines are printed from an archive component, and if -header is specified, print a header identifying the archive component name and the date of modification of the archive component, in the format

```
ARCHIVE::COMPONENT date time
```

where date and time are those stored in the archive. This control argument is the default if archive components were named with the :: convention, or if the entryname of the segment ends in ".archive", unless -no_archive is specified.

-chase
if a starname is specified, include links in the search. Do not complain about missing link targets for starnames.

-exclude STRING, -ex STRING
don't print lines containing STRING. Exclusion is done after matching. Thus, "-match A -exclude B" prints all lines with an A except those with a B.

-exclude /REGEXP/, -ex /REGEXP/
don't print lines containing a string matching the regular expression REGEXP. See the writeup of the qedx command for the definition of regular expressions.

-for N
print N lines from the file including the first line. The default is to print the whole file. If -to is also specified, printing stops when the first control argument is satisfied.

-from X, -fm X
begin printing from the X'th line. The default is line 1.

-from /REGEXP/, -fm /REGEXP/
begin with first line matching the regular expression REGEXP. See the writeup of the qedx command for the definition of regular expressions.

-from_page PP
start printing with the PP'th page, counting the first page as 1. The default is to print starting with the first page.

-header, -he
print a header of the form:

```
NAME date time
```

before each segment. If -archive is specified, the header is printed before each archive component instead of before each segment. This control argument is the default if no other control arguments are given, or if multiple pathnames or the star convention are used.

-indent N, -ind N
print N blanks before each line. This indents the printed output N columns. The default is no indentation.

-left_col N, -lc N
don't print columns 1 to N-1. This argument truncates on the left, printing each line of the file starting with column N. If a line has fewer than N columns, a blank line is printed. The default is to print starting with column 1.

-line_length N, -ll N
format the page with a maximum physical line length of N characters. Space generated by -indent and -number is not counted. If more than N characters are in an output line, the line is split and continued on the next line. The default maximum line length is 1024 characters (larger values may be specified.)

-match STRING
print only lines containing the character string STRING.

-match /REGEXP/
print only lines containing a string matching the regular expression REGEXP. See the writeup of the qedx command for the definition of regular expressions.

-name NAME, -nm NAME
take NAME literally, even if it is all numeric or begins with "-".

-no_archive, -nac
even if the file being printed is an archive, do not print headings for individual archive components; treat it as a single segment for line numbering and heading.

-no_chase
do not include links when processing starnames. This is the default.

-no_header, -nhe
suppress the header before segments or archive components. This is the default if only one pathname is given and other control arguments are used.

-no_vertsp
simulate formeed and vertical tab characters by outputting newline characters.

-number, -nb
print line numbers before each line. The line number and the spaces separating it from the line take up 10 spaces.

-page_length N, -pl N
start a new page by inserting a formfeed character after every N lines of the file are printed. The default is no pagination. See "Notes" below.

-phys_page_length N, -ppl N
determines how many newline characters should be inserted between pages when -no_vertsp is specified. N is the number of lines on a whole physical page of paper. The default value for N is 66. See "Notes" below.

-right_col N, -rc N
don't print columns past N. Lines extending past column N are truncated on the right. The default is to print all columns.

-stop, -sp
pause after each page until the user types a newline. Also pause before the first page.

-to N
stop printing with line number N. The default is to print all lines.

-to /REGEXP/
stop printing with the first line matching the regular expression REGEXP. The search for REGEXP begins after the first line printed. See the writeup of the qedx command for the definition of regular expressions.

-to_page N
stop printing after the N'th page. The default is to print the whole file.

-vertsp
send formfeed and vertical tab characters to the terminal. This is the default.

-wait, -wt
pause before the first page until the user types a newline.

NOTES

The -page_length control argument works with the -phys_page_length control argument to eject the proper amount of spacing between pages. For example:

```
pr test_file -pl 40 -no_vertsp
```

prints 40 lines of the segment test_file, and uses the default value for -phys_page_length of 66 to emit 26 blank lines before the next 40 lines are printed. If the printer paper is positioned so that text begins printing on the 13th line, then there will be even amounts of leading and trailing space on each page.

If any of -right_col, -line_length, -page_length, or -phys_page_length is specified, or -left_col is > 1, printing is done via the printer conversion software: overstrikes are replaced by multiple lines separated by CR (015) characters, and other control characters are ignored.

Numeric arguments are processed specially for compatibility with previous versions of this command. If no file name has been found, a number is interpreted as a file name; other numeric arguments are interpreted as -from and -to in that order. The -name control argument can be used to indicate that a number is intended as a pathname.

More than one -match control argument and more than one -exclude control argument can be specified; a line is printed if any of the -match arguments select it, unless one of the -exclude arguments prevents it from being printed.

EXAMPLES

The command line:

```
print xyz
```

prints the segment or multisegment file xyz. A header is printed.

The command line:

```
print **.archive -match "/bit (fixed/" -nb -he
```

scans all archive segments in the current working directory for lines matching the regular expression /bit (fixed/. Lines matching this regular expression are printed, with a line number giving the position in the archive component. Each new archive component is preceded by a header that names the component and gives its date of modification.

The command line:

```
print abc::**
```

prints all components of abc.archive. Headers are printed for each component by default.

Name: print_messages, pm

SYNTAX AS A COMMAND

```
pm {address} {-control_args}
```

FUNCTION

prints any interactive messages that were received (and saved in the user's mailbox) while the user was not accepting messages, not logged in, or accept_messages -hold was in effect.

ARGUMENTS

address
is the address of a mailbox. If no address is specified, the user's default mailbox is assumed. The mailbox must be specified in one of the following forms:

STR
is any argument that does not begin with a minus sign (-). If it contains either of the characters > or < it is interpreted as a mailbox pathname (the .mbx suffix is added if not present); otherwise it is interpreted as a User_id.

-pathname PATH, -pn PATH
specifies the pathname of the mailbox. The .mbx suffix is assumed if it is not present.

CONTROL ARGUMENTS

-all, -a
prints all messages, including those held by the accept_messages -hold control argument.

-call cmdline
instead of printing messages in the default format, the command processor is called with a command line of the form:

```
cmdline number sender time message {path}
```

where:

cmdline
is any Multics command line; cmdline must be enclosed in quotation marks 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.

-last, -lt
reprints only the latest message received.

-long, -lg
prints the sender and date-time of every message.

-new
when the accept_messages -hold control argument is in effect, prints only those messages that have not been printed before. The default is to print all messages.

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

NOTES

A default mailbox is created automatically the first time a user issues either print_mail, read_mail, accept_messages, or print_messages. The default mailbox is:

```
>udd>Project_id>Person_id>Person_id.mbx
```

Messages are deleted after they are printed (unless accept_messages -hold is in effect). However, the last message remains available for the life of the process or until replaced by a new message. See also the descriptions of last_message, last_message_sender, and last_message_time.

If you are deferring messages, it is a good practice to use the print_messages command periodically to print out pending messages.

Name: 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.

NOTES

The print_motd command records the information it needs to display changes to the message of the day in the user's default value segment. The value segment is created if necessary and is normally the segment:

```
>udd>Project_id>Person_id>Person_id.value
```

but can be changed by the value_set_path command.

The first time print_motd is used, it prints the entire contents of the message-of-the-day segment. Subsequent uses only print those lines that have been modified or added to the message of the day since the last use of the command.

Name: print_request_types, prt

SYNTAX AS A COMMAND

```
prt {rqt_names} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[prt {rqt_names} {-control_args}]
```

FUNCTION

prints information about request types handled by I/O Daemons. When invoked as an active function, prt returns the names of the selected request types which would have been printed.

ARGUMENTS

rqt_names
: are the names of request types to be printed. The star convention is allowed.

CONTROL ARGUMENTS

-access_name User_id, -an User_id
: prints information about request types serviced by the I/O driver process identified by User_id. See Notes below.

-brief, -bf
suppresses printing of a heading line.

-directory PATH, -dr PATH
specifies the pathname of a test directory to be used in place of the IO Daemon Directory (>ddd>idd). prt looks for an iod_working_tables segment in this directory.

-generic_type STR, -gt STR
lists request types of generic type STR. This can be used to support site-defined generic types.

-plot
prints information about request types associated with the plotter generic type.

-print, -pr
prints information about request types associated with the printer generic type.

-punch, -pch
prints information about request types associated with the punch generic type.

-user_defined, -udf
prints information about request types for which user-defined output control argument settings have been defined using the eor command. The printed output includes both the user-defined request type name and its target request type name. When used as an active function, only the user-defined request type name is returned.

NOTES

The User_id argument specified after -access_name 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.

The enter_output_request command allows the user to define named groups of default control argument settings. The names of these groups can be referenced as if they were user-defined request types. These names are shown in the output of the prt command, indented under the request type to which they apply. Also, the prt active function returns the names of any user-defined request types which match the selection criteria. Refer to the description of the enter_output_request command for further discussion of default control argument settings, and creation of user-defined request type names via eor -default_name.

Name: 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.

Name: 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 Programmers' Reference Manual.

Name: print_terminal_types, ptt

SYNTAX AS A COMMAND

```
print_terminal_types {path}
```

FUNCTION

prints the names of all terminal types defined in the terminal type table (TTT) currently in use. If the TTT being used is not the system default TTT, the command prints the current TTT's pathname at the head of the list of terminal names.

ARGUMENTS

path
specifies the pathname of the TTT. If omitted, the current TTT is used.

Name: program_interrupt, pi

SYNTAX AS A COMMAND

```
pi
```

FUNCTION

informs a suspended invocation of an interactive subsystem that the user wishes to abort a subsystem request and reenter the subsystem.

NOTES

To abort a subsystem request, the user uses the quit (or break) key to interrupt execution, and then gives the program_interrupt command. If the subsystem supports the use of the program_interrupt command, it will abort the interrupted request and ask the user for a new one.

If the subsystem does not support the use of program_interrupt, the command will print an error message. The user can then either restart the interrupted operation with the start command, or abort the entire subsystem invocation with the release command.

If there is more than one suspended command in the user's stack, the stack is searched for a program that supports the program_interrupt feature, and any intervening programs is released.

Name: qedx, qx

SYNTAX AS A COMMAND

```
qx {-control_args} {macro_path} {macro_args}
```

FUNCTION

The qedx editor is used to create and edit ASCII segments. It cannot be called recursively. This description summarizes the editing requests and addressing features provided by qedx. Complete tutorial information on qedx is available in the *qedx Text Editor Users' Guide*, Order No. CG40.

ARGUMENTS

macro_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 macro_path to obtain the complete pathname of the segment containing the qedx instructions.

macro_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 macro_path argument.

  The editor executes the qedx requests contained in the specified segment and then waits for you to type further requests. If macro_path is omitted, the editor waits for you to type a qedx request.

CONTROL ARGUMENTS

-pathname path, -pn path
: causes qedx to read the segment specified by path into buffer 0, simulating "r path", before executing a macro (see macro_path). This control argument must precede macro_path. If no macro is specified, the user is placed immediately in the editor request loop.

-no_rw_path
: prevents the user from making read (r) or write (w) requests with a pathname. All read and write requests for buffer 0 affect the pathname specified by the -pathname control argument. The -no_rw_path control argument is intended to be used within exec_coms which are providing a limited environment; the user is prevented from examining or altering segments other than the one specified with -pathname.

NOTES

Once the qedx editor is invoked, the user can immediately begin to issue qedx requests from the terminal. Requests fall into one of two general categories, input requests and edit requests. Input requests place the editor into input mode, and allow the user to enter new ASCII text from the terminal until an appropriate escape character sequence is typed to switch the editor back to edit mode. Edit requests allow the user to read and write ASCII segments and perform various editing functions on ASCII data. Input and editing operations are not performed directly on the target segments but in a temporary workspace known as a buffer.

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.

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 ($).

An address can be formed using a combination of techniques (`/foo/+5` means the line that is 5 lines ahead of the first line that contains the string "foo"). To specify a series of lines, two adresses must be given in the following general form:

```
ADR1,ADR2
```

The pair of addresses specifies the series of lines starting with the line addressed by ADR1 through the lines addressed by ADR2 inclusive. When a comma is used to separate addresses, the address computation of the second address is unaffected by the computation of the first address (i.e., the value of "." is not changed by the evaluation of the first address). However, if a semicolon is used to separate addresses instead of a comma, the value of "." is set to the line addressed by ADR1 before the evaluation of ADR2 begins. For example, the address pair:

```
/abc/;.+10
```

is equivalent to the address pair:

```
/abc/,/abc/+10
```

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 to 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 CURRENT LINE

All editor requests that alter the contents of the buffer or cause information to be output on the user's terminal change the value of "." (i.e., the current line). Usually, the value of "." is set to the last address specified (either explicitly or by default) in the editor request. The one major exception to this rule is the delete request, which sets "." to the line after the last line deleted. (If the line deleted was the last one in the buffer, then "." is set to "$+1".)

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 following request descriptions contain a brief function, the request format, the default if no ADR is given, and the value of "." after the request is given. 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

The editor can be placed in input mode with the use of the following input requests. The input request is followed by the literal text to be input in the buffer and can contain any number of ASCII lines. To exit from input mode and terminate the input request, the escape sequence `\f` is typed, usually as the first characters of a new line. The `\f` sequence can be followed immediately with other editor requests on the same line.

append (a)
appends lines typed from the terminal after a specified line.

Format:
```
ADRa
TEXT
\f
```

Default: `.a`

Value of ".": Set to last line appended.

change (c)
replaces the specified line or lines with lines typed from the terminal.

Format:
```
ADR1,ADR2c
TEXT
\f
```

Default: `.,.c`

Value of ".": Set to last line entered from the terminal.

insert (i)
inserts lines typed from the terminal before a specified line.

Format:
```
ADRi
TEXT
\f
```

Default: `.i`

Value of ".": Set to last line inserted.

LIST OF BASIC EDIT REQUESTS

delete (d)
Deletes specified line or lines from the buffer.

Format: `ADR1,ADR2d`

Default: `.,.d`

Value of ".": Set to line immediately following the last line deleted.

print (p)
Prints specified line or lines on the terminal; special case print needs address only.

Format: `ADR1,ADR2p`

Default: `.,.p`

Value of ".": Set to last line addressed by the print request (i.e., the last line to be printed.)

print line number (=)
Prints the line number of specified line.

Format: `ADR=`

Default: `.=`

Value of ".": `Set to line addressed by request.`

quit (q) (Q)
Exit from the editor.

The quit request does not itself save the results of any editing that might have been done. If the contents of a modified buffer are to be saved, the write request (w) must be issued.

Format: `q or Q`

Note: The quit request must be followed immediately by a newline character.

read (r)
Appends the contents of a specified ASCII segment after the addressed line.

Format: `ADRr path`

Default: `$r path`

Value of ".": Set to the last line read from the segment.

Note: The request "0r path" is used to insert the contents of a segment before line 1 of the buffer.

substitute (s)
Modifies the contents of the addressed line or set of lines by replacing all strings that match a given regular expression with a specified character string.

Format: `ADR1,ADR2s/regexp/string/`

Default: `.,.s/regexp/string/`

Note: If string contains the special character "&", each "&" is replaced by the characters that matched regexp. The special meaning of "&" can be suppressed by preceding it with the escape sequence `\c`. The escape sequence can also be used in a substitute request to insert a newline. By preceding the newline character (`\012`), or any ASCII character (such as "$", ".", "" or "\"), with `\c`.

The first character after s is the delimiter; it can be any character not appearing in either regexp or string. Strings matching regexp do not overlap and the result of substitution is not rescanned.

write (w)
Writes current buffer into specified segment.

Format: `ADR1,ADR2w {path}`

Default: `1,$w path`

Note: path is the pathname of the segment whose contents are to be the addressed lines in the buffer. If the segment does not already exist, a new segment is created with the specified name. If the segment does already exist, the old contents are replaced by the addressed lines. If path is omitted, a default pathname is used if possible; otherwise an error message is printed. The default pathname is the first pathname used with either a read or write request in this invocation of qedx. The default pathname is set to null if no pathname has been given in this invocation of qedx or if a pathname (either the same one or a different one) is used with a second read request.

LIST OF EXTENDED EDIT REQUESTS

execute (e)

Invokes the Multics command processor without leaving the qedx editor. The remaining characters in the request line are passed to the command processor.

Format: `e <command line>`

Note: If the user wishes to abort a command line invoked with the execute request by issuing the QUIT signal, the program_interrupt (pi) command aborts the command line and restores control to qedx.

global (g)

Prints, deletes, or prints line numbers of all addressed lines that contain a match for a specified regular expression.

Format: `ADR1,ADR2gX/regexp/`

Where X must be one of the following:

d	delete lines containing regexp.
p	print lines containing regexp.
=	print line numbers of lines containing regexp.

Default: `1,$gX/regexp/`

Value of ".": Set to ADR2 of request.

Note: The character immediately following the request X is taken to be the regular expression delimiter and can be any character not appearing in regexp.

exclude (v)

Prints, deletes, or prints line numbers of all addressed lines that do **not** contain a match for a specified regular expression.

Format: `ADR1,ADR2vX/regexp/`

Where X must be one of the following:

d	delete lines not containing regexp.
p	print lines not containing regexp.
=	print line numbers of lines not containing regexp.

Default: `1,$vX/regexp/`

Value of ".": Set to ADR2 of request.

Note: The character immediately following the request X is taken to be the regular expression delimiter and can be any character not appearing in regexp.

LIST OF BUFFER REQUESTS

change buffer (b)
designates an auxiliary buffer as the current buffer. The previously designated current buffer becomes an auxiliary buffer.

Format: `b(X)`

where X is the name of the buffer that is to become the current buffer. A single character buffer name need not be enclosed in parentheses.

Value of ".": Restored to the value of "." when buffer X was last used as the current buffer (i.e., the value of "." is maintained separately for each buffer and saved as part of the buffer status). If X is a new buffer, then "." is set to line 0.

move (m)
moves one or more lines from the current buffer to a specified auxiliary buffer. The addressed lines are deleted from the current buffer and replace the previous contents (if any) of the auxiliary buffer.

Format: `ADR1,ADR2m(X)`

Default: `.,.m(X)`

where X is the name of the auxiliary buffer to which the lines are to be moved. A single character buffer name need not be enclosed in parentheses.

Value of ".": Set to the line after the last line moved in the current buffer. Set to line 0 in the specified auxiliary buffer.

buffer status (x)
prints a summary status of all buffers currently in use.

Format: `x`

Value of ".": Unchanged.

Example: If the user has created the additional buffers alpha and beta and has designated alpha as the current buffer, the output from the buffer status request might be as follows:

```
157     (0)        demo.runoff
 32   ->(alpha)
 53     (beta)
```

This output indicates 157 lines in buffer 0 (the initial buffer), 32 lines in alpha (the current buffer) and 53 lines in beta. It also indicates that the default pathname for buffer 0 is demo.runoff (in the user's working directory) and that buffers alpha and beta have no default pathnames.

nothing (n)

Addresses a line in the segment (i.e., set the value of "." to a particular line). No other action is taken.

Format: `ADRn`

Default: `.n`

Value of ".": Set to line addressed by request.

comment (")

The editor ignores the remainder of this request line. This request is generally used to annotate qedx macros and can also be used to annotate online work.

Format: `ADR" <comment text>`

Default: `." <comment text>`

Value of ".": Set to line addressed by ADR.

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. 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 macro_path macro_args
```

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) 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.

Name: query

SYNTAX AS A COMMAND

```
query arg {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[query arg {-control_args}]
```

FUNCTION

asks the user a question and prints or 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.

CONTROL ARGUMENTS

-brief, -bf
suppresses extra spacing and newlines when asking questions.

-disable_cp_escape, -dcpe
disables the ability to escape to the command processor via the ".." response. See "Notes on command processor escape" below.

-enable_cp_escape, -ecpe
enables the ability to escape to the command processor via the ".." response. See "Notes on command processor escape" below.

-input_switch STR, -isw STR
specifies the I/O switch to use for input of the user's response. The default is user_input.

-long, -lg
adds a leading newline and three trailing spaces to the question. This is the default.

-output_switch STR, -osw STR
specifies the I/O switch to use for output of the question to the user. The default is user_output.

-repeat DT, -rp DT
repeats the question every DT if the user has not responded where DT must be in a form suitable for input to the convert_date_to_binary_ subroutine.

NOTES

The format_line active function can be used to insert other active function values into the question.

NOTES ON COMMAND PROCESSOR ESCAPE

The -disable_cp_escape and -enable_cp_escape control arguments override the system or subsystem default. The system default is "enabled". Subsystems may define the default to be either "enable" or "disable". See the command_query_ subroutine for details.

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
&if &[query [format_line "Do you want the default date of ^a?" [date]]]
&then
&else &quit
```

Name: read_mail, rdm

SYNTAX AS A COMMAND

```
rdm {mbx_specification} {-control_args}
```

FUNCTION

selectively lists, prints, deletes, saves and forwards messages and mail sent to a mailbox. Once the command is invoked, you are placed in the rdm subsystem, where you must use the rdm requests listed below.

ARGUMENTS

mbx_specification
specifies the mailbox to be printed. If not given, the user's default mailbox (>udd>Project>Person>Person.mbx) is used.

LIST OF MBX_SPECIFICATIONS

-mailbox path, -mbx path
specifies the pathname of a mailbox. The suffix "mbx" is added if necessary.

-user Person_id.Project_id
specifies the given user's default mailbox. This control argument is equivalent to:

```
-mailbox >udd>Project_id>Person_id>Person_id.mbx
```

-save path, -sv path
specifies the pathname of a savebox. The suffix "sv.mbx" is added if necessary.

-log
specifies the user's logbox and is equivalent to:

```
-mailbox >udd>Project_id>Person_id>Person_id.sv.mbx
```

STR
is any non-control argument and is first interpreted as `-mailbox STR`. If no mailbox is found, this specification is then interpreted as `-save STR`. If no savebox is found, this specification is then interpreted as `-user STR`.

CONTROL ARGUMENTS

Complete descriptions of control arguments can be found in the Commands and Active Functions manual.

For a description of the message specifiers, selection control arguments, and addresses used by the individual read_mail requests, type:

```
help message_specifiers.gi
help selection_control_args.gi
help addresses.gi
```

within the read_mail request loop.

LIST OF REQUESTS

In the following summary of read_mail requests, "spec" is used as shorthand for "message_specifier", "-selca" is used as shorthand for "-selection_args" and "-ca" is used as shorthand for "-control_args". For a complete description of any request, issue the read_mail request: `help request_name`

.
prints a line describing the current invocation of read_mail.

?
prints a list of requests available in read_mail.

`all -ca, [all -ca]`
prints/returns the message numbers of all messages of the specified type in the mailbox.

`append {specs} path -ca, app {specs} path -ca`
writes the ASCII representation of the specified messages to the end of a segment.

`copy {specs} path {-ca}`
copies the specified messages into another mailbox.

`current, c, [current], [c]`
prints/returns the current message number.

`delete {specs} {-ca} {-selca},`
`dl {specs} {-ca} {-selca},`
`d {specs} {-ca} {-selca}`
deletes the specified messages.

`first -ca, f -ca, [first -ca], [f -ca]`
prints/returns the message number of the first message of the specified type in the mailbox.

`forward {spec} {addresses} {-ca},`
`fwd {spec} {addresses} {-ca},`
`for {spec} {addresses} {-ca}`
forwards the specified message to the specified recipients.

`help {topics} {-ca}`
prints information about read_mail requests and other topics.

`last {-ca}, l {-ca}, [last {-ca}], [l {-ca}]`
prints/returns the message number of the last message of the specified type in the mailbox.

`list {specs} {-ca} {-selca}, ls {specs} {-ca} {-selca},`
`[list {specs} {-ca} {-selca}], [ls {specs} {-ca} {-selca}]`
displays a summary of the selected messages or returns their message numbers.

`list_help {topics}, lh {topics}`
displays the name of all read_mail info segments on given topics.

`list_requests {STRs} {-ca}, lr {STRs} {-ca}`
prints a brief description of selected read_mail requests.

`log {specs} {-ca}`
places a copy of the specified messages into the user's logbox.

`mailbox, mbx, [mailbox], [mbx]`
prints/returns the absolute pathname of the mailbox being read.

`next {-ca}, [next {-ca}]`
prints/returns the message number of the first message of the specified type after the current message.

`preface {specs} pathname {-ca}, prf {specs} pathname {-ca}`
writes the ASCII representations of the specified messages to the beginning of a segment.

`previous {-ca}, [previous {-ca}]`
prints/returns the message number of the last message of the specified type before the current message.

`print {specs} {-ca} {-selca},`
`pr {specs} {-ca} {-selca},`
`p {specs} {-ca} {-selca}`
prints the specified messages.

`print_header {specs} {-ca} {-selca}, prhe {specs} {-ca} {-selca}`
prints the specified messages' headers.

`quit {-ca}, q {-ca}`
exits read_mail.

ready, rdy
prints a Multics ready message.

ready_off, rdf
disables printing of a ready message after each request line.

ready_on, rdn
enables printing of a ready message after each request line.

`reply {specs} {-ca} {addresses}, rp {specs} {-ca} {addresses}`
creates a send_mail invocation to answer the specified messages.

`retrieve {specs} {-selca}, rt {specs} {-selca}`
retrieves the specified deleted messages.

`save {specs} path {-ca}, sv {specs} path {-ca}`
places a copy of the specified messages into a save mailbox.

`subsystem_name, [subsystem_name]`
prints/returns the name of this subsystem

`subystem_version, [subsystem_version]`
prints/returns the version number of this subsystem.

`write {specs} path {-ca}, w {specs} path {-ca}`
writes the ASCII representation of the specified messages to the end of a segment.

Name: ready, rdy

SYNTAX AS A COMMAND

```
rdy
```

FUNCTION

prints 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 the ready_on, ready_off, and general_ready commands.

EXAMPLES

```
r 9:47 3.61 29

r 15:03 .47 12 Level 2
```

Name: 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 the ready, ready_on, and general_ready commands.

Name: 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 the ready, ready_off, and general_ready commands.

Name: release, rl

SYNTAX AS A COMMAND

```
rl
```

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.

Name: rename, rn

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.

Name: reprint_error, re

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

-all, -a
prints messages corresponding to all existing sets of condition information.

-brief, -bf
prints the short form of the message.

-depth N, -dh N
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.

-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

Name: send_mail, sdm

SYNTAX AS A COMMAND

```
sdm {addresses} {-control_args}
```

FUNCTION

transmits a message to one or more recipients. It accepts either an existing segment or 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.

In the default case, send_mail prompts for a subject line ("Subject:") and the message text ("Message:"). After the message text has been typed, it can be terminated by a "." on a line by itself to send mail, or by a "\f" to invoke the qedx editor to modify the message (see the qedx request below), or by a "q" f to enter the send_mail request loop.

ARGUMENTS

addresses
specifies the primary recipients of the message. By default, the message has no primary recipients.

LIST OF ADDRESSES

-mailbox path, -mbx path
specifies the pathname of a mailbox. The suffix "mbx" is added if necessary.

-user Person_id.Project_id
specifies the given user's default mailbox. This control argument is equivalent to:

```
-mailbox >udd>Project_id>Person_ud>Person_id.mbx
```

-save path, -sv path
specifies the pathname of a savebox. The suffix "sv.mbx" is added if necessary.

-log
specifies the user's logbox and is equivalent to:

```
-mailbox >udd>Project_id>Person_id>Person_id.sv.mbx
```

STR -at System
is valid only on systems connected to the ARPA network and specifies an address on another computer system. STR identifies the user (or group of users) to receive the message and is not interpreted in any way by the local system. System identifies a remote system defined in the local system's host table; no distinction is made between upper and lower case characters in the host name.

STR
is any non-control argument. If STR contains either "<" or ">", it is interpreted as `-mailbox STR` otherwise, it is interpreted as `-user STR`.

-comment STR, -com STR
must appear immediately following one of the above forms of an address and supplies additional descriptive information about the address such as a user's full name.

CONTROL ARGUMENTS

A complete list of control arguments can be found in the Commands and Active Functions manual.

-acknowledge, -ack
requests an acknowledgement from the recipients when they read the message.

-no_acknowledge, -nack
does not request an acknowledgement. (Default)

-cc addresses
specifies the secondary recipients of the message. By default, the message has no secondary recipients.

-fill, -fi
reformats the message text according to "fill-on" and "align-left" mode in compose. The message is reformatted after initial input is completed and after each execution of the qedx and apply requests. (Default for terminal input)

-no_fill, -nfi
does not reformat the message text unless the fill request or -fill control argument of the qedx and apply requests is used. (Default for file input)

-from addresses
specifies the authors of the message. By default, the user issuing the send_mail command is the sole author of the message.

-in_reply_to STR, -irt STR
adds an In-Reply-To field containing STR to the message header.

-input_file path, -if path
takes the message text from the specified file rather than from the terminal.

-terminal_input, -ti
accepts the message text from the terminal. See "Notes on terminal input" below. (Default)

-message_id, -mid
adds a Message-ID field to the message header which uniquely identifies the message.

-no_message_id, -nmid
does not add a Message-ID field. (Default)

-reply_to addresses, -rpt addresses
specifies the list of recipients who will receive any replies to this message instead of the message's authors. By default, the authors of the message receive any replies.

-subject STR, -sj STR
specifies the subject of the message. By default, send_mail will prompt the user for the subject.

-no_subject, -nsj
specifies that the message has no subject.

-to addresses
specifies additional primary recipients of the message.

NOTES ON TERMINAL INPUT

By default or if -terminal_input is specified, send_mail issues the prompt "Message:" and reads the message text from the terminal.

If the user terminates the text with a line containing just a period (.), send_mail will reformat the message unless -no_fill is specified on the command line and attempt to send it to the specified recipients unless -request or -request_loop was also specified on the command line. If any errors occur while sending the message, send_mail will enter its request loop to allow the user to correct the problem.

If the user terminates the text with a line containing "\f" anywhere on the line, send_mail will enter the qedx editor on the message text. Any characters on the line after the "\f" are treated as qedx requests. Type:

```
help qedx
```

within send_mail for more information on the qedx request.

If the user terminates the text with a line containing "\q" anywhere on the line, send_mail will reformat the message unless -no_fill is specified on the command line and enter its request loop. Any characters on the line after the "\q" are ignored with a warning message.

NOTES ON ADDRESSES ON THE SEND_MAIL COMMAND LINE

Successive uses of the -to, -cc, -from, and -reply_to control arguments do not override previous uses. Instead the addresses specified in the multiple uses are merged to form the actual list.

For example:

```
sdm Gary.Multics -from Olin.PDO -to Dave.Multics
```

will send the message to Gary.Multics and Dave.Multics. The author of the message will be Olin.PDO.

LIST OF REQUESTS

In the following summary of send_mail requests, "-ca" is used as shorthand for "-control_args". For a complete description of any request, refer to the Commands and Active Functions manual or issue the send_mail request: `help request_name`

.
prints a line describing the current invocation of send_mail.

?
prints a list of requests available in send_mail.

append path, app path
writes the ASCII representation of the message to the end of a segment.

`cc {addresses}`
prints or updates the list of secondary recipients of the message.

copy path, cp path
copies the message into the specified mailbox.

`fill {-ca}, fi {-ca}`
reformats the text of the message.

`from {addresses}`
prints or updates the list of authors of the message.

`help {topics} {-ca}`
prints information about send_mail requests and other topics.

`in_reply_to {STRs}, irt {STRs}`
prints or changes the content of the message's In-Reply-To field.

`list_help {topics}, lh {topics}`
displays the name of all send_mail info segments on given topics.

`list_requests {STRs} {-ca}, lr {STRs} {-ca}`
prints a brief description of selected send_mail requests.

log
places a copy of the message into the user's logbox.

message_id, mid
prints the unique identifier of the message and causes inclusion of a Message-ID field in the message.

`print {-ca}, pr {-ca}, p {-ca}`
prints the message.

`print_header {-ca}, prhe {-ca}`
prints the header of the message.

`qedx {-ca}, qx {-ca}`
edits the message text and header using the Multiccs qedx editor.

`quit {-ca}, q {-ca}`
exits send_mail.

ready, rdy
prints a Multics ready message.

ready_off, rdf
disables printing of a ready message after each request line.

ready_on, rdn
enables printing of a ready message after each request line.

`remove {addresses} {-ca}, rm {adresses} {-ca}`
deletes addresses from the list of primary/secondary recipients, authors, or reply recipients and/or deletes the Subject, Message-ID, and/or In-Reply-To field.

`reply_to {addresses}, rpt {addresses}`
prints or updates the list of recipients of any replies to this message.

save path, sv path
places a copy of the message into the specified save mailbox.

`send {addresses} {-ca}`
delivers the message.

`subject {STRs}, sj {STRs}, [subject], [sj]`
prints, changes, or returns the subject of the message.

`subsystem_name, [subsystem_name]`
prints/returns the name of this subsystem

`subystem_version, [subsystem_version]`
prints/returns the version number of this subsystem.

`to {addresses}`
prints or updates the list of primary recipients of the message.

`write path {-ca}, w path {-ca}`
writes the ASCII representation of the message to the end of a segment.

The following requests may only be used within an invocation of send_mail that was created by use of the read_mail reply request. In this summary, "specs" is short for "message_specifiers" and "-c/sa" is short for "-control_args -selection_args":

`list_original {specs} {-c/sa}, lso {specs} {-c/sa},`
`[list_original {specs} {-c/sa}], [lso {specs} {-c/sa}]`
displays a summary of the messages being answered or returns their message numbers.

`log_original {specs} {-ca}, logo {specs} {-ca}`
places a copy of the messages being answered into the user's logbox.

`print_original {specs} {-c/sa}, pro {specs} {-c/sa}`
prints the messages being answered.

`print_original_header {specs} {-c/sa}, prohe {specs} {-c/sa}`
prints the message headers of the messages being answered.

`save_orginal {specs} path {-ca}, svo {specs} path {-ca}`
places a copy of the messages being answered into a save mailbox.

`write_original {specs} path {-ca}, wo {specs} path {-ca}`
writes the ASCII representation of the messages being answered to the end of a segment.

Name: send_message, sm

SYNTAX AS A COMMAND

`sm address {message}`

FUNCTION

sends messages (one or more, always sent one line at a time) to a given user on a given project.

ARGUMENTS

address
is the address of a mailbox. The mailbox must be specified in one of the following forms:

User_id
is of the form Person_id.Project_id to indicate the user of the mailbox.

-pathname path, -pn path
specifies the pathname of the mailbox. The .mbx suffix is assumed if it is not present.

message
is an optional string. 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 carriage return. In this case, input is terminated by a line consisting solely of a period.

NOTES

If the recipient is accepting messages (see the accept_messages and defer_messages commands), send_message immediately prints each message on the recipient's terminal after the line is sent. The user can also receive messages while in send_message input mode, and can therefore carry on an interactive conversation with a single invocation of the command.

Parentheses, quotes, brackets, and semicolons have their usual command language interpretation in the command line, although they do not on any succeeding message lines.

EXAMPLES

If WJones on the Alpha project sends the following to RTSmith on the Beta project by using the command line:

```
sm RTSmith.Beta I need access to your lsg command
```

the message prints on RTSmith's terminal (if RTSmith is accepting messages) as follows:

```
From WJones.Alpha 04/20/82 1200.6 mst Tue:
I need access to your lsg command
```

The command line:

```
sm RTSmith.Beta Testing complete; install this week
```

sends:

```
From WJones.Alpha 04/20/82 1203.4 mst Tue:
Testing complete
```

and then prints the error message "Segment install not found." because the characters typed after the semicolon are interpreted as another command line. However, semicolon works fine in input mode:

```
sm RTSmith.Beta
Input:
Testing complete;
install this week
.
```

The command line:

```
sm RTSmith.Beta so long (for now)
```

sends two lines:

```
so long for
so long now
```

In the above example, the sender's intended message would have been sent if it had been enclosed in quotation marks (e.g., "so long (for now)").

Name: set_acl, sa

SYNTAX AS A COMMAND

```
sa path model {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 Programmers' Reference Manual for a discussion of ACLs.

ARGUMENTS

path
: is the pathname of a segment, multisegment file, or directory. If it is -wd or -working_directory, 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 model.

modei
: are valid access modes. 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
are access control names 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 "Examples" below.) If no match is found and all three components are present, an entry is added to the ACL. If the last modeN has no User_id following it, the Person_id of the user and current Project_id are assumed.

CONTROL ARGUMENTS

-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.

-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. (Default)

-no_sysdaemon, -nsd
suppresses the addition of a "rw *.SysDaemon.*" term when using -replace.

-replace, -rp
deletes all ACL terms (with the exception of a default "rw *.SysDaemon.*" term unless -no_sysdaemon is specified) before adding the terms specified on the command line. The default is to add to and modify the existing ACL.

-sysdaemon, -sd
when -replace is specified, adds a "rw *.SysDaemon.*" ACL term before adding the terms specified. (Default)

Either of the following control arguments can be specified to resolve an ambiguous choice between segments and directories that occur only when modeN 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 requires 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

Some examples of User_ids and the ACL entries they match are:

..*	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 ".*.*".

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.

Name: 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_paths
: are search paths 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.

-default, -df
: replaces the search list with its system-defined default. No search_paths can be specified with this control argument.

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.

Name: set__search__rules, ssr

SYNTAX AS A COMMAND

```
set_search_rules {path} {-control_arg}
```

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 must be reset to the default search rules by the -default control argument.

CONTROL ARGUMENTS

-default, -df
: resets the search rules to the default search rules, as set for a new process.

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
expand into one or more directory pathnames. (An example of a site_defined keyword is system_libraries.) 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.

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.

Name: set__tty, stty

SYNTAX AS A COMMAND

```
stty {-control_args}
```

FUNCTION

modifies the terminal type associated with the user's terminal and/or various parameters associated with terminal I/O. The type as specified by this command determines character conversion and delay timings; it has no effect on communications line control.

CONTROL ARGUMENTS

-all, -a
is the equivalent of specifying the four control arguments -print, -print_edit, -print_frame, and -print_delay.

-brief, -bf
may only be used with the -print control argument and causes only those modes that are on plus those that are not on/off type modes (e.g., l179) to be printed.

-buffer_size N, -bsize N
specifies the terminal's buffer size to be used for output block acknowledgement where N is the terminal's buffer size in characters. (See the discussion of output

flow control in the Programmer's Reference manual.) If the end_of_block and acknowledgement characters have not been specified (either as part of the terminal type description or by means of the -output_etb_ack control argument to set_tty), this control argument may not be specified.

-delay STR, -dly STR
sets the delay timings for the terminal according to STR, which is either the word "default" or a string of six decimal values separated by commas. If "default" is specified, the default values for the current terminal type and baud rate are used. The values specify vert_nl, horz_nl, const_tab, var_tab, backspace, and vt_ff, in that order. (See "List of delay types" below.)

-edit edit_chars, -ed edit_chars
changes the input editing characters to those specified by edit_chars. The edit_chars control argument is a 2-character string consisting of the erase character and the kill character, in that order. If the erase character is specified as a blank, the erase character is not changed; if the kill character is omitted or specified as a blank, the kill character is not changed.

-initial_string, -istr
transmits the initial string defined for the terminal type to the terminal.

-input_flow_control STR, -ifc STR
sets the input_suspend and input_resume characters to those specified in STR, which is a string of one or two characters. (See the discussion of input flow control in the Programmer's Reference manual.) If STR contains two characters, the first character is the input_suspend character and the second one is the input_resume character. If STR contains only one character, it is the input_resume character and there is no input_suspend character.

-io_switch STR, -is STR
specifies that the command be applied to the I/O switch whose name is STR. If this control argument is omitted, the user_i/o switch is assumed.

-modes STR, -md STR
sets the modes for terminal I/O according to STR, which is a string of mode names separated by commas. Many modes can be optionally preceded by "^" to turn the specified mode off. For a list of valid mode names, see "List of modes" below. Modes not specified in STR are left unchanged.

-output_etb_ack STR, -oea STR
sets the output_end_of_block and output_acknowledge characters to those specified in STR, which is a string of two characters. The first character of STR is the end_of_block character and the second one is the acknowledge character. (See the discussion of output flow control in the Programmer's Reference manual.) If a buffer size has not been specified (either as part of the terminal type description or by means of the -buffer_size control argument to set_tty), this control argument may not be specified.

-output_suspend_resume STR, -osr STR
sets the output_suspend and output_resume characters to those specified in STR, which is a string of two characters. The first character of STR is the output_suspend character and the second is the output_resume character.

-print, -pr
prints the terminal type and modes on the terminal. If any other control arguments are specified, the type and modes printed reflect the result of the command.

-print_delay, -pr_dly
prints the delay timings for the terminal.

-print_edit, -pr_ed
prints the input-editing characters for the terminal.

-reset, -rs
sets the modes to the default modes string for the current terminal type.

-terminal_type STR, -ttp STR
sets the terminal type of the user to STR, where STR can be any one of the types defined in the terminal type table (TTT). The default modes for the new terminal type are turned on and the initial string for the terminal type, if any, is transmitted to the terminal. Refer to the print_terminal_types command for information on obtaining a list of terminal types currently in the TTT.

-frame STR, -fr STR
changes the framing characters used in blk_xfer mode to those specified by STR, where STR is a 2-character string consisting of the frame-begin and the frame-end character, respectively. These characters must be specified in the character code of the terminal, and may be entered as octal escapes, if necessary. The frame-begin character is specified as a NUL character to indicate that there is no frame-begin character; the same is true for a frame-end character. These characters have no effect unless blk_xfer mode is on. It is an error to set the frame-end character to NUL if the frame-begin character is not also set to NUL.

-print_frame, -pr_fr
prints the framing characters for the terminal.

LIST OF DELAY TYPES

vert_nl
is the number of delay characters to be output for all newlines to allow for the linefeed (`-127 <= vert_nl <= 127`). If it is negative, its absolute value is the minimum number of characters that must be transmitted between two linefeeds (for a device such as a TermiNet 1200).

horz_nl
is a number to be multiplied by the column position to obtain the number of

delays to be added for the carriage return portion of a newline (0 <= horz_nl <= 1). The formula for calculating the number of delay characters to be output following a newline is:

```
ndelays = vert_nl + fixed (horz_nl*column)
```

const_tab
is the constant portion of the number of delays associated with any horizontal tab character (0 <= const_tab <= 127).

var_tab
is the number of additional delays associated with a horizontal tab for each column traversed (0 <= var_tab <= 1). The formula for calculating the number of delays to be output following a horizontal tab is:

```
ndelays = const_tab + fixed (var_tab*n_columns)
```

backspace
is the number of delays to be output following a backspace character (-127 <= backspace <= 127). If it is negative, its absolute value is the number of delays to be output with the first backspace of a series only (or a single backspace). This is for terminals such as the TermiNet 300 that need delays to allow for hammer recovery in case of overstrikes, but do not require delays for the carriage motion associated with the backspace itself.

vt_ff
is the number of delays to be output following a vertical tab or formfeed (0 <= vt_ff <= 511).

The horz_nl and var_tab values are floating-point numbers; all other values are integers. If any of the six values is omitted, the corresponding delay value is not changed; if values are omitted from the end of the list, trailing commas are not required.

LIST OF MODES

The following is a list of modes which can be set with the -modes control argument. Some modes have a complement indicated by the circumflex character (^) that turns the mode off (e.g., ^erkl). For these modes the complement is displayed with the mode. Normal defaults are indicated for those modes that are generally independent of terminal type. The modes string is processed from left to right. Thus, if two or more contradictory modes appear within the same modes string, the rightmost mode prevails.

8bit, ^8bit
causes input characters to be received without removing the 8th (high-order) bit, which is normally interpreted as a parity bit. This mode is valid for HSLA channels only. (Default is off.)

blk_xfer, ^blk_xfer
: specifies that the user's terminal is capable of transmitting a block or "frame" of input all at once in response to a single keystroke. The system may not handle such input correctly unless blk_xfer mode is on and the set_framing_chars order has been issued. (Default is off.)

breakall, ^breakall
: enables a mode in which all characters are assumed to be break characters, making each character available to the user process as soon as it is typed. This mode only affects get_chars operations. (Default is off.)

can, ^can
: performs standard canonicalization on input. (Default is on.)

can_type=overstrike
: the canonicalization algorithm for use when the user is typing input on a terminal which is capable of displaying several characters in a single column. Canonicalization is only performed when the I/O switch is in "can" mode. This is the default for hard-copy terminals.

can_type=replace
: the canonicalization algorithm for use then the user is typing input on a column. (Examples of these terminals include most modern video (CRT) terminals.) Replacement canonicalization causes the canonical form of typed input to contain only the last character entered in any column. Canonicalization is only performed when the I/O switch is in "can" mode. This is the default for video terminals. See "Examples" below.

capo, ^capo
: outputs all lowercase letters in uppercase. If edited mode is on, uppercase letters are printed normally; if edited mode is off and capo mode is on, uppercase letters are preceded by an escape (\) character. (Default is off.)

crecho, ^crecho
: echoes a carriage return when a line feed is typed. This mode can only be used with terminals and line types capable of receiving and transmitting simultaneously.

ctl_char, ^ctl_char
: specifies that ASCII control characters that do not cause carriage or paper motion are to be accepted as input, except for the NUL character. If the mode is off, all such characters are discarded. (Default is off.)

default
: is a shorthand way of specifying erkl, can, ^rawi, ^rawo, ^wake_tbl, and esc. The settings for other modes are not affected.

echoplex, ^echoplex
: echoes all characters typed on the terminal. The same restriction applies as for crecho; it must also be possible to disable the terminal's local copy function.

edited, ^edited
suppresses printing of characters for which there is no defined Multics equivalent on the device referenced. If edited mode is off, the 9-bit octal representation of the character is printed. (Default is off.)

erkl, ^erkl
performs "erase" and "kill" processing on input. (Default is on.)

esc, ^esc
enables escape processing on all input read from the device. (Default is on.)

force
specifies that if the modes string contains unrecognized or invalid modes, they are to be ignored and any valid modes are to be set. If force is not specified, invalid modes cause an error message to be printed, and no modes are set.

fulldpx, ^fulldpx
allows the terminal to receive and transmit simultaneously. This mode should be explicitly enabled before enabling echoplex mode. (Default is on.)

hndlquit, ^hndlquit
echoes a newline character and performs a resetread of the associated stream when a quit signal is detected. (Default is on.)

iflow, ^iflow
specifies that input flow control characters are to be recognized and/or sent to the terminal. The characters must be set before iflow mode can be turned on.

init
sets all switch type modes off, sets line length to 50, and sets page length to zero.

lfecho, ^lfecho
echoes and inserts a line feed in the user's input stream when a carriage return is typed. The same restriction applies as for crecho.

llN, ^ll
specifies the length in character positions of a terminal line. If an attempt is made to output a line longer than this length, the excess characters are placed on the next line. If ^ll is specified, line length checking is disabled (i.e., no \c's appear). In this case, if a line of more than 255 column positions is output by a single call to iox_$put_chars, some extra white space may appear on the terminal.

no_outp, ^no_outp
causes output characters to be sent to the terminal without the addition of parity bits. If this mode and rawo mode are on, any 8-bit pattern can be sent to the terminal. This mode is valid for HSLA channels only. (Default is off.)

oddp, ^oddp
causes any parity generation that is done to the channel to assume odd parity.

Otherwise, even parity is assumed for line types other than 2741 and 1050. This mode is valid for HSLA channels only. (Default is off.)

`oflow, ^oflow`
specifies that output flow control characters are to be recognized when sent by the terminal. The characters and the protocol to be used must be set before oflow mode can be turned on.

`plN, ^pl`
specifies the length in lines of a page. When an attempt is made to exceed this length, a warning message (which usually defaults to EOP) is printed. When the user types a formfeed or newline character (any break character), the output continues with the next page. EOP is displayed on a new line after N consecutive output lines are sent to the screen (including long lines which are folded as more than one output line). To have the EOP displayed on the screen without scrolling lines off the top, N should be set to one less than the page length capability of the screen. If ^pl is specified, end-of-page checking is disabled. (See description of scroll mode below.)

`polite, ^polite`
does not print output sent to the terminal while the user is typing input until the carriage is at the left margin, unless the user allows 30 seconds to pass without typing a newline. (Default is off.)

`prefixnl, ^prefixnl`
controls what happens when terminal output interrupts a partially complete input line. In prefixnl mode, a newline character is inserted in order to start the output at the left margin; in ^prefixnl mode, the output starts in the current column position. (Default is on.) Polite mode controls when input may be interrupted by output; prefixnl controls what happens when such an interruption occurs.

`rawi, ^rawi`
reads the data specified from the device directly without any conversion or processing. (Default is off.)

`rawo, ^rawo`
writes data to the device directly without any conversion or processing. (Default is off.)

`red, ^red`
sends red and black shifts to the terminal.

`replay, ^replay`
prints any partial input line that is interrupted by output at the conclusion of the output, and leaves the carriage in the same position as when the interruption occurred. (Default is off.)

`scroll, ^scroll`
specifies that end-of-page checking is performed in a manner suited to scrolling

video terminals. If the mode is on, the end-of-page condition occurs only when a full page of output is displayed without intervening input lines. The mode is ignored whenever end-of-page checking is disabled. (Default is off.)

`tabecho, ^tabecho`
echoes the appropriate number of spaces when a horizontal tab is typed. The same restriction applies as for crecho.

`tabs, ^tabs`
inserts tabs in output in place of spaces when appropriate. If tabs mode is off, all tab characters are mapped into the appropriate number of spaces.

`vertsp, ^vertsp`
performs the vertical tab and formfeed functions, and sends appropriate characters to the device. Otherwise, such characters are escaped. (Default is off.)

`wake_tbl, ^wake_tbl`
causes input wakeups to occur only when specified wakeup characters are received. Wakeup characters are defined by the set_wakeup_table order. This mode is ineffective unless breakall mode is also on. This mode cannot be set unless a wakeup table has been previously defined.

NOTES

Invoking the set_tty command causes the system to perform the following steps in the specified order:

1. If the -terminal_type control argument is specified, set the specified type, turn on the default modes for that type and send the initial string for that type.

2. If the -reset control argument is specified, set the modes to the default modes string for the current terminal type.

3. If the -modes control argument is specified, turn on or off those modes explicitly specified.

4. If the -initial_string control argument is specified, transmit the initial string to the terminal.

5. If the -edit control argument is specified, set the editing characters.

6. If the -frame control argument is specified, set the framing characters.

7. If the -delay control argument is specified, set the delay values.

8. If the -input_flow_control control argument is specified, set the input flow control characters.
9. If the -buffer_size, -output_etb_ack, or -output_suspend_resume control argument is specified, set the corresponding output flow control parameters.

10. If the -print control argument is specified, print the type and modes on the terminal.

11. If the -print_edit control argument is specified, print the editing characters on the terminal.

12. If the -print_frame control argument is specified, print the framing characters on the terminal.

13. If the -print_delay control argument is specified, print the delay values on the terminal.

EXAMPLES
The command line:

```
set_tty -delay 6,0,0,0,-6,59
```

sets all six delay values to those used by a TermiNet 300.

The command line:

```
set_tty -delay 5,0.6,,,2,63
```

sets the delay values so that 5 delays will be output with a newline, plus 3 more for every 5 columns of carriage return; 2 delays will be used for each backspace, 63 for a vertical tab or formfeed, and whatever values were already in force for horizontal tabs.

The command line:

```
set_tty -delay ,1.3,,.8
```

sets horz_nl to 1.3 and var_tab to 0.8, while leaving all other delay values as they were before.

The command line:

```
set_tty -frame \002\003
```

sets the frame-begin and frame-end characters to the ASCII STX and ETX characters, respectively.

For example with can_type=replace, typing:

```
This is a tsetBBBest of tpying text.BBBBBBBBBBByp<LF>
```

where B is a backspace character and <LF> is the line-feed character will appear on the screen and be input as:

```
This is a test of typing text.
```

When using can_type=replace, it is not possible to overstrike a character with the erase character. In other words, it is not possible to delete a character in the middle of a typed line without repositioning to the character in question and retyping the rest of the line. Therefore, the user may wish to disable the erase character when using replacement canonicalization. This may be accomplished by the command line:

```
set_tty -edit \400
```

Name: start, sr

SYNTAX AS A COMMAND

```
sr {-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."

Name: status, st

SYNTAX AS A COMMAND

```
st paths {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[st path -control_args {-chase}]
```

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.

-author, -at
prints the author of the entry.

-chase
prints information about the branch targets of links instead of the links themselves. An error occurs for a null link or a link to a null link.

-chase_if_possible, -cip
prints information about the targets of links where branch targets exist, and for null links and links to null links prints information about the ultimate link in the chain. This control argument does not affect the processing of nonlinks.

-date, -dt
prints all the relevant dates on the entry.

-date_time_dumped, -dtd
prints the date-time-dumped by the hierarchy dumper.

-date_time_entry_modified, -dtem
prints the date-time-entry-modified.

-directory, -dr
selects directories when using the star convention.

-link, -lk
selects links when using the star convention.

-name, -nm
prints all the names on the entry.

-no_chase
prints link information about links. (Default)

-no_chase_if_possible, -ncip
prints link information about links. (Default)

-primary, -pri
prints the primary name on the entry.

-segment, -sm
selects segments when using the star convention.

-type, -tp
prints the type of entry: segment, directory, multisegment file, or link.

LIST OF TYPE SPECIFIC CONTROL ARGUMENTS

The following control_args can only be used for segments, multisegment files, and directories.

-access, -ac
prints the user's effective mode, ring brackets, access class (if different from the default), and safety switch (if it is on).

-access_class
prints the access class.

-all, -a, -long, -lg
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 off).

-bc_author, -bca
prints the bit count author of the entry.

-bit_count, -bc
prints the bit count.

-copy_switch, -csw
prints whether the copy switch is on or off.

-current_length, -cl
prints the current length in pages.

-damaged_switch, -dsw
prints whether the damaged switch is on or off.

-date, -dt
prints all the dates on the entry: i.e., date used, date contents modified, date branch modified, date dumped.

-date_time_contents_modified, -dtcm
prints the date-time-contents-modified.

-date_time_used, -dtu
prints the date-time-used.

-date_time_volume_dumped, -dtvd
prints the date-time-dumped by the volume dumper.

-device, -dv
prints the logical volume on which the entry resides.

-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.

-logical_volume, -lv
prints the logical volume on which the entry resides. This control argument is the same as the -device control argument.

-max_length, -ml
prints the maximum length of a segment.

-mode, -md
prints the user's effective mode.

-records, -rec
prints the records used.

-ring_brackets, -rb
prints the ring brackets.

-safety_switch, -ssw
prints whether the safety switch is on or off.

-unique_id, -uid
prints the entry's unique identifier.

LIST OF CONTROL ARGUMENTS FOR SEGMENTS

The following control arguments can only be used 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.

-usage_count, -use
prints the number of page faults taken on the segment since creation.

LIST OF CONTROL ARGUMENTS FOR LINKS
The following control arguments can only be used for links.

-long, -lg
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 Programmers' Reference Manual.

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 -long

   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
```

Name: switch_off, swf

SYNTAX AS A COMMAND

```
swf keyword paths {-control_args}
```

FUNCTION

turns off a specified switch for one or more entries. For an MSF, the switch of the MSF directory (when possible) and those of all the components are turned off.

ARGUMENTS

keyword
specifies the name of a switch. See "List of keywords" below.

paths
are the pathnames of segments, MSF's and directories for which it is possible to set the specified switch. The star convention is allowed, and includes links only if -chase is specified. A pathname that looks like a control argument or contains starname special characters not meant to be matched can be specified by "-name STR" or "-nm STR".

CONTROL ARGUMENTS

-chase
includes links and chases them when using the star convention.

-name STR, -nm STR
specifies a pathname that looks like a control argument or contains starname special characters not meant to be matched.

-no_chase
does not include links when using the star convention. (Default)

ACCESS REQUIRED

The user requires modify permission on the parent directory.

LIST OF KEYWORDS

copy_switch, csw
(segments) If ON, allows processes lacking write access to modify a copy of the segment in the process directory.

damaged_switch, dsw
(segments) If ON, the segment is assumed to have been damaged by a device error or system crash.

complete_volume_dump_switch, cvds
If ON, the entry is dumped during a complete volume dump of the physical volume on which it resides.

incremental_volume_dump_switch, ivds
If ON, the entry is dumped during an incremental dump cycle of the volume dumper.

perprocess_static_switch, ppsw
(object segment) If ON, the segment's internal static storage is not initialized when a run unit is created.

safety_switch, ssw
If ON, the delete command and delete_ subroutine query the user before deleting the entry.

Name: switch_on, swn

SYNTAX AS A COMMAND

```
swn keyword paths {-control_args}
```

FUNCTION

turns on a specified switch for one or more entries. For an MSF, the switch of the MSF directory (when possible) and those of all the components are turned on.

ARGUMENTS

keyword
specifies the name of a switch. See "List of keywords" below.

paths
are the pathnames of segments, MSF's and directories for which it is possible to set the specified switch. The star convention is allowed, and includes links only if -chase is specified. A pathname that looks like a control argument or contains starname special characters not meant to be matched can be specified by "-name STR" or "-nm STR".

CONTROL ARGUMENTS

-chase
includes links and chases them when using the star convention.

-name STR, -nm STR
specifies a pathname that looks like a control argument or contains starname special characters not meant to be matched.

-no_chase
does not include links when using the star convention. (Default)

ACCESS REQUIRED

The user requires modify permission on the parent directory.

LIST OF KEYWORDS

copy_switch, csw
(segments) If ON, allows processes lacking write access to modify a copy of the segment in the process directory.

damaged_switch, dsw
(segments) If ON, the segment is assumed to have been damaged by a device error or system crash.

complete_volume_dump_switch, cvds
If ON, the entry is dumped during a complete volume dump of the physical volume on which it resides.

incremental_volume_dump_switch, ivds
If ON, the entry is dumped during an incremental dump cycle of the volume dumper.

perprocess_static_switch, ppsw
(object segment) If ON, the segment's internal static storage is not initialized when a run unit is created.

safety_switch, ssw
If ON, the delete command and delete_ subroutine query the user before deleting the entry.

Name: time

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 the convert_date_to_binary_ subroutine. If no argument is specified, the current time is used. The DT argument is concatenated to form a single string even if it contains spaces, and need not be quoted.

Name: unlink, ul

SYNTAX AS A COMMAND

```
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 the Programmers' Reference Manual.

Name: where, wh

SYNTAX AS A COMMAND

```
wh names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wh name {-control_args}]
```

FUNCTION

uses the standard search rules to search for a given segment or entry point.

ARGUMENTS

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.

-brief, -bf
prints only the pathname of each entry found. (Default)

-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.

-inhibit_error, -ihe
does not print an error message if no segments can be found for a given name. For the where command, no output is printed; for the active function, null string is returned.

-long, -lg
prints the name of the search rule used to find each segment and the user's effective access to the segment, in addition to the pathname. This control argument is incompatible with -all.

-no_inhibit_error, -nihe
prints an error message if no segments can be found for a given name. (Default)

-segment, -sm
searches for segments. This is the default, unless name contains a dollar sign.

NOTES ON ACTIVE FUNCTION

The active function returns the pathname of the segment, as found by the search rules. Only one name can be specified. The -all, -brief and -long control arguments are not allowed. Unless -inhibit_error is specified, an error occurs if no segment can be found.

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"
```

Name: who

SYNTAX AS A COMMAND

```
who {User_ids} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[who {User_ids} {-control_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.)

ARGUMENTS

User_ids
are match names where:
Person_id
lists users with the name Person_id.
.Project_id
lists users with the project name Project_id.
Person_id.Project.id
lists users with the specified person and project.

CONTROL ARGUMENTS

-absentee, -as
lists absentee users. See Notes.

-all, -a
lists all the interactive, absentee, and daemon users.

-brief, -bf
suppresses the printing of the header. Not allowed for the active function.

-daemon, -dmn
lists daemon users. See Notes.

-interactive, -ia
lists interactive users. See Notes.

-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. Not allowed for the active function.

-name, -nm
sorts the output by the name (Person_id) of each user.

-project, -pj
sorts the output by the Project_id of each user.

NOTES

If none of the control arguments -interactive, -absentee, or -daemon is specified, and no User_ids are specified, then all interactive and absentee users are listed. If none of those control_args is specified, but User_ids are specified, then all matching users are listed.

If one or more of -interactive, -absentee, or -daemon is specified, only processes of the selected type(s) are listed. If User_ids are also specified, then only users matching all those control arguments and the User_ids are listed.

Absentee users are denoted in the list by an asterisk (*) following Person_id.Project_id.

Sometimes a Person_id.Project_id returned by the command will be followed by a "D" and/or an "S", where "D" refers to a disconnected process and "S" refers to a suspended process.

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 a User_name is specified, the header is suppressed even if the -long control argument is specified.

It is possible to prevent your own name from being listed by all users' invocations of who; to do this, the user should contact the project administrator.

NOTES ON ACTIVE FUNCTION

The active function returns a list of Person_id.Project_id pairs, requoted and separated by spaces. Control arguments can be used to select and sort.

Name: 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.

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: MULTICS COMMON COMMANDS

ORDER NO. GB58-00

DATED FEBRUARY 1983

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 155 Gordon Baker Road, Willowdale, Ontario M2H 3N7
In the U.K.: Great West Road, Brentford, Middlesex TW8 9DH
In Australia: 124 Walker Street, North Sydney, N.S.W. 2060
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

36210, 7.5C183, Printed in U.S.A.

GB58-00