Multics

Commands and Active Functions

Honeywell Bull

MULTICS

COMMANDS AND ACTIVE FUNCTIONS

SUBJECT

Description of Standard Multics Commands and Active Functions

SPECIAL INSTRUCTIONS

This publication supersedes the previous edition of the manual, Order No. AG92-05, dated January 1983, and its addendum AG92-05A, dated December 1983.

Marginal change indicators (change bars and asterisks) indicate technical changes.

SOFTWARE SUPPORTED

Multics Software Release 11.0

ORDER NUMBER

AG92-06

February 1985

Honeywell Bull

PREFACE

The Multics Commands and Active Functions manual is organized into four sections. Section 1 contains a basic introduction to manual use and term definition. Section 2 contains the standard Multics commands and active functions arranged by function. Section 3 contains the descriptions of those commands and active functions in alphabetical order. Section 4 describes the requests used to gain access to the system.

Throughout this manual, references are made to the *Multics Programmer's Reference Manual* (AG91) and the *Multics Subroutines and I/O Modules* (AG93). For convenience, these are referred to in the text as the Programmer's Reference Manual and the Subroutines manual.

Significant Changes in AG92-06B

The disconnect command was added to section 3.

The following commands changed to improve functionality:

- abbrev
- contents
- dial
- fortran
- help
- print_motd
- switch_on
- switch_off

The following commands have been extensively revised:

- copy_dump_tape
- compare_dump_tape
- read_tape_and_query

The enter_output_request command was revised to satisfy a customer change request.

Honeywell Bull disclaims the implied warranties of merchantability and fitness for a particular purpose and makes no express warranties except as may be stated in its written agreement with and for its customer. In no event is Honeywell Bull liable to anyone for any indirect, special or consequential damages.

The information and specifications in this document are subject to change without notice. Consult your Honeywell Bull Marketing Representative for product or service availability.

Copyright © Honeywell Bull Inc., 1985

File No.: 1L13 AG92-06

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.

You are encouraged to take advantage of the information available in the manual's detailed index and table of contents. The index alphabetically lists programs by name and subject. Cross-references among command descriptions assist in locating programs applicable to a given task.

DESCRIPTION OF MANUAL FORMAT

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

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 in the syntax line are enclosed in braces ({}). The syntax for active functions is always 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}, -control_args).

Keep in mind the difference between a plural argument name that is enclosed in braces (optional) and one that is not (required): if it is enclosed in braces, you need not give any argument of that type; if it is not, you must supply at least one argument of that type. Thus you could write "paths" in a usage line as:

```
path1 {path2...pathN}
```

The convention of using "paths" rather than using the above is merely to save space.

Different arguments that you must give in pairs are numbered:

```
xxx1 yyy1 {...xxxN yyyN}
```

To indicate that you must provide the same generic argument in pairs, the arguments are given letters and numbers:

```
arg1 arg2 {...arg1N arg2N}
```

Some of the standard arguments accepted by commands and active functions are:

STR
: any character string.

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

DT or time_string
: a date-time character string. Examples are "4/25/84 noon est Sun", "November 7", "7:30 pm 10 June 1985", and "midnight". (See "Date/Time Values" below for a description of valid time strings.)

star_name
: any pathname or User_id conforming to the star convention, described under "Star Names" in the Programmer's Reference Manual.

virtual_pointer
: A virtual pointer is a character string representation of a pointer value. It consists of a segment identifier (pathname, reference name, or segment number) and an optional octal offset into the segment. In the table that follows, W is an octal word offset from the beginning of the segment; it can have a value from 0 to 777777 inclusive. B is a decimal bit offset within the word; it can have a value from 0 to 35 inclusive. The possible forms are:

  path|W(B)
  : points to the octal word W, decimal bit B, of the segment or multisegment file (MSF) identified by absolute or relative pathname path. If the path you give identifies a MSF, the offset given is in component 0 of the MSF.

  path|W
  : same as path|W(0).

  path|
  : same as path|0(0).

  path
  : same as path|0(0).

  path|entry_pt
  : points to the word identified by entry point entry_pt in the object file (segment or MSF) identified by path.

dir>entry$entry_pt
points to the word identified by entry point entry_pt in the object file identified by pathname dir>entry.

<dir>entry$entry_pt
points to the word identified by entry point entry_pt in the object file identified by pathname <dir>entry.

<entry$entry_pt
points to the word identified by entry point entry_pt in the object file identified by pathname <entry.

ref_name$entry_pt
points to the word identified by entry point entry_pt in the file whose reference name is ref_name.

ref_name$W(B)
points to the octal word W, decimal bit B, of the segment or MSF whose reference name is ref_name. If ref_name is a reference name on an MSF (i.e., on component 0 of the MSF), the word and bit offsets are applied within component 0.

ref_name$W
same as ref_name$W(0).

ref_name$
same as ref_name$0(0).

segno|W(B)
points to the octal word W, decimal bit B, of the segment whose octal segment number is segno.

segno|W
same as segno|W(0).

segno|
same as segno|0(0).

segno
same as segno|0(0).

segno|entry_pt
points to the word identified by entry point entry_pt in the segment whose octal segment number is segno. If segno identifies component 0 of an object MSF, the pointer returned may not point within the segment identified, since the target of a definition in component 0 of an object MSF will be in another component of the object MSF.

A virtual pointer that does not contain $ or | is interpreted as a pathname if it contains > or <, as a reference name otherwise.

A null pointer is represented by the virtual pointer 77777|1, -1|1, or -1.

virtual_entry
is a character string representation of an entry value. It consists of a segment identifier and an optional offset into the segment. In the table that follows, W is an octal word offset from the beginning of the segment; it can have a value from 0 to 777777 inclusive. The possible forms are:

path|W
entry at octal word W of segment or multisegment file (MSF) identified by absolute or relative pathname path. If the path you give identifies a MSF, the offset given is in component 0 of the MSF.

path|
same as path|0.

path|entry_pt
entry at word identified by entry point entry_pt in the object file (segment or MSF) identified by path.

dir>entry$entry_pt
entry at word identified by entry point entry_pt in the object file identified by pathname dir>entry.

<dir>entry$entry_pt
entry at word identified by entry point entry_pt in the object file identified by pathname <dir>entry.

<entry$entry_pt
entry at word identified by entry point entry_pt in object file identified by pathname <entry.

path
same as path|[entry path].

ref_name$entry_pt
entry at word identified by entry point entry_pt in segment found via search rules whose reference name is ref_name.

ref_name$W
entry at octal word W of the segment or MSF found via search rules whose reference name is ref_name. If ref_name is a reference name on an MSF (i.e., on component 0 of the MSF), the word and bit offsets are applied within component 0.

ref_name$
same as ref_name$0.

ref_name
same as ref_name$ref_name, but like path if it contains > or < characters.

A virtual entry that does not contain $ or | is interpreted as a pathname if it contains > or <, as a reference name otherwise.

A null pointer is represented by the virtual pointer 77777|1, -1|1, or -1.

Use of a pathname in a virtual entry initiates the referenced segment with a reference name equal to its final entryname. Name duplication errors occurring during the initiation are resolved by terminating the previously known name.

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

The list of control arguments give the possible values for -control_args in the syntax line. The long name and the short one (if any) are given. For simplicity, common control argument values are indicated as follows:

STR
any character string; individual command descriptions indicate any restrictions (e.g., must not exceed 136 characters).

This page intentionally left blank.

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

DT
a date-time character string (see "Date/Time Values" below).

ID
a numerical request identifier as described in the Programmer's Reference Manual.

path
the pathname of an entry; unless otherwise indicated, it may be either a relative or an absolute pathname.

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

```
-access_name STR, -an STR
-ring N, -rg N
-date DT, -dt DT
-pathname path, -pn path
```

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

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

GENERAL DEFINITION OF A COMMAND

A command performs some action for you, such as displaying information on your 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. You can invoke commands at the beginning of a command line at command level and can put multiple commands on a single line, with a 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. You invoke an active function inside an active 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 together with the exec_com, abbrev, and do commands to implement command language macros.

When you give multiple commands 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 may affect the values of active functions that appear later in the line.

EXAMPLES OF COMMAND VS ACTIVE FUNCTION USE

You can invoke many programs as either a command or active function. The format of the active function return string is slightly different from the command's printed output. In these examples, and all interactive examples throughout this manual, lines you type are preceded with an exclamation point (!).

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

versus the corresponding status active function:

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

ERRORS

Commands report errors by signaling 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 signaling active_function_error. The default action is to print a message and return to command level. Respond by typing:

```
! release
```

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

The command_error and active_function_error conditions are further described in the Programmer's Reference Manual.

STORAGE SYSTEM ENTRY TYPES

The basic elements within the Multics storage system are segments and directories. Multics supports additional entry types that are maintained for convenience or to aid programmers who require a storage medium with special qualities or attributes. The various entry types are described below.

Segment

The segment is the unit of storage of the Multics System that is analogous to a file on other systems. A segment is a collection of instructions or data you specify.

Directory

A directory is a catalog of subordinate entries.

Link

A link entry is a reference to an entry in another directory. You make the reference by giving the pathname of the target entry.

Multisegment File

Very large data bases may exceed the size of a single segment. In such cases Multics treats this data base as a group of segments in a single multisegment file. The segments are grouped under a common directory whose multisegment file indicator is set. The directory and its contents are called a multisegment file (MSF).

Any directory whose multisegment file indicator is not 0 is an MSF. For an MSF this indicator is a count of the number of segments it contains. Not all of the attributes listed above are applicable to MSFs. Some of the attributes are the same for any entry; however, due to the nature of an MSF when viewed as a file, many of the attributes are implemented differently. For example, the bit count of an MSF is the sum of the bit counts of the segments it contains. The access control list for an MSF directory applies to all of the segments it contains. You can use the safety switch attribute; however if you set it for one of the segments in the MSF, you should set it for all of them. For more information on these and other attributes of MSFs, see the msf_manager_ subroutine.

Most standard system programs that work on segments also work on MSFs; however some commands and subroutines give unpredictable results when used on MSFs. You should consult the individual command or subroutine description before invoking it on an MSF.

Data Management File

A data management (DM) file is composed of a set of pages known as control intervals, numbered from 0 through N and addressable only through software calls to the file manager. Data is accessed by specifying a control interval number, byte offset, and length.

You can implement DM files with concurrency control and recovery support. At present the ability to use data management files is available only to programs accessing files through the Multics Relational Data Store (MRDS) facility.

Extended Entry Types

The Multics storage system supports special-case entry types called extended entry types. They are so called because the Multics storage system has been enhanced (extended) to treat these storage elements as segments (even though they are structured differently from segments). The following system-supplied storage system elements have been implemented as extended entries: mailboxes, forum meetings, message segments, before journals, and the person name table. Most file system commands (e.g., copy, set_acl, etc.) will operate on extended entries. Each extended entry is identified by a suffix appended to the entry name, as described below:

NAME	SUFFIX
mailbox	.mbx
forum meeting	.forum
message segment	.ms
before journal	.bj
person name table	.pnt

DATE/TIME VALUES

Multics use of date/time values is described in the following subsections. Multics accepts dates from the year 0001 through 9999. The Julian calendar is used for dates from 0001-01-01 through 1582-10-04. The Gregorian calendar is used for dates from 1582-01-15 through 9999-12-31. (The dates from October 5, 1582 through October 14, 1582 do not exist; they were dropped when the Gregorian calendar was adopted.) The leap day is always February 29. The lower limit on dates of January 1, 0001 A.D., was picked since it begins the era; the upper limit of December 31, 9999, was chosen to limit year numbers to four digits. The time zones as now defined are used regardless of the year. The Multics date/time software does not account for "leap seconds", and, therefore, the difference between any two binary clock values that are precisely an integral number of days (hours, minutes, seconds, etc.) apart is guaranteed to be evenly divisible by the number of microseconds in a day (hour, minute, second,etc.).

Date/Time Input Values

Often you must supply date and time information to a command. Programs that accept date and time information use the convert_date_to_binary_ subroutine (see the Subroutines manual) to convert a time string to an internal (binary) value.

TIME STRINGS (DT VALUES)

The time string can have up to six parts: adverbial offset, date, time, day of week, signed offset, and time zone. Adverbial offsets, if present, must appear leftmost in the string. Beyond that, all the parts are optional and can be in any order. The parts can be made up of alphabetic fields, numeric fields, and special characters.

An alphabetic field is made up of letters and must contain a whole word or an abbreviation (often made up of the first three letters of the word). No distinction is made between uppercase and lowercase characters. Although this description gives examples in English, each of the words is available in several languages. You can use any of these languages in time strings, but all words within a given string must be in the same language. To see the languages defined on your site, type

```
display_time_info -lang
```

A numeric field consists of an optionally signed integer of one or more decimal digits. The special characters that you can use in either alphabetic or numeric fields are: the slash (/), the period (.), the colon (:), the plus (+), the minus (-), and the comma (,). Blanks are not required between alphabetic and numeric fields in the time strings; however they are required between two numeric fields unless the second field begins

with a plus or minus sign. For example,

```
2days4hours10minutes
1245.17+7hours
10/17/79Wednesday
```

Unless otherwise indicated in the command description, supply the input time string as a single argument. This means that you must enclose within quotations time strings that contain spaces. Alternatively you can use underscores instead of blanks in the time string. For example,

```
09/25/79__1442.6_+5_hours
```

Usually when you enter a time string, the time zone is omitted. Although the time zone is seldom seen, it is very important: it determines the interpretation of items given in the time string; it is also involved in defaults supplied for missing items. All defaults are taken from the current absolute time, adjusted by a working time zone. If you give a zone in the string, that becomes the working zone; otherwise the process default time zone is used.

This means that whether you convert a string with an explicit zone, such as "XXXX_ast", or set the process default to "ast" and then convert the string "XXXX", you get the same absolute time. (Note that setting the process default also influences output conversion, while giving an explicit zone does not.) To display your default zone, type

```
print_time_defaults zone
```

The six parts of the time string are described below. In these descriptions whenever an assumed value is mentioned, it refers to the current date/time adjusted to the working zone.

1. date
   is the day of the year; you can specify only one date. You can supply a date using normal date format, calendar date format, day of the week, date keywords, fiscal week, request-id, or you can omit it entirely. If no date is present, it is assumed to be the next occurrence of the time specified; for instance, "10A" gives the date on which 10:00am next occurs. If you give no date and time, the current date is used.

   In normal date format, you can specify dates as month (or month abbreviation), day of month, and year; or as day of month, month, and year. The year is optional and, if omitted, is assumed to be the year in which the date occurs next; that is, if today is March 16, 1985, then March 20 is equivalent to March 20, 1985; while March 12 is the same as March 12, 1986. There are three forms of normal date:

   ```
   16 March   16 March 1985
   March 16   March 16 1985   March 16, 1985   (The comma is optional)
   3/16       3/16/85         3/16/1985
   ```

   The calendar date format allows you to supply dates as a year, month, and day of month, separated by minus signs. This is the International Standards Organization (ISO) standard format. The year is required, and you can give it as a year of the century. For example,

```
85-12-31 or 1985-12-31
```

represents December 31, 1985.

The day of the week is a date specifier if present with no other form of date. It then selects the first occurrence of the named day after today.

The date keywords are "yesterday", "today", and "tomorrow"; for instance,

```
6:35A today
yesterday +120days
```

The fiscal week is of the form FWyyyyww. FW is the fiscal indicator (in English), yyyy is the year number, and ww is the week number. The fiscal week begins on Monday and ends on Sunday. This form converts to the date of Monday, but you can select a day within the week by adding a day name; for example, "FW198413 m" gives "03/26/84 0000. Mon", while "FW198413 m Wed" gives "03/28/84 0000. Wed". You can separate the fiscal indicator from the number, but the ordering must remain, i.e., "FW185425" or "FW 185425", but not "185425 FW".

A request-id is a 19-character string used by several programs in the system, such as list_output_request. It contains a complete date from year, in century, down through microseconds in this form

```
yymmddHHMMSS.SSSSSS
```

If you provide no zone, it is interpreted in GMT, not the process default. A request-id specifies a time as well as a date, so you can give no other time specification.

2. day of week
   is a day of the week (e.g., Monday) and can be present only once. When the day of the week is present along with one of the other forms of date specification, that date must fall on the indicated day of the week. You can optionally follow it by a comma.

3. time
   is the time of day and can only be present once. If omitted, it is assumed to be the current time. You can give time as 24-hour format, 12-hour format, or the time keyword "now". The 24-hour time format consists of a four-digit number followed by a period: hhmm., where hh represents hours and mm is minutes. You can follow this number by an optional decimal fraction-of-a-minute field (e.g., hhmm.m). Also acceptable are hours and minutes fields separated by colons (hh:mm). You can optionally follow this by either a fraction-of-a-minute field (hh:mm.m) or a seconds field (hh:mm:ss). The seconds, in turn, can include a fraction-of-second field (e.g., hh:mm:ss.s). Examples of 24-hour time are:

```
1545
1545.715
15:45
15:45.715
15:45:42
15:45:42.08
```

You must end the 12-hour time format with a meridiem designator (i.e., A, P, am, pm, noon (n), midnight (m). You can indicate midnight and noon by giving just the meridiem designator. You can precede the designator by time expressed as hours, hours:minutes, or hours:minutes:seconds (including an optional fraction of a second or fraction of a minute). Examples of 12-hour time are:

```
midnight
5 am
5:45A
3:59:59.000001pm
11:07:30.5pm
12 n
```

There is a set of illegal times--24:00-24:59--which are handled anyway. These are taken to mean 00:00-00:59 of the following day; midnight (00:00) is the beginning of a day, not the end.

4. signed offset
is an adjustment to be made to the clock value specified by the other fields. You can supply offsets in any the following units:

year	years	yr
month	months	mo
week	weeks	wk
day	days	da
hour	hours	hr
minute	minutes	min
second	seconds	sec
microsecond	microseconds	usec

Each unit can be present one or more times, each preceded by an optionally signed fixed point number. If offset fields are the only thing present, the offsets are added to the default values of date and time, as described above.

If the month offset results in a nonexistent date (e.g., "Jan 31 3 months" would yield April 31), the last date of the resulting month is used (i.e., April 30). Examples of offset fields are:

```
3 weeks -60 hours  (60 hours before 3 weeks after now)
1.5 hr 5min  (an hour and 35 minutes from now)
1 hour 5 minutes (an hour and five minutes from now)
```

The order in which offset values are applied to the clock value can affect the resultant clock value. Offset values are applied in the following order:

year, month, week, day, hour, minute, second, microsecond

"Monday 6 am 2 weeks" means "two weeks after the next occurrence of Monday, at 6:00 am on that day".

Assuming that today is September 25, 1985, then

```
10/1 -1 day +1 month
```

results in a clock value for 10/31/85, rather than for 10/30/85.

Note: There is also a nonoffset use of these words, available in combination with the word "this". Some of these combinations can be used in building date and time parts. For example, "this_month_1,_this_year" or "this_hour:23" is valid, while just "this_day" is not. The exact form of this combination varies according to the language used. In some languages the word for "this" changes according to the gender of the unit it is applied to; in others there may be a single word that does the job. To list the word used as "this" for each unit, type

```
display_time_info -offset -language LANGUAGE_NAME
```

5. adverbial offset

is a before/after kind of adjustment that you can use any number of times. You can recognize it by the presence of "before", "on", or "after" in the time string. If present, it must appear first. These are the forms available:

```
DAY-NAME before
DAY-NAME on or before
DAY-NAME before or on
DAY-NAME after
DAY-NAME on or after
DAY-NAME after or on
SIGNED-OFFSETs before
SIGNED-OFFSETs after
```

When adverbial offsets are present, they partition a time string into a series of adjustments followed by a base time. These sections are processed from right to left. The example below has 3 sections: first "6:00 am 400sec" is handled, supplying all necessary defaults and making the ordinary (400sec) offset adjustment; then "Monday after" is applied to give a new value; finally "2 wk -5min after" is applied to this new value to give the final value.

```
2 wk -5min after Monday after 6:00 am 400sec
20 minutes before now
2 days after today
2500 weeks after 1776-7-4
Tue after Mon on or after 11/1
```

The last item describes election day in the USA: the first Tuesday after the first Monday in November.

6. zone

is the time zone to be used in making the conversion to Greenwich mean time, which is the internal form of all clock readings. It can be either a zone differential or any of the zone abbreviations known at your site. A zone differential is a five-character string, "sHHMM" (s is a sign, HH is a two-digit hour, and MM is a two-digit minute). You can use this only immediately following a time specification: "12:15-0330" says that 12:15 is the local time, and -0330 specifies that the local time was generated by subtracting 3.5 hours from GMT. To list the zone abbreviations known at your site, type

```
display_time_info -zones
```

If any defaults are needed, the current instant is broken down into years, months, days, and so forth with respect to a "working zone". This working zone can make much difference because, for example, at a given instant it can be Tuesday in New York and Wednesday in Bankok, or it can be 22:07 in London and 3:37 in Singapore. Thus the zone is as important in applying defaults to week days and years as it is to hours and minutes.

Many of the date/time commands allow you to supply a "-zone X" argument. In this case, X can be any of the zones known at you site; it can't be a time differential.

Date/Time Output Values

One way to get a clock value into a readable form is by using the date/time commands (calendar_clock, day, etc). The first argument to the clock command is a control string describing the format wanted. All other date/time commands have intrinsic formats. These commands convert a readable time string to an internal value and then convert this internal clock reading to the specified output time format.

An input time string is converted to internal form by convert_date_to_binary_. This is the usual form for storing dates in data bases. To convert an internal clock reading into a readable form, you can call date_time_ to get a 24-character form like this:

```
03/14/79 0000.0 cet Fri
```

But when other formats are needed, date_time_$format is available. It takes a clock value and a control string describing the format wanted and returns a string ready for printing.

An effort has been made to make all date/time outputs from the system software usable as date/time inputs to system software, but the time format mechanism is so flexible that you can easily use it to generate formats that are not recognizable. Also some strings are apparently recognized, even though they are ambiguous. For instance, "7/1/82" means the 7th month, first day in the United States, but in many European countries would mean the 7th day of the first month. Multics follows the American interpretation.

TIME FORMAT

The control string for the date_time_$format subroutine, clock command, and other commands that expect a time_format argument is either a keyword or a character string consisting of text and/or selectors. The selectors are always identified by a leading circumflex character (^). There are two types of selectors: ^<keyword>, which allows a keyword to be embedded within a format, and the general form ^XX. XX is a two-letter code that specifies what information is wanted. You can place an optional PL/I picture specification between the ^ and XX if the default form is not adequate. If the control string does not contain any circumflex characters, it must then be one of the known set of keywords. Each keyword identifies a control string for a predetermined format named by that keyword.

all
^9999yc-^my-^dm__^Hd:^MH:^99.(6)9UM^zd_^za_^da ^fi ^(6)9fw ^ma dy^dy dc^dc Uc^Uc.

calendar_clock
^9999yc-^my-^dm__^Hd:^MH:^99.(6)9UM_^za_^da.

clock
^9999yc-^my-^dm ^Hd:^MH:^99.(6)9UM ^za ^da.

date
is the process default value for date.

date_time
is the process default value for date and time.

iso_date
^9999yc-^my-^dm.

iso_date_time
^9999yc-^my-^dm ^Hd:^MH:^SM ^za.

iso_long_date
^9999yc-^my-^dm ^da.

iso_long_date_time
^9999yc-^my-^dm ^Hd:^MH:^99.(6)9UM ^za.

iso_long_time
^Hd:^MH:^99.(6)9UM.

iso_time
^Hd:^MH:^SM.

multics_date
^my/^dm/^yc.

multics_date_time
^my/^dm/^yc ^Hd^99v.9MH ^xxxxza^xxxda.

multics_time
^Hd:^MH.

request_id
^yc^my^dm^Hd^MH^99.(6)9UM. The output from this keyword is specified in the process default time zone; therefore if you want a valid request-id, specify -zone GMT in commands or give GMT as the zone argument when calling date_time_$format with the request_id keyword (see "Request IDs" in Section 3 of the Programmer's Reference Manual).

system_date_time
is the system default value for date and time.

system_date
    is the system default value for date.

system_time
    is the system default value for time.

time
    is the process default value for time.

Your site can change the "system" strings. For an application that depends upon the historic formats the three builtin "multics" strings are available.

Processing of a control string proceeds by scanning the control string until a circumflex is found or the end of the string is reached. Any text (including any blanks) passed over is copied to the output string. The selector is then interpreted and executed. This causes a datum from the input clock value to be edited into the output string. Processing continues in this way until the control string is exhausted.

You can express dates and times placed in the output string in units of years, months, weeks, days, hours, minutes, seconds, and microseconds, and the total calendar value as a single unit; for example, you could express the calendar value representing 79-09-08 9:42A GMT as 1979 years, as 722702 days, or as 722702.112499 days. This is the set of "total" selectors:

```
^yc   total number of years in the calendar value
^mc   total number of months in the calendar value
^dc   total number of days in the calendar value
^Hc   total number of hours in the calendar value
^Mc   total number of minutes in the calendar value
^Sc   total number of seconds in the calendar value
^Uc   total number of microseconds in the calendar value.
```

You can also express dates and times as the number of units remaining after a larger unit has been removed from the calendar value; for example, 09/08/79 09:42 includes units for the 9th month of the year, the 8th day of the month, the 9th hour of the day, and the 42nd minute of the hour. The following are the most common:

```
^my   month in the year
^dm   day of the month
^dw   day of the week
^Hd   hour of the day (24-hour format)
^Hh   hour in half day (12-hour format)
^MH   minute of the hour
^SM   second of the minute
^US   microsecond of the second.
```

There are several items of date/time data that are nonnumeric, such as day of week, day of month, and time zone used for conversion.

```
^mn   month name
^ma   month name, abbreviated (char (3))
^dn   day name
^da   day name, abbreviated (char (3))
^zn   time zone name
^za   time zone name, abbreviated (char (4))
^zd   zone differential (char(5))
^mi   meridiem indicator (A or P)
^fi   fiscal indicator (FW in English)
```

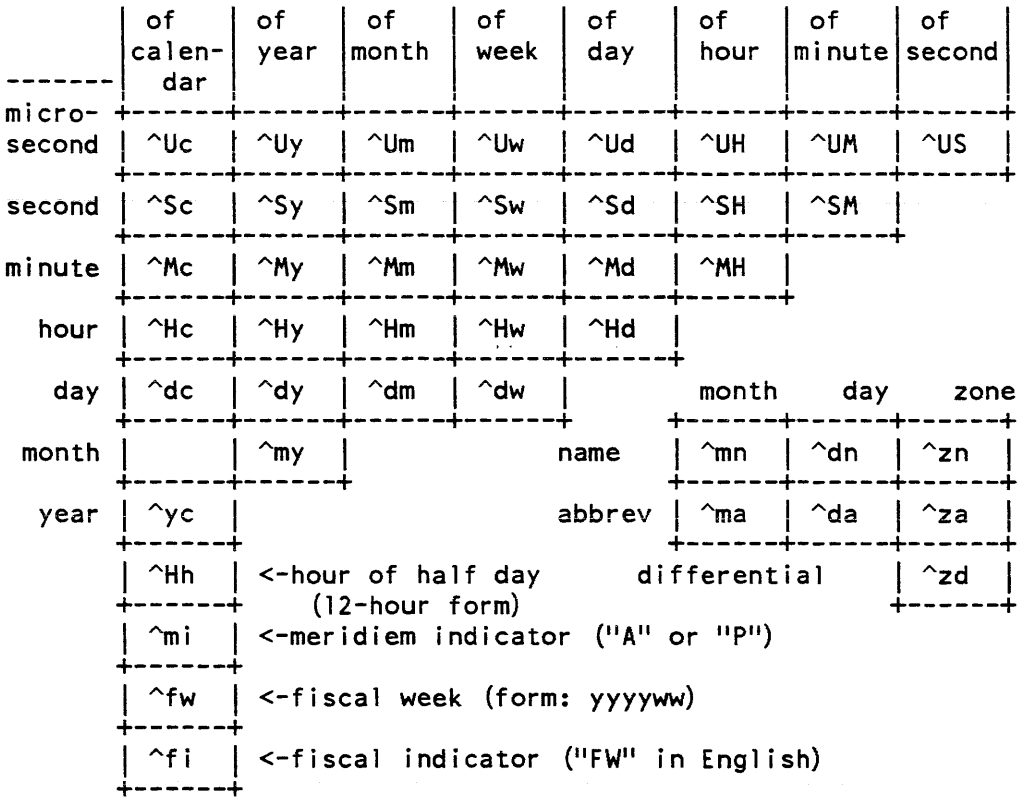
The selectors of numeric data are, in general, made up of two letters taken from this sequence: c y m w d H M S U. These letters stand for calendar, year, month, week, day, hour, minute, second, and microsecond, respectively. All 81 combinations are not, however, valid. The form can generally be read as "unit of unit", e.g., "seconds of week". The first unit is always smaller than the second one. In trying to keep the specifiers reasonably mnemonic (in English) there is a problem: both month and minute begin with an "m". So all date values are used as lowercase letters while all time values are in uppercase.

It is difficult to try to handle all the forms needed in a general manner. Hd is hour of the day and is thus 24-hour time; this is not always what is wanted. Hh is chosen as hour in half day to get the 12-hour form of time. To go along with this there is "mi" for Meridiem Indicator, which gives A or P to make up AM or PM. This does not give AM or PM because ANSI and ISO standards specify that time be given as "3P", not "3PM". If you want the M, put the literal in, e.g., "^miM".

Another way of looking at a calendar value is in terms of fiscal week. This is selected with the "^fw" code. Its value is four digits of year followed by two digits of week number, i.e., yyyyww. The default picture has been chosen to give a value of yww. The associated fiscal indicator is selected by "^fi". A complete value is obtained by specifying "^fi^fw".

The table below shows the complete set of selectors. The row specifies what unit is wanted, the column specifies within what other unit, e.g., ^Sy is seconds of year.

DATE/TIME SELECTORS



You can control the formatting of date and time values by an optional PL/I picture specification included in the selector; for instance, a code of ^OO99yc formats the total years in the calendar value into a two-digit year of the 20th century and ^9999yc provides a full, four-digit year. The following is a brief description of the most frequently used picture characters. For more details on PL/I pictures, see the Multics PL/I Language Specification manual (AG94) and the Multics PL/I Reference Manual (AM83).

9 represents a mandatory decimal digit in the displayed value.

z represents a decimal digit in the displayed value. Nonsignificant zeros on the left are replaced by a space when they occupy a "z" digit position.

. produces a period in the displayed value. This has no relation to the location of the decimal point in the value actually being displayed. If zero suppression is in effect, this is replaced with a space.

, produces a comma in the displayed value. It has all the characteristics of the period.

v locates the value's decimal point in the result. This determines how the value digits are oriented with respect to the picture specification. If you supply no "v", it is assumed to appear after the rightmost picture character.

The picture characters above are sufficient for displaying most numeric values. For example, the control string ^99Hd^99.v9MH represents the time in hours, minutes, and tenth of minutes; the control string ^zz9.999vUS represents the number of milliseconds of the second, using the decimal point and "v" to scale the microsecond unit. Scaling can also be performed by a picture scale factor.

f(N) scales the value by multiplying or dividing by a power of 10, thus shifting the location of the decimal point in the value. For example, f(2) shifts the decimal two places left, effectively dividing the value by 100; f(-3) shifts three places right, effectively multiplying by 1000.

Using a picture scale factor, you can display the milliseconds in excess of a second to the nearest tenth using the control string ^zz9.9f(3)US. You can display a value of 48634 microseconds as " 48.6" milliseconds.

There are two extensions to numeric picture handling that you can use in time format selectors:

Z represents a decimal digit in the displayed value. Nonsignificant zeros to the left of the decimal point are omitted when they occupy a "Z" digit position; to the right of the decimal point they are omitted when they occupy a "Z" digit position.

Z characters must appear as the leftmost or rightmost digit positions in the picture specification since these are the positions that nonsignificant zeros can occupy. Z performs a selective ltrim or rtrim (of zero) operation on the displayed value. For example, you can specify the millisecond specification given above as ^ZZ9.9ZZUS without using a picture scale factor; with this specification you can display 48630 microseconds as 48.63 milliseconds (without the leading space or trailing zero).

O represents a decimal digit in the displayed value that should be omitted. Specifying ^99yc for a year like 1941 results in a size condition since it takes four digits to handle that number. To get the year in century you can use ^OO99yc; this gives four digits into which the value is placed and then the first two digits are discarded. A picture like OOz9 with a value of 1502 gives 02 because the zero suppression applies to 1502, and then the first two digits are dropped.

You can format character date/time values such as day of the week, month name, and time zone using a character picture specification with the "x" picture character.

x represents a position that can contain any character. Since national characters occur in some of the time names, avoid use of the "a" picture character. Values are left-justified in the picture specification, with truncation of the rightmost characters if the value is longer than the picture or padding with spaces on the right if the value is shorter than the picture.

For example, ^xxxxxxxxdn displays Wednesday as "Wednesday" and Monday as "Monday ". You can use a picture repetition factor to shorten the control string to "^(9)xdw". With ^(5)xmn January is displayed as "Janua" and May is displayed as "May ". (Note that in some languages the abbreviation of a time name is not the first three letters of it.)

The selector picture specification allows an extension of the "x" picture specification.

X represents an optional character position in the displayed value. The character position is omitted if there is no corresponding character in the value being displayed.

X characters must appear as the rightmost character position in the picture specification since this is the position that nonsignificant spaces can occupy. X performs a selective rtrim operation on the displayed value.

The code ^(9)Xdw displays Wednesday and Monday both without trailing spaces.

The table below shows the default picture specifications for all selectors. The row specifies what unit is wanted, the column specifies within what other unit.

DEFAULT PICTURE VALUES

	of calendar	of year	of month	of week	of day	of hour	of minute	of second
microsecond	(18)Z9	(14)Z9	(13)Z9	(12)Z9	(11)Z9	(10)Z9	(8)Z9	(5)Z9
second	(12)Z9	(12)Z9	(8)Z9	(6)Z9	(5)Z9	(4)Z9	99	
minute	(10)Z9	(6)Z9	(5)Z9	(5)Z9	(4)Z9	99		
hour	(8)Z9	(4)Z9	(3)Z9	(3)Z9	99			
day	(7)Z9	999	99	9		month	day	zone
month		99			name	(32)X	(32)X	(64)X
year	0099				abbrev	(8)X	(8)X	(8)X

99	<-hour of half day (12-hour form)	differential s9999
x	<-meridiem indicator	
000999	<-fiscal week (form: yyyyww)	
xx	<-fiscal indicator	

The following table shows how date and times strings are displayed by a variety of control strings.

^mn ^Z9dm, ^9999yc
    displays September 8, 1979.

^mn ^z9dm, ^9999yc
    displays September 8, 1979.

^dm ^ma ^9999yc ^zn
    displays 08 Sep 1979 Mountain Standard Time.

^my/^dm/^yc ^Hd^99v.9MH ^za ^da
    displays 09/08/79 0242.4 mst Sat.

^Hd:^MH:^SM^zd
    displays 02:42:25-0700.

^9999yc-^my-^dm__^Hd:^MH:^99.(6)9UM_^za_^da
    displays "1979-09-08__02:42:25.048634_mst_Sat.

<-^<multics_time>xyz^<multics_date>->
    displays <-02:42xyz09/08/79->.

SECTION 2

REFERENCE TO COMMANDS AND ACTIVE FUNCTIONS

The Multics commands and active functions are presented in this section by functional use.

ACCESSING THE MULTICS SYSTEM

access_class
dial
echo
enter
hangup
hello
help, HELP
login
logout
MAP
modes
noecho
slave
terminal_id
terminal_type

COMMAND LINE PROCESSING

abbrev
answer
convert_ec
default
do
do_subtree
exec_com
execute_string
if
login_args
on
pause
program_interrupt
progress
query
release
repeat_query
response
run
select
set_epilogue_command
severity
start
stop_run
substitute_arguments
walk_subtree

PROCESS ENVIRONMENT

change_error_mode
exponent_control
general_ready
home_dir
logout
new_proc
no_save_on_disconnect
print_auth_names
print_proc_auth
process_dir
process_switch_off
process_switch_on
ready
ready_off
ready_on
reconnect_ec_disable
reconnect_ec_enable
reprint_error
save_on_disconnect
set_tty
system
user

STORAGE SYSTEM NAMES

add_name
branches
component
compare_entry_names
copy_names
default_wdir
delete_name
directories
directory
entries
entry
entry_path
equal_name
files
get_pathname
home_dir
is_component_pathname
links
list
list_subtree
list_ref_names
list_temp_segments
master_directories
move_names
msfs
nonbranches
nondirectories
nonfiles
nonlinks
nonmaster_directories
nonmsfs
nonnull_links
nonobject_files
nonobject_msfs
nonobject_segments
nonsegments
nonzero_files
nonzero_msfs
nonzero_segments
null_links
object_files
object_msfs
object_segments
path
process_dir
rename
segments
shortest_path
strip
strip_component
strip_entry
suffix
working_dir
zero_segments

CREATING AND EDITING SEGMENTS

adjust_bit_count
canonicalize
convert_characters
convert_ec
copy
create
delete
edm
emacs
expand_cobol_source
file_output
format_pl1
indent
merge_ascii
qedx
set_bit_count
sort_seg
teco
teco_error
teco_ssd

SEGMENT ATTRIBUTES

add_name
adjust_bit_count
check_file_system_damage
copy_acl
copy_names
delete_acl
delete_name
describe_entry_type
get_effective_access
list_acl
list_entry_types
list_temp_segments
rename
set_acl
set_bit_count
set_max_length
set_ring_brackets
status
switch_off
switch_on
truncate
vfile_adjust
vfile_status

SEGMENT MANIPULATION

archive
archive_sort
canonicalize
compare
compare_ascii
compare_pl1
contents
copy
create
decode
delete
dump_segment
encode
initiate
linkage_editor
mbx_create
merge_ascii
move
overlay
print
reorder_archive
sort_seg
tape_archive
terminate
terminate_refname
terminate_segno
terminate_single_refname
truncate

DATA MANAGEMENT FILE MANIPULATION

before_journal_status
bj_mgr_call
create_dm_file
dm_display_version
dm_user_shutdown
transaction

DIRECTORY ATTRIBUTES

add_name
check_file_system_damage
copy_acl
copy_iacl_dir
copy_iacl_seg
copy_names
delete_acl
delete_iacl_dir
delete_iacl_seg
delete_name
get_dir_quota
get_effective_access
get_quota
list_acl
list_iacl_dir
list_iacl_seg
move_dir_quota
move_quota
rename
set_acl
set_dir_ring_brackets
set_iacl_dir
set_iacl_seg
set_mdir_account
status
switch_off
switch_on

DIRECTORY MANIPULATION

comp_dir_info
copy_dir
create
create_dir
date_deleter
delete
delete_dir
directories
do_subtree
link
linkage_editor
list
list_dir_info
list_sub_tree
move_dir
rebuild_dir
save_dir_info
unlink
walk_subtree

EXTENDED ENTRY TYPES

add_name
copy
copy_names
delete
delete_acl
delete_name
describe_entry_type
entries
exists
list_acl
list_entry_types
move
move_names
rename
set_acl
set_bit_count
set_max_length
set_ring_brackets
status
switch_off
switch_on

LINKS AND SEARCH FACILITIES

add_search_paths
add_search_rules
change_wdir
change_default_wdir
default_wdir
delete_search_paths
delete_search_rules
get_system_search_rules
hunt
hunt_dec
initiate
list_ref_names
print_default_wdir
print_search_paths
print_search_rules
print_wdir
resolve_linkage_error
set_search_paths
set_search_rules
terminate
where
where_search_paths
working_dir

ACCESS CONTROL AND RINGS OF PROTECTION

check_iacl
copy_acl
copy_iacl_dir
copy_iacl_seg
delete_acl
delete_iacl_dir
delete_iacl_seg
get_effective_access
list_accessible
list_acl
list_not_accessible
list_iacl_dir
list_iacl_seg
print_auth_names *
print_proc_auth
set_acl
set_iacl_dir
set_iacl_seg |
set_dir_ring_brackets
set_ring_brackets

STORAGE SYSTEM, LOGICAL VOLUMES

attach_lv
delete_volume_quota
detach_lv
list_mdir
lv_attached
set_mdir_account |
set_mdir_owner |
set_mdir_quota
set_volume_quota |

STORAGE SYSTEM BACKUP AND RETRIEVAL

cancel_retrieval_request
compare_dump_tape
copy_dump_tape *
enter_retrieval_request
list_retrieval_requests

ONLINE INFORMATION

check_info_segs
explain_doc
help
how_many_users
list_help
print_motd
tutorial
validate_info_seg
where_doc
who

MENU AND VIDEO SYSTEM

menu_create
menu_delete
menu_describe
menu_display
menu_get_choice
menu_list
window_call

INTERUSER COMMUNICATION

accept_messages
accepting
defer_messages
delete_message
display_mailing_address
have_mail
have_messages
immediate_messages
last_message
last_message_destination
last_message_sender
last_message_time
mail
message_status
mbx_create
print_mail
print_messages
read_mail
send_mail
send_message
set_mailing_address
who

INPUT/OUTPUT SYSTEM CONTROL

attach_audit
cancel_daemon_request
cancel_output_request
close_file
connect
copy_cards
copy_file
detach_audit
dial_manager_call
dial_out
discard_output
display_audit_file
dprint
dpunch
enter_output_request
file_output
get_mode
have_queue_entries
io_call
kermit
l6_ftf
line_length
list_daemon_requests
list_emacs_ctls
list_output_requests
micro_transfer
move_daemon_request
move_output_request
network_request
print
print_attach_table
print_request_types
print_terminal_types
repeat_line
set_tty
tape_archive
tape_in
tape_out
total_output_requests
vfile_adjust
vfile_find_bad_nodes
vfile_status
window_call

FORMATTED OUTPUT FACILITIES

cancel_daemon_request
cancel_output_request
dprint
dpunch
enter_output_request
format_document
format_line
format_line_nnl
format_pl1
format_string
have_queue_entries
indent
list_daemon_requests
list_output_requests
move_daemon_request
move_output_request
overlay
picture
print
print_request_types
runoff
runoff_abs
set_cc
sort_strings
total_output_requests

TERMINAL INTERFACE PROGRAMS

cv_ttf
connect
dial_out
display_ttt
get_mode
kermit
line_length
l6_ftf
micro_transfer
network_request
print_terminal_types
print_ttt_path
set_ttt_path
set_tty
window_call

CONTROL OF ABSENTEE COMPUTATIONS

cancel_abs_request
cobol_abs
enter_abs_request
fortran_abs
list_abs_requests
move_abs_request
pl1_abs
runoff_abs

PROGRAMMING LANGUAGES (COMPILERS)

alm
alm_abs
apl
basic
cobol
cobol_abs
create_data_segment
fortran
fortran_abs
list_fortran_storage
pascal
pascal_area_status
pascal_cross_reference
pascal_create_area
pascal_delete_area
pascal_display
pascal_file_status
pascal_indent
pascal_reset_area
pascal_set_prompt
pl1
pl1_abs
pl1_macro
reductions

PROGRAMMING/DEBUGGING AIDS

add_pnotice
cancel_cobol_program
close_file
create_data_segment
cumulative_page_trace
debug
delete_external_variables
display_cobol_run_unit
display_entry_point_dcl
display_pl1io_error
display_pnotice
expand_cobol_source
exponent_control
fast
format_pl1
history_comment
indent
io_call
list_external_variables
list_fortran_storage
list_pnotice_names
nothing
print_bind_map
print_error_message
print_link_info
print_linkage_usage
print_sample_refs
probe
process_switch_off
process_switch_on
profile
progress
reset_external_variables
run
run_cobol
sample_refs
save_history_registers
set_fortran_common
set_cc
set_severity_indicator
severity
signal
stop_run
stop_cobol_run
trace
trace_meters
trace_stack
valid_pictured_data
watch

OBJECT SEGMENT MANIPULATION

archive
archive_table
bind
compare_object
cross_reference
date_compiled
display_component_name
hunt_dec
linkage_editor
print_bind_map
print_link_info
print_relocation_info
reorder_archive

AREA MANAGEMENT

area_status
create_area
set_system_storage
set_user_storage

PERFORMANCE MONITORING FACILITIES

cumulative_page_trace
page_trace
print_linkage_usage
profile
progress
trace
trace_meters
watch

SYSTEM LIBRARIES

add_pnotice
cross_reference
describe_psp
display_pnotice
generate_pnotice
get_library_segment
library_descriptor
library_fetch
list_pnotice_names
peruse_crossref

ARCHIVE SEGMENT MANIPULATION

archive
archive_sort
archive_table
bind
compare_ascii
component
get_library_segment
is_component_pathname
library_fetch
linkage_editor
merge_ascii
path
print
print_link_info
reorder_archive
strip_component
test_archive

SYSTEM MAINTENANCE AND DEBUGGING TOOLS

get_ips_mask
monitor_quota
reset_external_variables
reset_ips_mask
set_ips_mask

RESOURCE CONTROL PACKAGE

attach_lv
acquire_resource
assign_resource
cancel_resource
detach_lv
list_resources
list_resource_types
lv_attached
release_resource
reserve_resource
resource_status
set_resource
unassign_resource

TAPE MAINTENANCE UTILITIES

list_tape_contents
manage_volume_pool
read_tape_and_query
mtape_delete_defaults
mtape_get_defaults
mtape_set_defaults
tape_archive
tape_in
tape_out

CONDITION HANDLING

change_error_mode
display_pl1io_error
on
reprint_error
signal

SETTING AND STORING VARIABLES

delete_external_variables
list_external_variables
reset_external_variables
value_defined
value_delete
value_get
value_list
value_path
value_set
value_set_path

ADMINISTRATIVE UTILITIES

check_file_system_damage
compare_configuration_deck
delete_volume_quota
list_mdir
monitor_quota
print_configuration_deck
set_mdir_account
set_mdir_owner
set_volume_quota

ARITHMETIC OPERATIONS

calc
ceil
divide
floor
index_set
max
minus
min
mod
plus
quotient
times
trunc

LOGICAL OPERATIONS

and
equal
exists
greater
if
less
nequal
ngreater
nless
not
or
select

CONVERSION OPERATIONS

binary
convert_characters
convert_ec
cv_ttf
decimal
dump_segment
hexadecimal
octal

CHARACTER STRING OPERATIONS

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

DATE AND TIMES

calendar
calendar_clock
clock
date
date_time
date_time_after
date_time_before
date_time_equal
date_time_interval
date_time_valid
day
day_name
display_time_info
hour
long_date
long_year
memo
minute
month
month_name
print_time_defaults
set_time_default
time
year

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 {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ab]
```

FUNCTION

provides a mechanism for abbreviating parts of or whole command lines in the Multics command environment. As an active function, returns "true" if abbreviation expansion of command lines is currently enabled, "false" otherwise.

CONTROL ARGUMENTS

-escape STR, -esc STR
: changes the abbrev escape character used to indicate that a command line is actually a request line. STR must be a single, nonblank character. (See "Notes on Control Requests" and the .escape control request.) (Default: a period [.])

-off
: disables abbreviation expansion in subsequent command lines (see the .quit request).

-on
: enables abbreviation expansion within subsequent command lines until you use either -off or .quit. (Default)

-profile path, -pf path
: changes the pathname of the profile segment. The "profile" suffix is assumed if you don't supply it. If the specified segment is nonexistent, you are asked for permission to create it. (See the .use request.) (Default: >udd>Project_id>Person_id>Person_id.profile) cbf

NOTES

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

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 .s and .<space>, is neither checked for embedded abbreviations nor (even in part) passed on to the command processor. If the command line is not an abbrev request line, abbrev expands it and passes it on to the current command processor.

LIST OF CONTROL REQUESTS

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

.

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

.<space> <rest of line>

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

.a <abbr> <rest of line>

adds the abbreviation <abbr> to the current profile segment. It is an abbreviation for <rest of line>. The <rest of line> string can contain any characters. If the abbreviation already exists, you are asked whether to redefine it. You 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>

adds 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.

.abf <abbr> <rest of line>

adds an abbreviation that is expanded only at the beginning of a line and forces it to replace any previous one with the same name. You are not asked whether to redefine it.

.af <abbr> <rest of line>

adds an abbreviation to the profile segment and forces it to overwrite any previous one with the same name. You are not asked whether to redefine it.

.d <abbr1>...<abbrN>

deletes the specified abbreviations from the current profile segment.

.f

enters a mode (the default) that forgets each command line after executing it (see .r and .s).

.l <abbr1>...<abbrN>

lists the specified abbreviations and the strings they stand for. If none are given, all abbreviations in the current profile segment are listed.

.a NAME LINE
adds the abbreviation NAME to the current profile segment. It is an abbreviation for LINE. The LINE string can contain any characters except break sequences. (See "Notes on Break Sequences.") If the abbreviation already exists, you are asked whether to redefine it; respond with "yes" or "no."

.ab NAME LINE
adds an abbreviation that is expanded only if found at the beginning of a line or after a semicolon (;), semicolon vertical bar pair (;|), or left bracket ([) in the expanded line. In other words, this is an abbreviation for a command name.

.abf NAME LINE
adds an abbreviation that is expanded only at the beginning of a line and forces it to replace any previous one with the same name. You are not asked whether to redefine it.

.af NAME LINE
adds an abbreviation to the profile segment and forces it to overwrite any previous one with the same name. You are not asked whether to redefine it.

.debug
invokes debug to debug a process in which it is no longer possible to execute commands although it is still possible to execute abbrev request lines.

.delete NAMEs, .dl NAMEs, .d NAMEs
deletes the specified abbreviations from the current profile.

.edit NAME
invokes Qedx to edit the definition of the specified abbreviation (see "Notes on Editing Abbreviations").

.escape {STR}, .esc {STR}
changes the escape character used to indicate that a command line is actually a request line. STR must be a single, nonblank character. If you give no STR, the escape character presently in use is displayed. (Default: a period [.])

.forget, .f
disables .remember; i.e., it forgets each command line after executing it (see .remember and .show). (Default)

.l {NAMEs}
displays the names, switches, and definitions of the specified abbreviations in alphabetical order. If you give no names, all abbreviations in the profile are listed.

.la STRs
displays the names, switches, and definitions of any abbreviations whose names start with one of the given strings. Supply at least one string.

.lab, la^b STRs
displays the names, switches, and definitions of abbreviations which are beginning-of-line abbreviations (lab) or not beginning-of-line abbreviations (la^b), starting with STRs.

.lb, l^b {names}
displays the names, switches, and definitions of the given abbreviations; lb for beginning-of-line, l^b for not beginning of line abbreviations. If no names are given, lists all of the abbreviation-type.

.ls STRs
displays the names, switches, and definitions of any abbreviation which contain STRs in its name.

.lsb, ls^b STRs
displays the names, switches, and definitions of any beginning-of-line abbreviations (lsb) or not beginning-of-line abbreviations whose name contains STRs.

.lx STRs
displays the names, switches and definitions of abbreviations whose definitions contain STRs.

.lxb, lx^b STRs
displays the names, switches and definitions of beginning-of-line abbreviations (lxb) or not beginning-of-line abbreviations (lx^b) whose definitions contain STRs.

parentheses	()
apostrophe	'
period	.
semicolon	;
less than	<
greater than	>
brackets	[]
braces	{}
vertical bar	|

The two-character-sequence archive component pathname delimiter (::) is also recognized as a break sequence.

EXAMPLES

Suppose that you wish to abbreviate the pathname of a directory in which you do a lot of work. Instead of having to type the entire pathname every time you need 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, you can change to that directory.

```
! cwd myinfo
```

Change to the inferior directory called data_dir.

```
! cwd myinfo>data_dir
```

Another useful abbreviation is for the enter_output_request command, when you frequently use a certain printer queue and a special request type. The do command is used to substitute arguments into the abbrev. For example:

```
! .ab printx do "eor &1 -q 2 -rqt x1200 -nt -he "By George""
```

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

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

With the do command you can also perform a series of functions that are defined by one simple abbrev; for example:

```
!  .ab send_cp do "sms Lincoln.States A copy of &1 that I've prepared
   this week is being printed for you.; printx -dl -he Lincoln &1"
```

NOTES ON BREAK SEQUENCES

When abbrev expands a command line, it treats certain character sequences as special break sequences. An abbreviation cannot contain break sequences. Any character string up to eight characters long and bounded by break sequences can be expanded. The string is looked up in the current profile segment and, if found, the expanded form is placed in a copy of the command line to be passed on to the normal command processor. The following single-character break sequences are recognized by abbrev:

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

The beginning and end of the line and the two-character-sequence archive component pathname delimiter (::) are also break sequences.

LIST OF ABBREVIATION DEFINITION SWITCHES

The following switch is part of the definition of each abbreviation:

beginning_of_line, bol
: specifies that this abbreviation is only expanded in a command when appearing at the beginning of a line or immediately after the semicolon (;), semi-colon vertical bar pair (;|) or left bracket ([). (I.e., when the abbreviation is used as the command name). If this switch is off, the expansion occurs anywhere on a command line.

NOTES ON EDITING ABBREVIATIONS

When you invoke the edit request to edit an abbreviation, it first displays the definition of the abbreviation and then invokes Qedx with the definition in buffer 0.

Using the Qedx write request without a pathname saves the revised definition in the profile segment. Using the read or write request with a pathname, in any buffer, makes the pathname be interpreted as the name of an abbreviation. Presently, you can't read a buffer from, or write it to, a segment.

When writing a buffer and an abbreviation of the given name does not exist, it is created with the bol switch set off. If the abbreviation already exists and is not the default for the buffer as displayed by the Qedx status request, abbrev asks for permission to overwrite the definition of the abbreviation. In this case, the abbreviation retains its original setting of the bol switch.

EXAMPLES

Suppose that you wish to abbreviate the pathname of a directory in which you do a lot of work. Instead of having to type the entire pathname every time you need to reference it, you can use fewer keystrokes, as in the following examples:

Invoke the abbrev command:

```
! ab
```

Define the abbreviation

```
! .a Opinfo >user_dir_dir>Antarctica>Opus>Opus.profile
```

Now that "Opinfo" is defined, you can change to that directory.

```
! cwd Opinfo
```

Change to the inferior directory called data_dir.

```
! cwd Opinfo>data_dir
```

When you frequently use a certain printer queue and a special request type, an abbreviation for the enter_output_request command is

```
! .ab printx do "eor &1 -q 2 -rqt x1200 -nt -he "By Jove""
```

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

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

With the do command you can also perform a series of functions that are defined by one simple abbrev; for example,

```
!  .ab send_cp do "sms Opus.Antarctica A copy of &1 that I've prepared
   this week is being printed for you.; printx -dl -he Opus &1"
```

Then the following send a message and a copy of data.list to Opus.

```
send_cp data.list
```

An abbreviation can invoke other abbreviations, as seen above. If you want to ensure, within the do's command line, that a string not be expanded, enclose it in an extra layer of quotes; for example,

```
.ab eor do """"eor"""" &1 -rqt x1200 -q 3"
```

Name: accept_messages, am

SYNTAX AS A COMMAND

```
am {mbx_specification} {-control_args}
```

FUNCTION

initializes or reinitializes your process both for accepting messages that are sent by send_message and for notifications.

ARGUMENTS

mbx_specification
: specifies the mailbox on which messages are to be accepted. If not given, the user's default mailbox (>udd>Project>Person>Person.mbx) is used.

LIST OF MBX SPECIFICATIONS

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

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

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

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

-user STR
: specifies either a user's default mailbox or an entry in the system mail table.

STR
: is any noncontrol argument, and is first interpreted as -mailbox STR; if no mailbox is found, STR is then interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

CONTROL ARGUMENTS

-brief, -bf
prevents accept_messages from informing you that it is creating a mailbox, and prints messages in short format.

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

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

where:

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

number
is the sequence number of the message, assigned when you use -hold_messages; otherwise it 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 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 suppress a previous -call, give -call with no cmdline argument.

-flush DT
discards messages sent before the specified date-time (see Section 1 for a description of valid DT values). This control argument should be used by operators and consultants.

-hold_messages, -hdmsg
holds messages until explicitly deleted by delete_message. Messages printed when -hold_messages is in effect are preceded by an identifying number.

-hold_notifications, -hdnt
holds notifications in the mailbox after being printed. This implies -notifications.

-long, -lg
precedes every message printed by the sender's Person_id and Project_id and prints the date-time string. It prints the message number only if you use -hold_messages. (Default)

-no_hold_messages, -nhdmsg
reverts -hold_messages.

-no_hold_notifications, -nhdnt
deletes notifications after being printed. (Default)

-no_notifications, -nnt
deletes notifications as they are received. This implies -no_hold_notifications.

-no_print, -npr
does not print old messages. (Default)

-no_short_prefix, -nshpfx
does not print the prefix when messages are printed in short format.

-notifications, -nt
prints notifications. (Default)

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

-print, -pr
prints all messages that you received since the last time you were accepting messages. The messages are deleted after printing, unless you are holding them.

-short, -sh
precedes consecutive messages from the same sender by "=" instead of the Person_id and Project_id, and prints the date-time string only if less than five minutes have passed since the previous message. It omits the date if the current message and the previous one are received on the same date.

-short_prefix, -shpfx
prints the prefix when messages are printed in short format. (Default)

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

```
You have X messages
```

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

NOTES

A default mailbox is created the first time you issue print_mail, read_mail, or accept_messages. The default mailbox is

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

Messages sent when you are not logged in or when you are deferring messages (see defer_messages) are saved in the mailbox; you can read them later with print_messages. The send_mail command stores mail in the same mailbox.

Don't share the same mailbox with others.

At any time, only one process can be accepting messages from a given mailbox. If you create two processes that accept messages from the same mailbox, the second process (i.e., the one issuing an accept_messages most recently) automatically take over the command function. The first process receives 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 if you send a message to that mailbox, you are informed that the addressee is currently not accepting messages or is not logged in. So if you are registered on multiple projects using a common mailbox, be aware that this behavior affects your processes.

Generally don't accept messages in absentee processes; the start_up.ec should distinguish between interactive and absentee processes, and should issue accept_messages only in an interactive process.

You can accept messages on more than one mailbox at a time and on a mailbox other than the default. If you use a nondefault mailbox and it does not exist, accept_messages queries you whether it should be created. When messages are printed from a nondefault mailbox, the mailbox is always identified.

Name: accepting

SYNTAX AS A COMMAND

```
accepting address
```

SYNTAX AS AN ACTIVE FUNCTION

```
[accepting address]
```

FUNCTION

determines whether messages are being accepted on the mailbox specified by the address supplied.

ARGUMENTS

address
: can be of the form Person_id.Project_id to specify a mailbox belonging to that person; a string containing at least one > or < to specify the pathname of a mailbox; one of the arguments -mailbox (-mbx), -log, or -save (-sv), immediately followed by a string giving the pathname of a mailbox, logbox, or save box, respectively; -last_message_destination (-lmds) if you have used send_message in this process; or -last_message_sender (-lms) if a message has been received in the user's default mailbox.

Name: acquire_resource, aqr

SYNTAX AS A COMMAND

```
aqr type STR1 {...STRs} {-control_args}

aqr type -number N {-control_args}
```

FUNCTION

selects a resource of a given type from a free pool of all such resources and makes you the accounting owner of the resource. You are given full control over the access rights for all users of the resource, as well as control over many parameters of the resource. Ownership of the resource is terminated by release_resource.

ARGUMENTS

type
: is a resource type defined in the resource type description table (RTDT).

STRs
is the unique identifying name of the particular resource being acquired. If STR looks like a control argument, precede it by -name (-nm). If you give no -name, a resource is chosen to satisfy any constraints imposed by the control arguments given.

CONTROL ARGUMENTS

-access_class accr, -acc accr
sets the initial AIM access class parameters, where accr is an access class range. Users at any authorization within the access class range inclusive are allowed to read and write to the resource (provided they also meet other access requirements). (See "Notes.")

-acs_path path
specifies the pathname of the access control segment (ACS) for this resource. You must create the ACS and set the desired access control list. If the ACS doesn't exist or you don't specify it, the default access is rew to the accounting owner and null to all others. If path is a null string, any existing ACS is disassociated from the resource.

-alloc STR
sets the allocation state of the resource to allocated or free, where STR must be either "on" (allocated) or "off" (free). The allocation state flag exists for your convenience and is largely ignored by resource management. (Default: off)

-attributes STR, -attr STR
searches for resources possessing the attributes specified in STR. If you give -attr with -nm, then the resource specified by the explicit name is searched for, and, when found, its attributes are set to those specified with -attr.

-comment STR, -com STR
specifies the desired value of the comment string for this resource, where STR can be an arbitrary comment string with a maximum length of 168 characters.

-lock STR
locks or unlocks the resource, preventing or allowing use of that resource, where STR must be either "on" (prevents use of the resource) or "off" (allows use of the resource). (Default: off)

-number N, -nb N
specifies that the number of such resources to be acquired is N. If you supply no -nb, 1 is assumed. You can supply -nb only if you supply no name.

ARGUMENTS

path
is the pathname of a segment, multisegment file, data management file, directory, extended entry, or link. This argument can consist of "-name STR" to specify a nonstandard entryname STR which already exists and which begins with a hyphen or contains ASCII control characters or any of the nonstandard characters ", <, >, $, %, ?, *, =, (,), [,], ::.

names
are additional names to be added. This argument can consist of "-name STR" when the entryname begins with a hyphen. The other nonstandard characters detailed above are not recommended for entrynames and this command will not generate entrynames which contain them.

CONTROL ARGUMENTS

-brief, -bf
suppresses the error message "Name already on entry."

-long, -lg
does not suppress the error message "Name already on entry." (Default)

ACCESS REQUIRED

You need modify permission on the parent directory.

NOTES

Two entries in a directory cannot have the same entryname; therefore add_name takes special action if the added name already exists. 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 you are informed of this action. If the added name is the only name of another entry, you are asked whether to delete this entry. If you answer "yes," the entry is deleted and the name is added to the entry specified by path; if you answer "no," no action is taken.

See 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 ==.personal
```

adds to every entry having a name with "private" as the last component a name with "personal" as the last component.

Name: add_pnotice

SYNTAX AS A COMMAND

```
add_pnotice path {-control_args}
```

FUNCTION

protects source code programs by adding, at the beginning of a program, a software protection notice (copyright public domain, or trade secret notice) in a box delimited as a comment. Multiple protection notices are supported. You can protect archives of source code programs using this command. The archive pathname convention is supported. If a particular language or suffix is not supported, an appropriate message is printed.

ARGUMENTS

path
: is the name of a source code program or an archive of source programs. You can give an archive component pathname to name a single archive component. You must include the language suffix or archive suffix.

CONTROL ARGUMENTS

-brief, -bf
: suppresses printing of both the source program name and the name of the pnotice that was added.

-default_copyright, -dc
: specifies that the notice to be added to the segment is the default copyright notice. Normally, this is a Honeywell copyright, but your site can change the default (see "Notes").

-default_trade_secret, -dts
: specifies that the notice to be added to the segment is the default trade secret notice. Normally, this is a Honeywell trade secret notice, but your site can change the default (see "Notes").

-long, -lg
: specifies that both the name of the source program and the name of the pnotice are printed when a protection notice is added. (Default)

-name STR, -nm STR
: where STR specifies the name of a protection notice template to be added (see "Notes").

NOTES

If you give no control arguments and there are no existing pnotices in the program, an error message is issued and no changes are made to the program. If copyright pnotices are found and you use -dc or -nm, the 10-year rule is applied to the named pnotice; that is, if the notice is more than nine years old, a new copy of the notice is added with the current year. If copyright pnotices exist and you give neither -dc nor -nm, the 10-year rule is applied to the most recent copyright pnotice.

You can obtain a list of available copyright and trade secret protection notice template names with list_pnotice_names; you can use -nm to specify any of these templates.

To list the pnotice segments in a source porgram, use display_pnotice.

A given program may contain several copyright notices or a trade secret notice or a public domain notice, but cannot contain a mixture of pnotice types.

-owner STR, -ow STR
specifies that this is an acquisition for the user specified by STR. If you give STR as "system," the resource is assigned to the system pool; if as "free," the resource is acquired to the free pool (effectively the same as no -ow). If STR is of the form Person_id.Project_id (where neither Person_id nor Project_id can be a star), the user specified has all the rights of ownership to the resource, as if he had acquired it personally, except that if you give "-rll on", the owner can't release (give up ownership of) the resource voluntarily. (See "Notes.")

-priv
specifies that a privileged call is to be made to obtain the status of this resource.

-release_lock STR, -rll STR
specifies whether this resource can be released by the owner or only by a privileged process, where STR must be either "on" or "off." If you supply no -rll, the resource can be released by the owner. (See "Notes.")

ACCESS REQUIRED

You need execute access to the rcp_admin_ gate to use -access_class, -owner, -priv, or -release_lock.

NOTES

This command acquires a resource for either you (requestor) or the user specified by -ow. If you are registered on more than one project and need corresponding access, or other users (on any project) need access to acquire a resource, you must create or modify the ACS. You must then specify the new/modified ACS by using "aqr -acs_path." The User_id (Person_id.Project_id) specifies the user to be added to, or deleted from, the ACS.

You must give -priv with -acc, -ow, and -rll.

Name: add_name, an

SYNTAX AS A COMMAND

```
an path names {-control_args}
```

FUNCTION

adds alternate name(s) to a segment, multisegment file (MSF), directory, link, data management (DM) file, or extended entry.

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_paths, 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 <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: add_search_rules, asr

SYNTAX AS A COMMAND

```
asr path1 {-control_args}...pathN {-control_args}
```

FUNCTION

adds pathnames and keywords to the search rules for object segments.

ARGUMENTS

pathi
is the absolute or relative pathname of a directory or one of the keywords listed below.

CONTROL ARGUMENTS

-after path, -af path
appends the previous path argument after the existing search rule named by path.

-before path, -be path
inserts the previous path argument before the existing search rule named by path.

-force, -fc
deletes any old occurrence of path in the search rules before adding the new rule.

-no_force, -nfc
fails and prints an error message if a rule to be added already exists in a different position. (Default)

LIST OF KEYWORDS

Both pathi and path arguments can be either pathnames or keywords. The defined keywords are:

initiated_segments
referencing_dir
working_dir

In addition, path in control args can be:

home_dir
process_dir
any site-defined keywords

NOTES

No warning is printed if a rule to be added already exists in the same position as that for which it is intended.

The limit on the number of search rules allowed for a process is 21.

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. (Default: the last nonzero word)

-chase
: chases links when using the star convention. (Default: to chase links only for nonstarred 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. (Default)

ACCESS REQUIRED

You must have write access on the segment or multisegment file.

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: after, af

SYNTAX AS A COMMAND

```
af STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[af STRA STRB]
```

FUNCTION

returns the string following the first occurrence of STRB in STRA. If STRB does not occur in STRA, the null string is returned.

EXAMPLES

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

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

Name: alm

SYNTAX AS A COMMAND

```
alm path {-control_args}
```

FUNCTION

ALM is the standard Multics assembly language. It is commonly used for privileged supervisor code, higher level support operators and utility packages, and data bases. It is occasionally used for efficiency or for hardware features not accessible in higher level languages; however, its routine use is discouraged.

The alm command invokes the ALM assembler to translate a segment containing the text of an assembly language program into a Multics standard object segment. A listing segment can also be produced. These segments are placed in the user's current working directory.

The ALM language is described briefly in this command description. The *Multics Processor Manual* (AL39) fully describes the instruction set.

ARGUMENTS

path
is the pathname of an ALM source segment that is to be translated by the ALM assembler. If path does not have a suffix of alm, one is assumed. However, the suffix must be the last component of the name of the source segment.

CONTROL ARGUMENTS

are optional arguments that can only appear after the path argument. The control arguments are:

-arguments STR, -ag STR
indicates that the assembled program may expect arguments. If present, it must be the last control argument to the alm command and must be followed by at least one argument. See "Macros in ALM" later in this description.

-brief, -bf
prevents errors from being printed on the terminal. Any errors are flagged in the listing (if one has been requested).

-list, -ls
produces an assembly listing segment.

-no_symbols
suppresses the listing of a cross-reference table in the listing segment. This cross-reference table is included by default in the listing segment when the -list control argument is given.

NOTES

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

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

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

The assembler is serially reusable and sharable, but cannot be reentered once translation has begun; that is, it cannot be interrupted during execution, invoked again, then restarted in its previous invocation.

ERROR CONDITIONS

Errors arising in the command interface, such as inability to locate the source segment, are reported in the normal Multics manner. Some conditions can arise within the assembler that are considered malfunctions in the assembler; these are reported by a line printed on the terminal and also in the listing. Any of the above cases is immediately fatal to the translation.

Errors detected in the source program, such as undefined symbols, are reported by placing one-letter error flags at the left margin of the erroneous line in the listing segment. Any line so flagged is also printed on the user's terminal, unless the -brief control argument is in effect. Flag letters and their meanings are given below.

LIST OF FLAGS

B mnemonic used belongs to obsolete (Honeywell Model 645) processor instruction set.

D error in macro definition or macro expansion; more detailed diagnostic for specific error given in listing.

E malformed expression in arithmetic field.

F error in formation of pseudo-operation operand field.

M reference to a multiply defined symbol.

N unimplemented or obsolete pseudo-operation.

O unrecognized opcode.

P phase error; location counter at this statement has changed between passes, possibly due to misuse of org pseudo-operation.

R expression produces an invalid relocation type.

S error in the definition of a symbol.

T undefined modifier (tag field).

U reference to an undefined symbol.

7 digit 8 or 9 appears in an octal field.

The errors B, E, M, O, P, and U are considered fatal. If any of them occurs, the standard Multics "Translation failed" error message is reported after completion of the translation.

ALM LANGUAGE

An ALM source program is a sequence of statements separated by newline characters or semicolons. The last statement must be the end pseudo-operation.

Fields must be separated by white space, which is defined to include space, tab, new page, and percent characters.

A name is a sequence of uppercase and lowercase letters, digits, underscores, and periods. A name must begin with a letter, period, or underscore and cannot be longer than 31 characters.

LABELS

Each statement can begin with any number of names, each followed immediately by a colon. Any such names are defined as labels, with the current value of the location counter. A label on a pseudo-operation that changes location counters or forces even alignment (such as org or its) might not refer to the expected location. White space is optional. It can appear before, after, or between labels, but not before the colon.

OPCODE

The first field after any labels is the opcode. It can be any instruction mnemonic or any one of the pseudo-operations listed later in this description under "Pseudo-operations." The opcode can be omitted, and any labels are still defined. White space can appear before the opcode, but is not required.

alm

OPERAND

Following the opcode, and separated from it by mandatory white space, is the operand field. For instructions, the operand defines the address, pointer register, and tag (modifier) of the instruction. For each pseudo-operation, the operand field is described under "Pseudo-operations" below. The operand field can be omitted in an instruction. Those pseudo-operations that use their operands generally do not permit the operand field to be omitted.

NOTES

Since the assembler ignores any text following the end of the operand field, this space is commonly used for comments. In those pseudo-operations that do not use the operand field, all text following the opcode is ignored and can be used for comments. Also, a quote character (") in any field introduces a comment that extends to the end of the statement. (The only exceptions are the acc, aci, and bci pseudo-operations, for which the quote character can be used to delimit literal character strings.) The semicolon ends a statement and therefore ends a comment as well.

INSTRUCTION OPERANDS

The operand field of an instruction can be of several distinct formats. Most common is the direct specification of pointer register, address, and tag (modifier). This consists of three subfields, any of which can be omitted. The first subfield specifies a pointer register by number, user-defined name, or predefined name (pr0, pr1, pr2, pr3, pr4, pr5, pr6, pr7). The subfield ends with a vertical bar. If the pointer register and vertical bar are omitted, no pointer register is used in the instruction. The second subfield is any arithmetic expression, relocatable or absolute. This is the address part of the instruction, and its default is zero. Arithmetic expressions are defined below under "Arithmetic Expressions." The last subfield is the modifier or tag. It is separated from the preceding subfields by a comma. If the tag subfield and comma are omitted, no instruction modification is used. (This is an all zero modifier.) Valid modifiers are defined below under "Modifiers."

Other formats of instruction operands are used to imply pointer registers. If a symbolic name defined by temp, tempd, or temp8 is used in the address subfield (it can be used in an arithmetic expression), then pointer register 6 is used if no pointer register is specified explicitly. This form can have a tag subfield.

Similarly, if an external expression is used in the address subfield, then pointer register 4 is implied; this causes a reference through a link. The pointer register subfield may not be specified explicitly. If a modifier subfield is specified, it is taken as part of the external expression; the instruction has an implicit n* modifier to go through the link pair. External expressions are defined below under "External Expressions."

A literal operand begins with an equal sign followed by a literal expression. The literal expression can be enclosed in parentheses. It has no pointer register but can have a tag subfield. A literal reference normally causes the instruction to refer to a word in a literal pool that contains the value of the literal expression. However, if the modifier du or dl is used, the value of the literal is placed directly in the instruction address field. Literal expressions are defined below under "Literal Expressions."

SPECIAL INSTRUCTION FORMATS

Certain instructions assembled by the ALM assembler do not follow the standard opcode-operand format as described above. These instructions fall into three basic classes: the repeat instructions, special treatment of the index and pointer register instructions, and EIS instructions. Each of these special cases is described below.

REPEAT INSTRUCTIONS

The repeat instructions are used to repeat either one or a pair of instructions until specified termination conditions are met. There are two basic forms:

```
rpt tally,delta,term1,term2,...,termN
```

generates the machine rpt instruction as described in the *Multics Processor Manual*. Both tally and delta are absolute arithmetic expressions. The termi specify the termination conditions as the names of corresponding conditional transfer instructions. This same format can be used with the rpt, rpd, rpda, and rpdb pseudo-operations:

```
rptx ,delta
```

generates the machine rpt instruction with a bit set to indicate that the tally and termination conditions are to be taken from index register 0. This format can be used with rplx and rpdx.

INDEX REGISTER INSTRUCTIONS

The opcodes for manipulation of the index registers have the general form opxN, where N specifies the index register to be used in the operation. ALM allows the more general form:

```
opx index,operand
```

which assembles opxN, where index is an absolute arithmetic expression whose value is N. This format can be used for all index register instructions.

POINTER REGISTER INSTRUCTIONS

As with the index register instructions, the opcodes for the manipulation of the pointer registers have the general form oprN, where N specifies the pointer register to be used. ALM extends this form to allow:

```
opr pointer,operand
```

which assembles as oprN, where N is found as follows: If pointer is a built-in pointer name (pr0, pr1, etc.), that register is selected; otherwise, pointer must be an absolute arithmetic expression whose value is N. This format can be used with all pointer register instructions except spri.

EIS MULTIWORD INSTRUCTIONS

An EIS multiword instruction consists of an operation code word, followed by one or more descriptor words. The descriptor words can be assembled by using the desc pseudo-operations listed under "Pseudo-operations" below. The operation code word has the following general form:

```
eisop (MF1),(MF2),keyword1(octexpression),keyword2
```

where:

MF1,MF2
are EIS modification fields as described in "EIS Modifiers" below.

keyword1
can be either fill, bool, or mask.

octexpression
is a logical expression that specifies the bits to be placed in the appropriate parts of the instruction.

keyword2
can be round, enablefault, or ascii; these cause single option bits in the instruction to be set.

Keywords can appear in any order, before or after an MF field. This format can be used for all Multics EIS multiword instructions.

EIS SINGLE-WORD INSTRUCTIONS

The Multics processor contains a set of 10 instructions that may be used to alter the contents of an address register. These instructions have the following general form:

```
opcode pr|offset,modifier
```

where:

pr
selects the address register that is to be modified by the instruction.

offset
is a value whose interpretation is dependent upon the opcode used.

modifier
must be one of the register modifiers (au, ql, x0, etc.).

These instructions have two modes of operation depending on the setting of bit 29 in the instruction. If bit 29 is 1, the current contents of the selected address register are used in determining its new contents; if bit 29 is 0, the contents of the word and bit offset portions of the selected address register are assumed to be zero at the start of the instruction (this results in a load operation into the selected address register). ALM normally sets bit 29 to 1, unless the opcode ends in x (e.g., awdx is an awd instruction with bit 29 set to 0). This format can be used with a4bd, a6bd, a9bd, abd, awd, s4bd, s6bd, s9bd, sbd, and swd.

EXAMPLES OF INSTRUCTION STATEMENTS

Seven examples of instruction statements are shown below. A brief description of each example follows the sample statements.

```
xlab:   lda     pr0|2,*                     " Example 1.
        eax7    xlab-1

        rccl    <sys_info>|[clock_],*       " Example 2.

        segref  sys_info,time_delta         " Example 3.
        adl     time_delta+1

        temp    nexti                       " Example 4.
        lxl0    nexti,*

        link    goto,<unwinder_>|[unwinder_]  " Example 5.
        tra     pr4|goto,*

        ana     =o777777,du                 " Example 6.
        ada     =v36/list_end-1

        a9bd    pr3|0,qu                    " Example 7(a).

        a9bdx   pr3|0,qu                    " Example 7(b).
```

Example 1 shows direct specification of address, pointer register, and tag fields. In the second instruction, no pointer register is specified, and the symbol xlab is not external, so no pointer register is used.

Example 2 shows an explicit link reference. Indirection is specified for the link, as the item at clock_ (in sys_info) is merely a pointer to the final operand.

Example 3 uses an external expression as the operand of the adl instruction. In this particular case, the operand itself is in sys_info.

Example 4 uses a stack temporary. Since the word is directly addressable using pr6, the modifier specified is used in the instruction.

Example 5 shows a directly specified operand that refers to an external entity. Unlike segref, it is necessary in this case to specify the pointer register and modifier fields.

Example 6 uses two literal operands. Only the second instruction causes the literal value to be stored in the literal pool.

In Example 7(a), the values in pr3 are added to the values calculated using the q register (see Section 4 of the Multics Processor Manual, AL39).

In Example 7(b), the word and bit offset of pr3 are replaced by those calculated using the q register.

ARITHMETIC EXPRESSIONS

An arithmetic expression consists of names (other than external names) and decimal numbers joined by the ordinary operators + - * /. You can use parentheses with their normal meaning.

An asterisk in an expression, when not used as an operator, has the value of the current location counter.

All intermediate and final results of the expression must be absolute or relocatable with respect to a single location counter. A relocatable expression cannot be multiplied or divided.

LOGICAL EXPRESSIONS

A logical expression is composed of octal constants and absolute symbols combined with the Boolean operators + (OR), - (XOR), * (AND), and ^ (NOT). You can use parentheses with their normal meaning.

EXTERNAL EXPRESSIONS

An external expression refers symbolically to some other segment. It consists of an external name or explicit link reference, an optional arithmetic expression added or subtracted, and an optional modifier subfield. An external name is one defined by the segref pseudo-operation. An explicit link reference must begin with a segment name enclosed in angle brackets (<>) and followed by a vertical bar (|). You can optionally follow this by an entryname in square brackets ([]). For example:

```
<segname>|[entryname]
<segname>|0,5*
```

An alternative form of external expression must begin with a segment name followed by a dollar sign. You can follow this by an entryname, an arithmetic expression, or a modifier, all of which are optional. For example:

```
segname$
segname$entryname-1
segname$+3,5
```

A segment name of *text, *link, or *static indicates a reference to this procedure's text, linkage, or static sections.

A segment name of *system indicates a reference to the external variable (or common block) entryname, which is managed by the linker. A link pair is constructed for each combination of segment name, entryname, arithmetic expression, and tag that is referenced.

LITERAL EXPRESSIONS

A literal reference causes the instruction to refer to a word in a literal pool that contains the value specified. However, the du and dl modifiers cause the value to be stored directly in the address field of the instruction. The literal pool is allocated in the text section. The various formats of literals are described in the following paragraphs.

A decimal literal can be signed. If it contains a decimal point or exponent, it is floating point. If the exponent begins with "d" instead of "e", it is double precision. A binary scale factor beginning with "b" indicates fixed point and forces conversion from floating point. The binary point in a literal with a binary scale factor is positioned to the right of the bit indicated by a decimal integer following the "b".

An octal literal begins with an "o" followed by up to 12 octal digits.

ASCII literals can occur in two forms. One form begins with a decimal number between 1 and 32 followed by "a" followed by the number of data characters specified by the integer preceding the "a", which can cross statement delimiters. The other form begins with "a" followed by up to four data characters, which can be delimited by the newline character.

A GBCD literal begins with "h" followed by up to six data characters, which can be delimited by the newline character. Translation is performed to the 6-bit character code.

An ITS (ITP) literal begins with "its" ("itp") followed by a parenthesized list containing the same operands accepted by the its (itp) pseudo-operation. The value is the same as that created by the pseudo-operation.

A variable-field literal begins with "v" followed by any number of decimal, octal, and ASCII subfields as in the vfd pseudo-operation. You must enclose it in parentheses if a modifier subfield is to be used.

If you use a variable-field literal, octal literal, or fixed-point literal (decimal literals with a "b" binary scale factor) with du or dl modification, then the lower 18 bits of the literal are placed in the address field of the instruction. If you use any other type of literal with du or dl modification, then the upper 18 bits of the literal are placed in the address field of the instruction.

MODIFIERS

These specify indirection, index register address modification, immediate operands, and miscellaneous tally word operations. They can be specified as 2-digit octal numbers (particularly useful for instructions like stba) or symbolically using the mnemonics described here.

Simple register modification is specified by using any of the register designators listed below. It causes the contents of the selected register to be added to the effective address.

Designators		Register
x0	0	index register 0
x1	1	index register 1
x2	2	index register 2
x3	3	index register 3
x4	4	index register 4
x5	5	index register 5
x6	6	index register 6
x7	7	index register 7
n	none	(no modification)
au		A bits 0-17
al		A bits 18-35 or 0-35
qu		Q bits 0-17
ql		Q bits 18-35 or 0-35
ic		instruction counter

In addition to the above, any symbol that is not otherwise a valid modifier (e.g., au, ql, x7) may be used as a modifier to designate an index register. Thus,

```
equ     regc,3
lda     sp|0,*regc
```

is equivalent to:

```
lda     sp|0,*3
```

Register-then-indirect modification is specified by using any of the register designators followed by an asterisk. If the asterisk is used alone, it is equivalent to the n* modifier. The register is added to the effective address, then the address and modifier fields of the word addressed are used in determining the final effective address. Indirect cycles continue as long as the indirect words contain an indirect modifier.

Indirect-then-register modification is specified by placing an asterisk before any one of the register designators listed above.

Direct modifiers are du and dl. They cause an immediate operand word to be fabricated from the address field of the instruction. For dl, the 18 address bits are right-justified in the effective operand word; for du they are left-justified. In either case, the remaining 18 bits of the effective operand are filled with 0's.

Segment addressing modifiers are its and itp; they can only occur in an indirect word pair on a double-word boundary. The addressing modifier its causes the address field of the even word to replace the segment number of the effective address, then continues the indirect cycle with the odd word of the pair. Nearly all indirection in Multics uses ITS pairs. For itp, see the *Multics Processor Manual*.

Tally modifiers i, ci, sc, scr, ad, sd, id, di, idc, and dic control incrementing and decrementing of the address and tally fields in the indirect word. They are difficult to use in Multics because the indirect word and the data must be in the same segment. Fault tag modifiers f1, f2, and f3 cause distinct hardware faults whenever they are encountered. The modifier f2 is reserved for use in the Multics dynamic linking mechanism; the other modifiers result in the signalling of the conditions fault_tag_1 and fault_tag_3.

EIS MODIFIERS

An EIS modifier appears in the first word of an EIS multiword instruction. It affects the interpretation of operand descriptors in subsequent words of the instruction. No check is made by ALM to determine whether the modifier specified is consistent with the operand descriptor specified elsewhere.

An EIS modifier consists of one or more subfields separated by commas. Each subfield contains either a keyword as listed below, a register designator, or a logical expression. The values of the subfields are OR'ed together to produce the result.

Keyword	*Meaning*
pr	Descriptor contains a pointer register reference.
id	Descriptor is an indirect word pointing to the true descriptor.
rl	Descriptor length field names a register containing data length.
xN	Descriptor address is offset by the value in index register N (N can be 0 - 7, as above).

SEPARATE STATIC OBJECT SEGMENTS

If a separate static object segment is desired, a joint pseudo-operation specifying static should exist in the program.

LIST OF PSEUDO-OPERATIONS

The pseudo-operations are listed below in alphabetical order. Additional pseudo-operations are provided by the macro facility. See "Macros in ALM" (following this list of pseudo-operations) for a further description of their syntax.

acc /string/,expression
assembles the ASCII string <string> into as many contiguous words as are required (up to 42). The delimiting character (/ above) can be any character other than white space. The quoted string can contain newline and semicolon characters. The length of the string is placed in the first character position in
acc format. If present, expression defines the length of the string; otherwise, the length is the actual length of the quoted string. If the given string is shorter than the defined length, it is padded on the right with blanks. If it is longer, it will be truncated to the defined length.

aci /string/,expression
is similar to acc, but no length is stored. The first character position contains the first character in aci format.

ac4 /string/,expression
is similar to aci, but only the rightmost four bits of each ASCII character are stored into the corresponding character position of a string of 4-bit characters. If the given string is shorter than the defined length, it is padded on the right with zeros.

arg operand
assembles exactly like an instruction with a zero opcode. Any form of instruction operand can be used.

bci /string/,expression
is similar to aci, but uses GBCD 6-bit character codes and GBCD blanks for padding.

bfs name,expression
reserves a block of expression words with name defined as the address of the first word after the block reserved.

bool name,expression
defines the symbol name with the logical value expression. See the definition of logical expressions above under "Logical Expressions."

bss name,expression
defines the symbol name as the address of a block of expression words at the current location. The name can be omitted, in which case the storage is still reserved.

call routine(arglist)
calls out to the procedure routine using the argument list at arglist. Both routine and arglist can be any valid instruction operand, including tags. If arglist and the parentheses are omitted, an empty argument list is created. All registers are saved and restored by call.

dec number1,number2,...,numberN
assembles the decimal integers number1, number2, through numberN into consecutive words.

desc4a address(offset),length
desc6a address(offset),length
desc9a address(offset),length
: generates one of the operand descriptors of an EIS multiword instruction. The address is any arithmetic expression, possibly preceded by a pointer register subfield as in an instruction operand. The offset is an absolute arithmetic expression giving the offset (in characters) to the first bit of data. It can be omitted if the parentheses are also omitted. The length is either a built-in index register name (al, au, ql, x0, etc.) or an absolute arithmetic expression for the data length field of the descriptor. The character size (in bits) is specified as part of the pseudo-operation name.

desc4fl address(offset),length,scale
desc4ls address(offset),length,scale
desc4ns address(offset),length,scale
desc4ts address(offset),length,scale
: generates an operand descriptor for a decimal string. The scale is an absolute arithmetic expression for a decimal scaling factor to be applied to the operand. It can be omitted, and is ignored in a floating-point operand. Data format is specified in the pseudo-operation name: desc4fl indicates floating point, desc4ls indicates leading sign fixed point, desc4ns indicates unsigned fixed point, and desc4ts indicates trailing sign fixed point. Nine-bit digits can be specified by using desc9fl, desc9ls, desc9ns, and desc9ts.

descb address(offset),length
: generates an operand descriptor for a bit string. Both offset and length are in bits.

dup expression
: duplicates all source statements following the statement containing the dup. The number of times that the statements are duplicated is equal to the value of the expression. This value must be positive and nonzero. Also, dup statements may not be nested.

dupend
: terminates the range of a dup pseudo-operation.

eight
: (see the even pseudo-operation)

end
: terminates the source segment.

entry name1,name2,...,nameN
generates entry sequences for labels name1, name2, through nameN and makes the externally defined symbols name1, name2, through nameN refer to the entry sequence code rather than directly to the labels. The entry sequence performs such functions as initializing base register pr4 to point to the linkage section, which is necessary to make external symbolic references (link, segref, explicit links). The entry sequence can use (alter) base register pr2, index registers 0 and 7, and the A and Q registers. It requires pr6 and pr7 to be properly set (as they normally are).

entrybound
places the current value of the location counter in the object_map entrybound field. If more than one such operation is encountered, the last one is effective. (See gate_macros.incl.alm). Setting the entry bound of the object segment's directory entry is still necessary (see hcs_$set_entry_bound).

equ name,expression
defines the symbol name with the arithmetic value expression.

even
inserts padding (nop) to a specified word boundary.

ext_entry label{/code_label} {,size} {,name}
makes a probe-able entry sequence for label with a stack frame size of size and with descriptors at label name. If you specify code_label, it is assigned the value of the address of the code associated with the entry sequence.

firstref extexpression1(extexpression2)
calls the procedure extexpression1 with the argument pointer extexpression2 the first time (in a process) that this object segment is linked to by an external symbol. If you omit extexpression2 and the parentheses, an empty argument list is supplied. The expressions are any external expressions, including tags.

If extexpression2 exists, the actual argument of the trap procedure (extexpression1) is a pointer to the link to extexpression2. For example, suppose that the trap procedure is an external pl1 procedure and that the argument is a number in the text section of the program containing the firstref statement, then the firstref statement looks like

```
firstref trap_proc$trap_proc(<*text>|firstref_argument)
```

where firstref_argument is a label on the number and the trap procedure looks like

```
trap_proc: proc(arg_ptr);
   dcl arg_ptr ptr;
   dcl based_ptr ptr based;
   dcl arg fixed fin (35) based;
   dcl number fixed bin (35);
   ...
   number = arg_ptr-->based_ptr-->arg;
   ...
```

getlp
sets the pointer register pr4 to point to the linkage section. You can use this with segdef to simulate the effect of entry. This operator can use pointer register pr2, index registers 0 and 7, and the A and Q registers, and requires pr6 and pr7 to be set properly.

include segmentname
inserts the text of the segment segmentname.incl.alm immediately after this statement. The "translator" (trans) search list is used to locate the segment (see the search commands).

inhibit off
instructs assembler to turn off the interrupt inhibit bit (bit 28) in subsequent instructions. This mode continues until you use the inhibit on pseudo-operation.

inhibit on
instructs assembler to turn on bit 28 in subsequent instructions. This mode continues until you use the inhibit off pseudo-operation.

itp prno,offset,tag
generates an ITP pointer referencing the prno pointer register.

its segno,offset,tag
generates an ITS pointer to the segno segment, word offset <offset>, with optional modifier tag. If the current location is not even, a word of padding (nop) is inserted, which causes any labels on the statement to be incorrectly defined.

join /text/name1,name2,.../link/name3,name4,.../static/name5,name6,....
appends the location counters name1, name2, etc., to the text section; appends the location counters name3, name4, etc., to the linkage section; and appends the location counters name5, name6, etc., to the static section. Any number of names can appear. Each name must have been previously referred to in a use statement. Any location counters not joined are appended to the text section. If you give both link and static in join pseudo-operations, a warning is printed on the terminal.

link name,extexpression
defines the symbol name with the value equal to the offset from lp to the link pair generated for the external expression extexpression. An external expression can include a tag subfield. The name is not an external symbol, so an instruction should refer to this link by "pr4|name,*".

maclist keyword {save}
indicates how listing of statements generated by macro expansion is to be done. The following keywords are accepted:

off
suppresses the listing of macro-generated statements and object code.

on
lists such statements and their associated object code.

object
lists only the object code .

restore
reverts the macro listing mode to a previously saved setting.

The save argument, if present, saves the current macro listing in a pushdown stack. The default macro listing mode is on.

macro name
indicates the start of a macro definition. When a macro name is defined, it may then be used as a pseudo-operation to trigger the expansion of the macro. See "Macros in ALM" below for a complete description of the definition and expansion of macros in ALM.

mod <expression>
inserts padding (nop) to an <expression> word boundary.

name objectname
specifies again the object segment name as it appears in the object segment. By default, the storage system name is used.

null
is ignored. This pseudo-operation is used for comments.

oct number1,number2,...,numberN
is like dec, with octal integer constants.

odd
(see the even pseudo-operation)

org expression
sets the location counter to the value of the absolute arithmetic expression <expression>. The expression can only use symbols previously defined.

perprocess_static
turns on the object segment's perprocess static switch. See the description of the run command for an explanation of perprocess static.

push expression
creates a new stack frame for this procedure, containing expression words. If expression is omitted (the usual case), the frame is just large enough to contain all cells reserved by temp, tempd, and temp8.

rem
(see the null pseudo-operation)

alm

return
is used to return from a procedure that has performed a push.

segdef name1,name2,...,nameN
makes the labels name1, name2, through nameN available to the linker for referencing from outside programs, using the symbolic names name1, name2, through nameN. Such incoming references go directly to the labels name1, name2 through nameN so the segdef pseudo-operation is usually used for defining external static data. For program entry points, the entry pseudo-operation is usually used.

segref segname,name1,name2,...,nameN
defines the symbols name1, name2, through nameN as external symbols referencing the entry points name1, name2, through nameN in segment segname. This defines a symbol with an implicit base register reference.

set name,expression
assigns the arithmetic value expression to the symbol name. Its value can be reset in other set statements.

short_call routine
calls out to routine using the argument list pointed to by pr0. Only pr4 and pr6 are preserved by short_call. short_return is used to return from a procedure that has not performed a push.

sixtyfour
(see the even pseudo-operation)

source_line num1,num2,num3,num4{,num5}
generates a statement map entry for statement num5 (default 1) of line num2 in file num1 that starts after character num3 and has a length of num4.

source_seg num1,path{,num2{,num3}}
generates a source map entry for file num1 with pathname path, a unique id of num2, and a date time contents modified clock value of num3. The unique id and dtcm will be looked up if not specified.

temp name1(n1),name2(n2),...,nameN(nN)
defines the symbols name1, name2, through nameN to reference unique stack temporaries of n1, n2, through nN words each. Each ni is an absolute arithmetic expression and can be omitted (the parentheses should also be omitted). The default is one word per namei.

temp8 name1(n1),name2(n2),...,nameN(nN)
is similar to temp, except that 8-word units are allocated, each on an 8-word boundary.

tempd name1(n1),name2(n2),...,nameN(nN)
is similar to temp, except that n1 (n2 through nN) double words are allocated, each on a double-word boundary.

use name
assembles subsequent code into the location counter name. The default location counter is ".text.".

vfd T1L1/expression1,T2L2/expression2,...,TNLN/expressionN
is variable format data. Each expressioni is of type Ti and is stored in the next Li bits of storage. As many words are used as required. Individual items can cross word boundaries and exceed 36 bits in length. Type is indicated by the letters "a" (ASCII constant) or "o" (logical expression) or none (arithmetic expression). Regardless of type, the low-order Li bits of data are used, padded if needed on the left. The Ti can appear either before or after Li.

Restrictions: The total length of the variable format data cannot exceed 128 words. A relocatable expression cannot be stored in a field less than 18 bits long, and it must end on either bit 17 or bit 35 of a word.

zero expression1,expression2
assembles expression1 into the left 18 bits of a word and expression2 into the right 18 bits. Both subfields default to zero.

MACROS IN ALM

The ALM macro facility provides a means for defining and using sequences of text to be inserted at various points in an ALM program. Each such sequence of text, called a macro, is defined by the use of the macro pseudo-operation in ALM. A macro definition consists of all text following the line containing the macro pseudo-operation until the character string, &end. The sequence of text is named by the symbol appearing as the operand to the macro pseudo-operation.

At any point in a program subsequent to the definition of a macro, the macro name can be used as a pseudo-operation in ALM. Whenever it is so used, ALM inserts the text sequence defined as that macro.

The macro facility is purely text manipulative. It deals with macro definitions as a continuous stream of text characters interspersed with control sequences. Each control sequence begins with the & character. The control sequence &end terminates the macro definition. When a macro is invoked by using its name as a pseudo-operation, the macro definition is scanned from left to right. All text between control sequences is copied, and variable information is inserted in place of the control sequences. The resulting macro expansion is presented to ALM for assembly.

Macros may be given arguments by placing operands in fields corresponding to the operands of a pseudo-operation. These arguments can be substituted into the expanded copy of the macro as specified by various control sequences within the macro definition. Control sequences are also provided to facilitate iteration, conditional text selection, unique symbol generation, and other operations.

The macro facility also provides a set of special pseudo-operations that are distinct from the regular ALM pseudo-operations. These special pseudo-operations allow for the conditional assembly of source lines and the printing of messages to the user's terminal during assembly. The argument syntax of these pseudo-operations is the same as that of macros, not the expressions and symbols of the ALM assembler.

CONTENTS OF A MACRO

The body of a macro (i.e., the text starting on the line following the macro pseudo-operation and ending just before the character string &end) can include any text and control sequences which, when expanded, yield valid ALM source code. The body of a macro can include invocations of other macros and even the definition of other macros.

Macro definitions are shown in the assembly listing with their internal line numbers to the left of the ALM source line number. (These internal numbers are used in diagnostics produced by the macro expander.) Macros may be redefined, the later definition replacing the earlier. Macros may also redefine all existing ALM operations and pseudo-operations.

An example macro is given below:

```
macro  move_a_to_b
lda    a
sta    b
&end
```

INVOKING A MACRO

A macro is invoked by specifying its name as a pseudo-operation. Arguments to the macro can appear in the variable field separated by commas. A comment may follow the argument list, separated from it by white space or a double quote. Arguments to macros that include spaces, tabs, newline characters, commas, or semicolons must be enclosed in matching parentheses. The parentheses are stripped from the argument during macro expansion. The use of parentheses allows macro invocations to extend over several lines. Argument lists may also be continued on successive source lines by following the last macro argument of a line with a comma. Leading white space preceding the continuation of the argument list on the next line is ignored.

Code and statements produced by the macro facility are placed in the assembly listing without source line numbers. Symbols used by a macro expansion appear in the cross-reference listing as though they were referenced on the line of the macro invocation. The listing of statements produced by macro expansion may be controlled through the use of the maclist pseudo-operation. See the description under "Pseudo-operations" above.

RESTRICTIONS

Any macro definition that begins in an include file must end in that include file.

A macro must be defined before it is expanded. It can appear before its definition within another macro definition, but that other macro may not be expanded until the macro it invokes is defined.

Macros may be invoked in code produced by macro expansions. The depth of such recursion, however, must not exceed the current limit of 100.

LIST OF CONTROL SEQUENCES

Character substitutions and conditional expansions at the time of macro expansion are effected by the control sequences detailed below. The use of any ampersand followed by any sequence not defined below is noted by ALM as an assembly error.

&0, &1, &2
the character & followed immediately by any positive decimal integer (< 100) is replaced, upon expansion, with the corresponding argument passed to the macro (see "Notes" and "Examples" below).

The special sequence &0 causes a reference to a unique label at the start of the macro expansion. The label is generated only if the &0 sequence is generated within a macro.

&u
is expanded to be a unique character string of the form ...00000, ...00001, etc., that is different from any other such strings expanded with &u control.

&p
is expanded to be the same string as the previous &u expansion.

&n
is expanded to be the same string as the next &u expansion.

&U
is expanded to be a unique character string of the form .._00000, .._00001; however, multiple occurrences of &U within the same macro yield the same string.

&(N
indicates the beginning of an iteration sequence. The text following the &(N and up to but not including the next &) is expanded repeatedly (see "Iteration" below).

&i
is expanded to the particular element of the iteration set for which the current iteration is being performed (see "Iteration" below).

&x
is expanded into the decimal integer corresponding to the relative position of the particular element of the iteration set over which the current iteration is being performed.

&AN
is expanded to be the Nth argument following the -ag or -arguments control argument to the alm command.

&K
is expanded as a decimal number equal to the number of arguments in the current macro invocation.

&k
is expanded as a decimal number equal to the number of elements in the current iteration set.

&lN
is expanded as a decimal number equal to the length in characters of the Nth argument in the current macro invocation.

&&
is expanded to a single & character. This facilitates macro definitions within macro expansions.

&FN
expands to a string constructed by concatenating all arguments to the macro invocation, from the Nth onward, separated by commas. If N is not given, 1 is assumed.

&FqN or &FQN
is similar to &FN, except that each argument is enclosed in parentheses as it is concatenated to the expanded string. This control sequence should be used when sublists of macro arguments are to be passed to other macros and there is a possibility that some of these arguments may contain white space, newline characters, etc.

&fN
is similar to &FN, except that the elements of the current iteration set are concatenated.

&fqN or &fQN
is similar to &FqN and &FQN, except that the elements of the current iteration set are enclosed in parentheses.

&RM,N
is used to cause iteration over the arguments in a macro invocation, as opposed to the iteration elements of a single macro argument. The use of &R affects the operation of the next &(control sequence. The M is a decimal number equal to the number of the first argument to be selected; N is a decimal number equal to the number of the last argument to be selected. If N is missing or zero, it is assumed to be equal to the number of arguments in the macro invocation. If M is missing or zero, it is assumed to be 1 (see "Notes" below).

&[
marks the start of a selection group. The text following the &[and up to but not including the matching &] is expanded conditionally. The elements of a selection group are separated by the control sequence & . Each element can contain other selection groups to a nesting depth of 10. When a macro is expanded, only one element of a selection group is used. This element is chosen by a control sequence preceding the &[control sequence.

&sN
selects the Nth element of the following selection group. All expanded text between the &s and &[control sequences is interpreted as the decimal number N. If N is zero or greater than the number of elements in the selection group, no element is selected.

&=c1,c2
all expanded text between the &= and the next &[control sequence is broken into two character strings. If no comma is found in the expanded text, c2 is taken to be a null string. If the two strings are equal, by character string comparison, the first element of the following selection group is used. Otherwise, the second element, if present, is used.

&^=c1,c2
the &^= control sequence is identical to the &= control sequence, except that the first element is selected if the strings are unequal, and the second, if present, is selected if they are equal.

&>n1,n2
&<n1,n2
&>=n1,n2
&<=n1,n2
these control sequences are similar to the &= and &^= control sequences, except that the expanded text between this control sequence and the next &[control sequence is interpreted as two decimal integers. If no comma is found, n2 is taken to be zero. An arithmetic comparison of the numbers is performed, as specified by the particular control sequence used. A result of true causes the first element of the following selection group to be used. A result of false causes the second element, if present, to be used.

&end
signifies the end of the macro definition. The statement containing the &end control sequence is not part of the macro body, and hence, is not included as part of the macro definition.

NOTES

Decimal numbers produced by &K, &k, and &x are generated with no leading blanks or zeros. The number zero is generated as the single digit 0.

Numeric arguments to &N, &(N, &FN, &fN, &FqN, &fqN, and &AN can be comprised of from zero to three digits. These numbers must appear as such in the unexpanded macro definition. If numeric text is to follow one of the above control sequences, all three digits of N must be supplied.

The numbers used by &RM,N, as well as the strings and numbers used by the relational and selection control sequences can be of any length. They appear in the expanded text and need not necessarily be in the macro definition. These expanded strings and numbers are, of course, not placed in the final macro expansion being generated.

If a given macro argument is not specified in a particular invocation of that macro, a null character string is used for that argument during macro expansion.

ITERATION

The macro facility provides the ability to map the expansion of a subset of a macro definition over a set of elements, expanding that part of the definition repeatedly, selectively substituting each element of the iteration set in turn. By means of this technique, lists may be processed.

An iteration set consists of elements separated by commas. It has the same syntax as the argument list of a macro invocation, including conventions on the use of parentheses for quoting and continuation via the trailing comma. Two types of iteration sets may be referenced in a macro expansion:

The argument list to a macro invocation itself may be used as an iteration set, in which case the arguments of the macro invocation are the elements. This type of iteration set is specified by means of the &R control sequence.

Any argument to a macro invocation may be used as an iteration set, if it, internally, has the same syntax as an argument list to a macro invocation. This type of iteration set is specified when &R is not used.

The text between the sequences &(and &) is expanded once for each element in the iteration set, in left to right order. If the second form of iteration set is used, the number of the argument to the macro invocation may appear (one to three digits, no digits are mapped into 1) immediately after the &(sequence. Any occurrence of the sequence &i between the sequences &(and &) is replaced by the current element of the iteration set. The sequence &x is replaced by the decimal number of the relative position of that element in the iteration set (not the argument number, in the first type of iteration set). Iterations may not be nested. Any iteration that starts in an element of a selection group must end in that element of a selection group. No iteration may end in any element of a selection group unless it started in that element of that selection group.

MACRO FACILITY PSEUDO-OPERATIONS

The macro facility provides a set of pseudo-operations in addition to the macro pseudo-operation already described. These pseudo-operations are different from the other pseudo-operations provided by the assembler insofar as the syntax of their arguments, which is the syntax of macro invocation arguments, with all quoting and continuation conventions of them, and not the syntax of other pseudo-operation arguments to the assembler.

The use of these pseudo-operations, like all other ALM pseudo-operations, is not limited to code produced by macro expansion. They can be placed anywhere in source segments and include files, as well as in macro code, but the conditional pseudo-operations can not be nested.

warn
: prints out its first argument on the user's terminal, preceded by the string "ALM assembly:" and followed by a newline character. This argument, without the prefix, is also placed in the program listing.

ife
: the character strings that are the first and second arguments to ife are compared. If they are the same character string, all assembler statements between the one containing the end of the argument list to ife, and the next one containing the string ifend in any context at all are assembled. No part of the line containing the string ifend is assembled. If the first and second arguments are not equal, none of these lines are assembled.

ine
: the same as ife, but assembly of the text up to ifend proceeds only if the first two arguments are not equal by character string comparison.

ifint
: the first argument to the ifint pseudo-operation is inspected to see if it is a valid decimal integer. If so, all assembler statements between the one containing the end of the argument list to ifint and the next one containing the string ifend in any context at all are "If the first argument to ifint is not a valid integer, none of these lines are assembled.

inint
: the same as ifint, but assembly of the text up to ifend proceeds only if the first argument is not a valid decimal integer.

ifarg
all of the arguments to the alm command following the -ag or -arguments control argument are inspected, and compared with the first argument to ifarg. If any of these command arguments compare equal, by character string comparison, to the first argument to ifarg, all assembler statements between the one containing the end of the argument list to ifarg and the first one containing the string ifend in any context at all are assembled. No part of the line containing the ifend is assembled. If the first argument to ifarg does not appear among the arguments following -ag or -arguments, none of these lines are assembled.

inarg
the same as ifarg, but assembly of the text up to ifend proceeds only if the first argument to inarg is not found among the arguments to the alm command following -ag or -arguments.

In all of the conditional constructs above, the key string, ifend, must appear in the same source segment or macro expansion as the statement containing the conditional pseudo-operation. If the ifend key string appears in the ifend_exit string, and the entire construct appears in a macro expansion, and the predicate of the conditional construct is met (i.e., the statements are being assembled, not skipped), the assembler ceases to take input from that macro expansion, as though the last statement in that macro expansion had been assembled.

EXAMPLES

The following macro definitions show typical expansions:

```
        macro   load
        ld&1     &2
        &end
```

might be used as follows:

```
   load         x0,temp                    ldx0    temp
or:
   load         a,(sp|3,*)                 lda     sp|3,*
```

The use of parentheses in the second example causes the comma to be ignored as a parameter delimiter. The macro definition:

```
        macro   test
        lda     &1
        tpl     &U
        sta     last_minus
&U      sta     &2
        &end
```

might be used as follows:

```
test          a,b                              lda       a
                                               tpl       ..._00000
                                               sta       last_minus
                                     ..._00000: sta       b
```

The following example shows how iteration is used. The macro definition:

```
                    macro     table
&R&(                vfd       18/&i,18/&0
&)
                    & end
```

might be used as follows:

```
el:       table     4,6,8,10                         vfd       18/4,18/el
                                                     vfd       18/6,18/el
                                                     vfd       18/8,18/el
                                                     vfd       18/10,18/el
```

The following example shows how conditional expansion can be used. The macro definition:

```
                    macro     meter
                    lda       &1
                    ife       &2,on
                    aos       meterword,al
                    ifend
                    &end
```

might be used as follows:

```
meter     foo,on                           lda       foo
                                           aos       meterword,al
```

The following macro shows how &x might be used. The macro definition:

```
                    macro     callm
&(3                 eppbp     &i
                    spribp    &2+&x*2
&)
                    eaq       2*&x-2
                    lls       36
                    staq      &2
                    call      &1(&2)
                    &end
```

might be used as follows:

```
                    callm     sys,arg,(=1,(=14aError from ^d.))
```

yielding:

```
                    eppbp     =1
                    spribp    arg+1*2
                    eppbp     =14aError from ^d.
                    spribp    arg+2*2

                    eaq       2*4-2
                    lls       36
                    staq      arg
                    call      sys(arg)
```

The following macro definition shows how conditional expansion might be used:

```
                    macro     tab9
&R&(&=&x,1&[                  vfd       &;,&]o9/&i&)
                    &end
```

This macro might be invoked as follows:

```
                    tab9      16,42,13,36,67
```

expanding to:

```
                    vfd       o9/16,o9/42,o9/13,o9/36,o9/67
```

The following example shows how macros may be defined by macros, and used to powerful effect. These macros allow a call like a PL/I call to be generated, with descriptors.

The following macro is invoked to declare variables by specifying their address, data type, and precision:

```
                    macro     declare
                    macro     dcl_&1
                    epp0      &2
                    epp1      =v1/1,6/&3,17/0,12/&4
                    &&end
               &end
```

This macro may be invoked as follows:

```
     declare        count,buffer+2,fixed,17
```

or:

```
     declare        progname,(lp|xlink,*),char,32
```

These macro invocations cause the following macro definitions to be produced:

```
        macro     dcl_count
        epp0      buffer+2
        epp1      =v1/1,6/fixed,17/0,12/17
        &end

        macro     dcl_progname
        epp0      !p|xlink,*
        epp1      =v1/1,6/char,17/0,12/32
        &end
```

Assume that at some point in the assembly the statements:

```
        equ       char,21
        equ       fixed,1
```

defining the PL/I descriptor types for these data types appear.

The following macro definition, when invoked, generates a full PL/I call with descriptors. Assume that the statement:

```
        tempd arg1(16)
```

appears at some point in the program.

```
        macro     gcall
&R2&(   dcl_&i
        spri0     arg1+2*&x
        spri1     arg1+2*&K-2+2*&x
&)
        ldaq      =v18/2*&K-2,18/0,18/2*&K-2,18/4
        staq      arg1
        call      &1(argi)
        &end
```

When the following macro invocation is issued:

```
        gcall program,count,progname
```

the following expansion is immediately produced:

```
        dcl_count
        spri0     arg1+2*1
        spri1     arg1+2*3-2+2*1
        dcl_progname
        spri0     arg1+2*3-2
        spri1     arg1+2*3-2+2*2

        ldaq      =v18/2*3-2,18/0,18/2*3-2,18/4
        staq      arg1
        call      program(arg1)
```

This is further expanded when the dcl_count and dcl_progname macros are expanded to:

```
epp0      buffer+2
epp1      =v1/1,6/fixed,17/0,12/17
spri0     arg1+2*1
spri1     arg1+2*3-2+2*1
epp0      lp|xlink,*

epp1      =v1/1,6/char,17/0,12/32
spri0     arg1+2*2
spri1     arg1+2*3-2+2*2
ldaq      =v18/2*3-2,18/0,18/2*3-2,18/4
staq      arg1

call      program(arg1)
```

which is precisely the code required for a full PL/I call.

Name: alm_abs, aa

SYNTAX AS A COMMAND

```
aa paths {-alm_arg} {-dp_args} {-control_args}
```

FUNCTION

submits an absentee request to perform ALM assemblies. The absentee process for which alm_abs submits a request assembles the segments named and sends to the printer and deletes each listing segment if it exists. If you don't give -output_file, an output segment (path.absout) is created in your working directory. If you supply more than one path, the first is used. If the segment to be assembled cannot be found, no absentee request is submitted.

ARGUMENTS

paths
: are pathnames of segments to be assembled.

alm_arg
: can be the -list control argument to the alm command.

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

CONTROL ARGUMENTS

-hold
specifies that alm_abs should not dprint or delete the listing segment.

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

-output_file path, -of path
specifies that absentee output is to go to a segment with a pathname of path.

-queue N, -q N
is the priority queue of the request. The default queue is defined by your system administrator. (See "Notes.")

NOTES

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

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

When performing several assemblies, it is more efficient to give several segment pathnames in one command rather than several commands. With one command, only one process is set up. The links that need to be snapped when setting up a process and when invoking the assembler need be snapped only once.

If the -queue control argument is not specified, the request is submitted into the default absentee priority queue defined by the site and, if requested (via -list), the listing files are dprinted in the default queue of the request type specified on the command line (via dp_args). (If no request type is specified, the "printer" request type is used.)

If requested (via -list) when the -queue control argument is specified, the listing files are dprinted in the same queue as is used for the absentee request. If the request type specified for dprinting (via dp_args) does not have that queue, the highest-numbered (i.e., the lowest priority) queue available for the request type is used and a warning is issued.

Name: and

SYNTAX AS A COMMAND

```
and {tf_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[and {tf_args}]
```

FUNCTION

returns true if all the tf_args are equal to true, otherwise it returns false. If there are no tf_args, it returns the and-identity "true". If any of the tf_args has a value other than true or false, an error message is printed.

EXAMPLES

The command line

```
!   and [st -ssw ([segs **])]
```

returns true if all segments in the current working directory have their safety switches on, or if there are no segments in the working directory.

The active function

```
   [and [equal &r1 a] [equal &r2 b]]
```

inside an exec_com returns true only if the first argument to ec is "a" and the second is "b".

Name: answer

SYNTAX AS A COMMAND

```
answer STR {-control_args} command_line
```

FUNCTION

provides preset answers to questions asked by another command.

ARGUMENTS

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

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

CONTROL ARGUMENTS

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

-call STR
evaluates the active function string STR to obtain the next answer in a sequence. STR must be quoted if it contains command language characters. The surrounding brackets must be omitted, as in "segs *.pl1". The return value "true" is translated to "yes", and "false" to "no". All other return values are passed as is.

-match STR
answers only questions whose text matches STR. If STR is surrounded by slashes (/), it is interpreted as a qedx regular expression. Otherwise, answer tests whether STR is literally contained in the text of the question. Multiple occurrences of -match and -exclude are allowed (see Notes below). They apply to the entire command line.

-exclude STR, -ex STR
passes on, to the user or other handler, questions whose text matches STR. If STR is surrounded by slashes (/), it is interpreted as a qedx regular expression. Otherwise, answer tests whether STR is literally contained in the text of the question. Multiple occurrences of -match and -exclude are allowed (see Notes below). They apply to the entire command line.

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

-then STR
supplies the next answer in a sequence.

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

NOTES

Answer provides preset responses to questions by establishing an on unit for the condition command_question, and then executing the designated command. If the designated command calls the command_query_ subroutine to ask a question, the on unit is invoked to supply the answer. The on unit is reverted when the answer command returns to command level. See the Programmer's Reference Manual for a discussion of the command_question condition.

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

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

The -match and -exclude control arguments are applied in the order specified. Each -match causes a given question to be answered if it matches STR, each -exclude causes it to be passed on if it matches STR. A question that has been excluded by -exclude is reconsidered if it matches a -match later in the command line. For example, the command line

```
answer yes -match /fortran/ -exclude /fortran_io/
   -match /^fortran_io/
```

answers questions containing the string "fortran", except that it does not answer questions containing "fortran_io", except that it DOES answer questions BEGINNING with "fortran_io".

EXAMPLES

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

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

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

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

The help command prints the first block, then prints another block every time the user answers yes. In this example, the first three blocks are printed before the user is interrogated. Sequences of answers are especially useful in exec_coms and absentee jobs. To supply the sequence of answers "yes, no, no, yes", type:

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

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

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

To supply the answer "start_seg" once and call the temp_seg active function successive times, type:

```
answer start_seg -call "temp_seg args" command_line
```

To substitute the query "More?" for the one printed by help, type:

```
answer -call "query More?"  -bf help fo
```

Name: apl, v2apl

SYNTAX AS A COMMAND

```
apl {workspace_id} {-control_args}
```

FUNCTION

invokes the APL interpreter, optionally loading a saved workspace.

ARGUMENTS

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

CONTROL ARGUMENTS

-brief_errors, -bfe
: prints short error messages. (Default)

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

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

-long_errors, -lge
: prints long error messages. The short form of the message is printed, followed by a more detailed explanation of the error.

-no_quit_handler, -nqh
: ignores the quit condition. (Default: to trap all quits within APL)

-temp_dir path, -td path
changes the directory that is used to hold the temporary segments that contain the active workspace to path. (Default: to use the process directory)

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

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

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

-user_number N
sets the APL user number (returned by some APL functions) to N. (Default: 100)

NOTES

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

For a complete description of the APL language, terminal conventions, and directions for converting Version 1 APL workspaces, refer to *Multics APL* (AK95).

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. For more information on how archives can be sorted and reordered, see the archive_sort and reorder_archive commands.

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 you do 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 you can employ to create new archive segments and to maintain 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 your 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 your 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 your 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 your working directory.

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

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

cudf
: copy, update, and delete force; operates like udf, placing an updated copy of the archive segment in your 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 your 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 your 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., you are 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. You are 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 you 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 you when (unforced) deletion is requested of a segment to which you do not have write permission. If you 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 you have 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
```

You create 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                 09/12/82 1435.0 rw  09/12/82 1434.2  441
beta                  09/12/82 1435.0 rw  09/12/82 1434.2  257
gamma                 09/12/82 1435.0 rw  09/12/82 1434.2  694
delta                 09/12/82 1435.0 rw  09/12/82 1434.2  109
```

After changing the segment delta, you replace it in the archive segment and appends (using the replace operation) the segment epsilon to the archive segment. You also delete 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

09/12/82 1435.0       alpha
09/12/82 1435.0       beta
09/12/82 1437.5       delta
09/12/82 1437.5       epsilon
```

You later replace 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

09/12/82 1641.5       alpha
09/12/82 1435.0       beta
09/12/82 1437.5       delta
09/12/82 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, you copy and update 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

09/12/82 1648.3       alpha
09/12/82 1435.0       beta
09/12/82 1437.5       delta
09/12/82 1437.5       epsilon

ac t <dir_one>greek

       >udd>Project_id>dir_one>greek.archive

 updated                 name

09/12/82 1641.5       alpha
09/12/82 1435.0       beta
09/12/82 1437.5       delta
09/12/82 1437.5       epsilon
```

Notice that the entry in the updated column for the component alpha differs in the two tables of contents. Finally, you extract 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
```

Name: archive_sort, as

SYNTAX AS A COMMAND

```
as paths
```

FUNCTION

sorts the components of an archive segment. The components are sorted into ascending order by name using the standard ASCII collating sequence. The original archive segment is replaced by the sorted archive. For more information on archives and reordering them, see the archive and the reorder_archive commands.

ARGUMENTS

paths
: are the pathnames of the archive segments to be sorted. You need not supply the archive suffix.

NOTES

There can be no more than 1000 components in an archive segment that is to be sorted.

Storage system errors encountered while attempting to move the temporary sorted copy of the archive segment back into your original segment result in diagnostic messages and preservation of the sorted copy in your process directory. If the original archive segment is protected, you are interrogated to determine whether it should be overwritten.

Name: archive_table, act

SYNTAX AS A COMMAND

```
act archive_path {starnames} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[act archive_path {starnames} {-control_args}]
```

FUNCTION

returns the names of specified archive components in a specified archive segment. As a command, archive_table prints one component name per line. As an active function, it returns names individually requoted and separated by single spaces.

ARGUMENTS

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

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

CONTROL ARGUMENTS

-absolute_pathname, -absp
: causes component names to be returned as absolute pathnames, of the form ARCHIVE_DIR>ARCHIVE_NAME::COMPONENT_NAME, rather than just the component names.

-bit_count, -bc
: returns the bit count of the selected components.

-component_name, -cnm
: returns only the component name portion of the selected components. It has no effect if -no_name is selected. (Default)

-date_time_contents_modified, -dtcm
: returns the date-time-contents-modified of the segment when the component was last updated in the archive.

-date_time_updated, -dtud
: returns the date-time when the selected components were last updated in the archive.

-header, -he
: prints a header. Not accepted by the active function.

-mode, -md
returns the access mode of the selected components.

-name, -nm
returns the name of the selected components. (Default)

-no_bit_count, -nbc
suppresses bit count information. (Default)

-no_date_time_contents_modified, -ndtcm
suppresses date-time-contents-modified information. (Default)

-no_date_time_updated, -ndtud
suppresses component update time information. (Default)

-no_header, -nhe
prints no header. (Default)

-no_name, -nnm
suppresses component name information.

-no_requote
does not requote component attribute groups.

-requote
causes the attributes of each component to be requoted as a single entity. This control argument is ignored by the command. (Default)

NOTES ON ACTIVE FUNCTION

If -name is given, archive_table always requotes the component name (if -component_name is selected) or archive pathname (if -absolute_pathname is selected).

If more than one of -bit_count, -date_time_contents_modified, -date_time_updated, -mode, and -name is supplied, the selected attributes are returned, separated by a space. The order of items is always: name, date-time-contents-modified, mode, date-time-updated, bit count; which is the same order found when using the archive command's "tl" key.

If -no_requote is used, the selected attributes for each component are returned separated by spaces. If more than one component is specified, successive component attributes are separated by a space.

If -requote is given, the selected attributes for each component are requoted separated by spaces. If more than one component is supplied, then each component's requoted attribute group is separated from the others by a space.

EXAMPLES

The following examples assume an archive ("sample") with three components: "one", "two", and "three (3)". Note that the practice of including characters such as " " in segment names typically requires more complicated command line constructs and should be avoided; it is included here only to clarify the requoting of names.

```
!  act sample -he -nm -dtcm -dtud -mode -bc

     >udd>A_Project>A_person>sample.archive

NAME            UPDATED          MODE          MODIFIED          LENGTH

one          12/01/82_1513.4      r        12/01/82_1512.9        36
two          12/01/82_1513.4      r        12/01/82_1513.0        63
three (3)    12/01/82_1513.4      r        12/01/82_1513.2        90

!  do "do ""format_line """""name = ^a, length = ^a."""""" &&rf1"" &f1"
!... ([act sample -m -bc])
name = one, length = 36.
name = two, length = 63.
name = three (3), length = 90.

!  do "do ""format_line """""item = ^a."""""" &&rf1"" &rf1"
!... ([act sample -nm -bc -no_requote])
item = one.
item = 36.
item = two.
item = 63.
item = three (3).
item = 90.

!  format_line "bit count of component ""one"" is ^a."
!... [act sample -nnm -bc one]
bit count of component "one" is 36.
```

Name: area_status

SYNTAX AS A COMMAND

```
area_status virtual_pointer {-control_args}
```

FUNCTION

displays some information about an area.

ARGUMENTS

virtual_pointer
is a virtual pointer specifier to the area to be looked at (see Section 1 for a description of virtual pointers).

CONTROL ARGUMENTS

-long, -lg
dumps the contents of each block in both octal and ASCII format.

-trace
displays a trace of all free and used blocks in the area.

NOTES

If the area has internal format errors, they are reported. The command does not report anything about (old) buddy system areas except that the area is in an obsolete format.

Name: assign_resource, ar

SYNTAX AS A COMMAND

```
ar resource_type {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ar resource_type {-control_args}]
```

FUNCTION

calls the Resource Control Package (RCP) to assign a resource to the user's process.

ARGUMENTS

resource_type
specifies the type of resource to be assigned. Currently, only device types can be designated. The -device control argument is used to name a specific device to assign. Other control arguments are used to indicate characteristics of the device to be assigned. The following device type keywords are supported:

tape_drive
disk_drive
console

printer
punch
reader
special

CONTROL ARGUMENTS

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

-density N, -den N
supplies the density capability characteristic of a tape drive. There can be more than one instance of this argument. A tape drive is assigned that is capable of being set to all of the indicated densities. The acceptable values for this argument are: 200, 556, 800, 1600, and 6250.

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

-device STR, -dv STR
specifies the name of the device to be assigned. If used, other control arguments that indicate device characteristics are ignored (see "Examples" below). If used with -long, a message containing the name of the assigned device is printed on the user's terminal; otherwise, no message is printed. If found several times on the command line, the last one supplied overrides any previous ones.

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

-long, -lg
prints all of the device characteristics of the assigned device. If not supplied, only the name of the assigned device is printed.

-model N
specifies the device model number characteristic. Only a device that has this model number is assigned. To find the acceptable model numbers, use print_configuration_deck.

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

-speed N
gives the speed of a tape drive. The acceptable values depend on the particular hardware on the system and can be: 75, 125, or 200.

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

-track N, -tk N
specifies the track characteristic of a tape drive. The value can be either 9 or 7. If both -track and -volume are not given, a track value of 9 is used when assigning a tape device.

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

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

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

NOTES

An assigned device still must be attached by a call to some I/O module. If a device is successfully assigned, the name of the device is printed. If the user requests a specific device that is successfully assigned, the name of the device is not printed unless the user asks for it.

NOTES ON ACTIVE FUNCTION

The active function returns "true" if an assignment was successful or "false" if the resource is unavailable. Other errors are reported by active_fnc_error_. The -long control argument is not allowed. Use list_resources to obtain the name of the assigned device(s).

EXAMPLES

In the first example, the user issues assign_resource with the tape_drive keyword and -model. The system responds with the name of the assigned device.

```
! ar tape_drive -model 500

   Device tape_04 assigned
```

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

```
! ar tape_drive -device tape_05 -long

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

In the last example, the user issues assign_resource as an active function with tape_drive and -model. The system returns "true" if a model 610 tape drive was assigned, "false" if not.

```
! format_line [ar tape_drive -model 610]

   true
```

Name: attach_audit, ata

SYNTAX AS A COMMAND

```
ata {old_switch {new_switch}} {-control_args}
```

FUNCTION

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

ARGUMENTS

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

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

CONTROL ARGUMENTS

-modes STR
set the modes on the switch being audited using STR as the mode string.

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

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

NOTES

If used with no arguments, attach_audit sets up auditing for the user_i/o I/O switch with input and output audited and editing on. Auditing of old_switch is done by moving the attachment of old_switch to new_switch and then attaching old_switch to new_switch via audit_. See the audit_ I/O module and the detach_audit command for more information.

LIST OF AUDITING REQUESTS

A three-character sequence is used to make an auditing request: the audit trigger character ("!" by default), followed by the specific request character, followed by a newline. An auditing request can either be alone on a line or have text preceding it on the same line. When an unrecognized request is given, the entire line is treated as a regular input line (with no special processing).

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

!?
prints a brief description of available auditing requests.

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

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

!a
expands abbreviations in the input line (see the abbrev command).

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

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

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

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

NOTES ON AUDIT FILE

The audit file, by default, has the pathname:

```
>udd>Project_id>Person_id>date.audit
```

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

```
[home_dir]>[date].audit
```

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

Audit files contain binary information. Use the display_audit_file command to print the contents of audit files.

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

EL edit line

IC input characters

IL input line

OC output characters

TC trace of control operations

TM trace of modes operations

NOTES ON AUDIT EDITOR

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

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

LIST OF EDITING REQUESTS

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

FAMILIAR REQUESTS

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

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

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

`=`
prints the current entry number. This value is dependent on the current default search tag. If the default search tag changes, the current entry may also change.

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

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

SPECIAL REQUESTS

expand, .expand
expands abbreviations in the edit buffer (see the abbrev command).

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

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

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

r[STR], .r[STR]
quits the editor and returns the string STR to the I/O module being audited. If STR is not specified, the r request quits the editor and returns the edit buffer.

n, .n
returns a newline character.

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

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

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

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

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

NOTES

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

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

If execution of a request should fail for any reason, the processing of that request line is aborted, you are informed of the failure and a new request is prompted for. Note that this means you are left in the editor when a problem is encountered executing a request line associated with an E audit request.

The audit editor may be entered recursively, and each level of the editor has its own memory for the last returned line from its level.

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

EXAMPLES

To set up with a default audit file in your home_dir:

```
! ata
```

To set up with an audit file in the process_dir:

```
! ata -pn [pd]>my_audit_file
```

To set the audit file to be a circular file of 5 records:

```
! io modes user_i/o audit_file_size=5
```

To re-execute the last use of the pl1 command:

```
! </^pl1/r!E
```

To execute the above command line again:

```
! !r!E
```

In the example given below, there has been such extensive use of the erase character that you may want to see it displayed. In order to verify the input line given, it can be replayed by using the !r request. The ! at the beginning of the line indicates lines typed by you.

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

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

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

does not correct the error, but returns:

```
   stty -pmodes redmodes red
```

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

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

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

An alternative procedure is the following:

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

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

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

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

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

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

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

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

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

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

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

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

Name: attach_lv, alv

SYNTAX AS A COMMAND

```
alv volume_name
```

FUNCTION

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

ARGUMENTS

volume_name
    specifies the name of the volume to be attached.

ACCESS REQUIRED

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

NOTES

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

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

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

Name: basic

SYNTAX AS A COMMAND

```
basic path {-control_arg}
```

FUNCTION

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

ARGUMENTS

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

CONTROL ARGUMENTS

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

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

NOTES

The -compile and -time control arguments are incompatible.

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

For a description of the BASIC language on the Multics system, consult the *Multics BASIC manual* (AM82).

For information on using the FAST subsystem to compile BASIC source code, refer to the *Multics FAST Subsystem Users' Guide* (AU25).

Name: before, be

SYNTAX AS A COMMAND

```
be STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[be STRA STRB]
```

FUNCTION

returns the string preceding the first occurrence of STRB in STRA. If STRB does not occur in STRA, the entire string STRA is returned.

EXAMPLES

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

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

Name: before_journal_status, bjst

SYNTAX AS A COMMAND

```
bjst {PATHS} {-control_args}
```

FUNCTION

displays status information for before journals that you have access to open. This command is part of the command level interface to Multics data management (DM) (see the Programmer's Reference Manual).

ARGUMENTS

PATHS
: are the relative pathnames of before journals for which status is desired. If you supply no pathnames, status information for all journals in use in the process is displayed. If you don't give the .bj suffix, it is assumed.

CONTROL ARGUMENTS

-all
: displays the status of all journals active in the current invocation of the data management system (DMS) that you have access to open.

-brief, -bf
: displays the pathname, unique identifier, usage state or activity, control interval size, and control intervals in the before journal for each journal specified that is either in use or not in use (see "Examples").

-long, -lg
: for each journal specified that is in use, displays, besides the above information, the disposition of control intervals in use, i.e., if they are buffered, put, flushed, or on disk; the last time a control interval was queued or written; the time the header was updated; the last record id; the status of images not yet written on disk or not being flushed; and the number of users and transactions using the journal. For each journal specified that is not in use, displays, besides the information given by -brief, the time the header was updated. (See "Examples.")

NOTES

If you give neither -brief nor -long, the command yields the information supplied by -brief plus the disposition of control intervals in use at the time of the request if the journal(s) specified is in use.

EXAMPLES

The example below requests the status, in long form, of the system_low system default before journal, which is in use.

```
!  bjst >site>dm>system_low>system_default -lg

pathname:                       >site>Data_Management>system_low>system_default.bj
journal uid:                    132233107561
activity:                       in use
control interval size:          4096 bytes
control intervals:              4000
control intervals used:         86
last control interval
          buffered:             86
          put:                  86
          flushed:              86
          on disk:              86
time last control interval
          queued:               01/14/85  1104.9 est
          written:              01/14/85  1104.9 est
time header updated:            01/14/85  1104.9 est
last record id:                 000001260013
images not on disk:             0
images being flushed:           0
users:                          2
transactions:                   1
```

where:

pathname
: is the pathname of the before journal.

journal uid
: is the octal unique identifier of the before journal.

activity
: is "in use" if a process currently has the before journal open, "not in use" otherwise.

control interval size
: is the size of each control interval in the before journal, in bytes. Currently 4096 bytes is the only supported size.

control intervals
: is the number of control intervals in the before journal.

before_journal_status

control intervals used
is the number of control intervals in the before journal containing before images still needed to roll back modifications made by a transaction. Images that are not needed include those that have already been used in a complete rollback and those for a transaction that has ended.

last control interval buffered
indicates the last control interval put in a special buffer used for before journals.

last control interval put
indicates the last control interval put into the before journal.

last control interval flushed
indicates the last control interval flushed to disk.

last control interval on disk
indicates the last control interval safely on disk.

time last control interval queued
is the last time a before image was put in the before journal.

time last control interval written
is the last time a control interval was written to disk.

time header updated
is the last time the header of the before journal was written.

last record id
is the address of the last before image in the journal.

images not on disk
is the number of images not written to disk yet.

images being flushed
is the number of before images for which a flush from memory to disk has been requested.

users
is the number of users with openings.

transactions
is the number of active transactions in the before journal.

The example below requests the status, in long form, of the system_low system default before journal, which is not in use.

```
! bjst >site>dm>system_low>system_default -lg

pathname:                    >site>dm>system_default.bj
journal uid:                 127120202215
activity:                    not in use
control interval size:       4096 bytes
control intervals:           4000
time header updated:         08/26/84  1228.6 edt
```

Name: binary, bin

SYNTAX AS A COMMAND

```
bin values
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bin values]
```

FUNCTION

returns one or more values in binary.

ARGUMENTS

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

EXAMPLES

```
! string [binary 657.4o]
  110100111.1

! string [bin abcu]
  110000100110001000110001 1
```

Name: bind, bd

SYNTAX AS A COMMAND

```
bd path_specs {-control_args}
```

FUNCTION

produces a single bound object segment from one or more unbound object segments, which are called the components of the bound segment. You can use archive segments or unarchived segments to specify pathnames of object components.

ARGUMENTS

path_specs
can be one or more of the following logically concatenated in a left-to-right order to produce a single sequence of input component object segments.

-archive PATHs, -ac PATHs
indicates that each PATH is the pathname of an archive segment containing one or more object segments. If the .archive suffix does not exist, it is assumed. (All arguments following -archive but preceding the next control argument are considered to be pathnames.)

-segment PATHs, -sm PATHs
indicates that each PATH is the pathname of a stand-alone segment. The pathname is tried as given, i.e., no suffixes are assumed. (All arguments following -segment but preceding the next control argument are considered to be pathnames.)

PATHs
functions exactly as -archive PATHs.

CONTROL ARGUMENTS

-bindfile path, -bdf path
specifies the name (not pathname) of the bindfile to be used to control the binding process. The suffix .bind is assumed. (See "Notes on Bindfile" below.)

-brief, -bf
suppresses printing of warning messages.

-force_order, -fco
is equivalent to including a Force_Order statement in the bindfile. Since the need to use Force_Order is often temporary and caused by update archives that have had components deleted, this is preferable to using the Force_Order statement because you need only use it while the temporary condition exists.

-force_update path_specs, -fud path_specs
is similar in function to -update except that the path_specs (see the path_specs argument above) specified following -force_update need not exist. Any path that exists is treated the same way as for -update and any that doesn't is simply ignored. This is useful for constructing abbreviations used for binding objects that may or may not have update paths in various locations.

-list, -ls
produces a listing segment whose name is derived from the name of the bound object segment plus a suffix of list. The listing segment is generated to dprint; it contains the bound segment's bind control segment (see "Notes on Bindfile"), its bind map, and that information from the bound object segment printed by the print_link_info command. You can't invoke -list with -map. In the absence of -list or -map, no listing segment is generated.

-map
produces a listing segment (with the suffixes list and map) that contains only the bind map information. It is incompatible with -list. In the absence of -list or -map, no listing segment is generated.

-update path_specs, -ud path_specs
indicates that the following list of path_specs (see the path_specs argument above) specifies update rather than input object segments. The update object segments are matched against the input object segments by object segment name. Matching update object segments replace the corresponding input object segments; unmatched ones are appended to the sequence of input object segments. If several update object segments have the same name, only the last one encountered is bound into the bound segment.

NOTES

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

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

NOTES ON OUTPUT

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

NOTES ON BINDFILE SELECTION

As the binder is examining the archive components and loose segments, it is also looking for a bindfile. Any segment whose name ends with the suffix "bind" is considered a bindfile. If you specify -bindfile, only bindfiles by that name are considered and the last one by that name is selected; otherwise the first bindfile found among the input segments and all bindfiles among the update segments are considered and the last one is selected. If more than one bindfile is found among the input segments, the second through last are ignored and generate a warning.

NOTES ON BINDFILE

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

LIST OF MASTER KEYWORDS

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

Order
: the parameters are a list of objectnames in the desired binding order. In the absence of an order statement, binding is done in the order of the input sequence. If an Order statement is present in the bindfile, every object segment being bound must be mentioned in its parameter list.

Force_Order
: same as Order except that the list of parameters can be a subset of the input sequence, allowing the archive segments to contain additional segments that are not to be bound (e.g., source programs). However, the parameter list must include all segments mentioned in objectname statements.

Partial_Order
same as Order except that the list of parameters can be a subset of the input sequence; the named objectnames are placed in the bound output segment in the order specified and the remaining objects are placed after those named, in the order of the input sequence.

Ignore
the parameters can be a subset of the input sequence, indicating objects not to be included in the bound output segment. The ignored objects are still mentioned in the bound segment's source map.

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

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

No_Table
does not require parameters. It omits from the bound segment the symbol tables from all the component symbol sections containing symbol tables. If you don't give this keyword, all symbol tables are kept.

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

The Order, Force_Order, and Partial_Order statements are mutually contradictory; only one of these can be present in any bindfile.

If you supply no bindfile, the binder assumes default parameters corresponding to the following:

```
Objectname: segment name of the first input archive file.
Global: retain; /*regenerate all definitions*/
```

LIST OF NORMAL KEYWORDS

objectname
the single parameter is the name of a component object as it appears in the archive segment. The objectname statement indicates that all following normal statements (up to but not including the next objectname statement) pertain to the component object whose name is the parameter of the objectname statement.

synonym
the parameters are symbolic segment names declared to be synonymous to the component object's objectname. When b is declared to be a synonym for a, references (in the bound components) of the form b or b$x (any x) are resolved during binding by searching for a definition of b or x in component a. Give the synonym instruction if such references are to be prelinked. The synonym instruction also affects dynamic linking so that if b is a reference name for the bound segment, then references of the form b or b$x are resolved by searching component a. In this case the synonym instruction may reduce the cost of dynamic linking, and it avoids possible ambiguities when two components contain definitions for the symbol b. State explicitly in a synonym statement all the normally used segment names of a component object.

For example, the create and create_dir commands are implemented in one procedure, and both have abbreviations; thus a bindfile for the bound segment in which this procedure resides contains

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

Failure to state segment names results in inefficient linker performance.

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

delete
the parameters are the names of entrypoints defined within the component object segment that you don't wish to be retained as entrypoints of the new bound segment.

no_link
the parameters are the names of entrypoints that are not to be prelinked during binding. This statement implies that the specified names be retained.

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

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

table
does not require parameters. It retains the symbol table for the component and is needed to override the No_Table master keyword.

LIST OF BINDFILE DELIMITERS

: keyword delimiter used to identify a keyword followed by one or more parameters. A keyword that is followed by no parameters is delimited by a statement delimiter.
; statement delimiter
, parameter delimiter. The last parameter is delimited by a statement delimiter.
/* begin comment.
*/ end comment.

NOTES ON ERROR MESSAGES

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

```
ERROR J SEVERITY 3 IN LINE N
...
```

or

```
WARNING J IN LINE N
```

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

The message

```
"bind:  Fatal error has occurred; binding unsuccessful."
```

indicates that it was impossible for the binder to produce an executable object segment because of errors detected during binding.

EXAMPLES

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

```
Objectname:  bound_debug;
Global:      delete;     /*delete all old definitions*/
Addname;                 /*add names debug, db, list_arg_ and gr_print
                           to bound segment bound_debug*/
objectname:  debug;
synonym:     db;         /*indicate db is synonymous to debug*/
retain:      debug,
             db;         /*retain entrynames debug$debug and debug$db*/

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

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

The following illustrates other uses of the bindfile:

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

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


No_Table;                /*omit all symbol tables*/

objectname:  reset;
retain:      reset;

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

Name: bj_mgr_call, bjmc

SYNTAX AS A COMMAND

```
bjmc key {path} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc key {path} {-control_args}]
```

FUNCTION

enables you to manipulate before journals in your process by calling before_journal_manager_ entry points from command level. This command is part of the command level interface to Multics data management (DM) (see the Programmer's Reference Manual).

ARGUMENTS

key
: designates the before journal manager operation to be performed. See "List of Operations" below for a description of each operation, its command and active function syntax lines, and specific application.

path
: specifies the absolute or relative pathname of the before journals being manipulated (required for all key operations except get_default_path). Give -pathname (-pn) PATH with pathnames constructed with leading minus signs to distinguish them from control arguments. If you supply no .bj suffix, it is assumed.

CONTROL ARGUMENTS

can be one or more control arguments, depending on the particular operation.

LIST OF OPERATIONS

Each operation is described in the general format of a command/active function. Where appropriate, notes and examples are included for clarity.

Operation: close, cl

SYNTAX AS A COMMAND

```
bjmc cl path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc cl path]
```

FUNCTION

closes the before journal specified by path. Separate pathnames by spaces if multiple journals are to be closed. Specifically close by name each journal opened in the process. The active function returns true if the journals were closed successfully, false otherwise.

ARGUMENTS

path
: is the absolute or relative pathname of before journals to be closed. If you supply no .bj suffix, it is assumed.

NOTES

If a before journal being closed by this operation is the default journal, the last journal opened in the process becomes the default.

Operation: closed

SYNTAX AS A COMMAND

```
bjmc closed path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc closed path]
```

FUNCTION

returns true if the before journal specified by path is not open in your process, false otherwise.

ARGUMENTS

path
: is the absolute or relative pathname of a before journal. If you don't give the .bj suffix, it is assumed.

bj_mgr_call

Operation: create, cr

SYNTAX AS A COMMAND

```
bjmc cr path {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc cr path {-control_args}]
```

FUNCTION

creates the before journal specified by path. The active function returns true if the before journal is created successfully, false otherwise.

ARGUMENTS

path
is the absolute or relative pathname of the before journals to be created. If you supply no .bj suffix, it is assumed.

CONTROL ARGUMENTS

-length N, -ln N
specifies the size of the before journal, where N is the number of 4096-byte control intervals. Once established, you can't alter a journal's size. (Default: if you specify no value at the time of creation, the size is 64 control intervals)

-transaction_storage_limit N, -tsl N
specifies the maximum number of bytes a single transaction can use in the before journal (Default: the entire journal)

NOTES

Before journals are extended entry types; you can delete them using the delete command. You can only delete before journals if they are not required for recovery.

Operation: get_default_path, gdp

SYNTAX AS A COMMAND

```
bjmc gdp
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc gdp]
```

FUNCTION

returns the pathname of the process's default before journal.

Operation: open, o

SYNTAX AS A COMMAND

```
bjmc o path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc o path]
```

FUNCTION

opens the before journal specified by path. The active function returns true if the journal is opened successfully, false otherwise.

ARGUMENTS

path
: is the absolute or relative pathname of before journals to be opened in your process. If you supply no .bj suffix, it is assumed.

NOTES

If no journal has been specifically designated as the default (see the set_default_path operation) for your process, the last before journal opened in the process becomes the default. If no journal is opened in your process when a transaction is started, the system before journal is opened and used as the default.

Operation: opened

SYNTAX AS A COMMAND

```
bjmc opened path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc opened path]
```

FUNCTION

returns true if the before journal specified by path is opened in your process, false otherwise.

ARGUMENTS

path
is the absolute or relative pathname of a before journal. If you supply no .bj suffix, it is assumed.

Operation: set_default_path, sdp

SYNTAX AS A COMMAND

```
bjmc sdp path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[bjmc sdp path]
```

FUNCTION

sets the default before journal for the process to the specified pathname. The active function returns true if the pathname is successfully set, false otherwise.

ARGUMENTS

path
is the absolute or relative pathname of the before journal to be used as the default by your process. If you supply no .bj suffix, it is assumed.

NOTES

If no default before journal is set for your process, the last journal opened in the process is used as the default (see the open operation). If no before journal is open in the process when a transaction is started, the system before journal is opened and used as the default.

Name: bool

SYNTAX AS A COMMAND

```
bool B1 B2 B3
```

SYNTAX AS AN ACTIVE FUNCTION

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

FUNCTION

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

ARGUMENTS

B1, B2, and B3
are bit strings entered as 0 and 1 characters.

NOTES

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

B3 must be four bits long. It causes the following logical operations to be performed on B1 and B2.

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

EXAMPLES

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

Name: branches

SYNTAX AS A COMMAND

```
branches star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

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

FUNCTION

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

ARGUMENTS

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

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames. (Default: to return entrynames)

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

byte

Name: byte

SYNTAX AS A COMMAND

```
byte N
```

SYNTAX AS AN ACTIVE FUNCTION

```
[byte N]
```

This page intentionally left blank.

FUNCTION

Prints or returns the character in a specified position in the Multics ASCII collating sequence.

ARGUMENTS

N
is the zero-origin position of the desired character in the collating sequence. If the number ends with the character "o", it is interpreted as an octal number; otherwise it is decimal.

Name: calc

SYNTAX AS A COMMAND

```
calc {expression}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[calc expression]
```

FUNCTION

provides you 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. You 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.

You 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 you to command level. The order of evaluation is as follows:

expressions within parentheses

function references

prefix +, prefix -

**

*, /

+, -

For example, if you type:

```
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: calendar

SYNTAX AS A COMMAND

```
calendar {paths} {-control_args}
```

FUNCTION

prints a calendar page for one month.

ARGUMENTS

paths
: are the pathnames of segments that contain a list of events in the form of text to be inserted into the calendar (see "Notes" below).

CONTROL ARGUMENTS

-box_height N, -bht N
: sets the number of lines available for notes in each box to N, an integer number. The default for N is 7 if you don't use -box_height. If N is less than 7, the marginal calendars for the previous and following months cannot be printed.

-date DT, -dt DT
: identifies which month is printed. If you supply no -date, the current month is printed. (See Section 1 for a description of valid DT values.)

-fiscal_week, -fw
: labels boxes with fiscal week numbers. This command assumes that each fiscal week begins on Monday and ends on Sunday and fiscal week 1 is the first full week of the calendar year; therefore fiscal week 1 of 1984 begins on Monday, January 2.

-force, -fc
: prints a calendar regardless of errors in the input files.

-julian, -jul
: places in the lower left corner of each day box the number of days of the year that have passed, including the current day, and in the lower right the number of days remaining in the year after the current day. This control argument effectively reduces the number of lines in each box that are available for notes.

-stop, -sp
: waits until you type a single newline character before printing the calendar and after printing it.

-wait, -wt
: waits for a single newline character from you before printing the calendar. Characters typed before the newline are ignored. This allows you to position the paper and to add simple top-of-form notes.

NOTES ON OUTPUT

Each box for a calendar day is 16 characters wide. Its height is determined by the -box_height control argument; but if that control argument is not used, each box is 7 lines high. Each box in the calendar contains the number of the day of the month; other information can also appear in the box, at your option. The month preceding the specified month and the month following it are also printed.

NOTES ON INPUT

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

```
opcode,dtfield,...,dtfield,text
```

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

NOTES

If the command finds errors in its arguments it reports the errors and does not print a calendar. If it finds errors in an input file, it stops after all errors have been reported, unless you give -force to indicate that the calendar should be printed in spite of errors.

Users can insert several lines of text for any date. This is accomplished by supplying multiple date or rel entries for the desired date (see Washington's birthday under "Examples" below). The number of lines available is controlled by the -box_height control argument.

THE DATE OPERATION CODE

The date operation code has the following syntax:

```
date,DT,TEXT
```

The first field is the date operation code itself, date.

The second field, DT, is any date acceptable to the convert_date_to_binary_ subroutine. This date is converted relative to the day before the beginning of the month, so that "Mon" is the first Monday in the month, etc.

The third field, TEXT, is arbitrary text. Up to 16 characters are inserted into the calendar in the appropriate place, if the date specified by DT falls in the month for which the calendar is being printed.

An example of the date operation code:

```
date,07/04,Independence Day
```

THE REL OPERATION CODE

The rel operation code allows a note to be inserted for a day that is calculated relative to the beginning of a month. It is useful for events that are defined to occur, for example, on the first Tuesday after the first Monday of a month. Its syntax is as follows:

```
rel,MONTHNO,RELDT1,RELDT2,TEXT
```

The first field is the rel operation code itself, rel.

The second field, MONTHNO, is a one or two digit number, or 0, -1, or +1. If it is a simple number (1 through 12 are accepted), it indicates the month from which the event is to be calculated. If it is 0 it indicates that the target date is to be calculated relative to the month for which the calendar is being printed. A MONTHNO of -1 indicates that the date is calculated relative to the month before the printed month; +1 indicates the month after the printed month.

The third field, RELDT1, is a date (acceptable to the convert_date_to_binary_ subroutine) that is converted relative to the month specified by the MONTHNO of the second field. More precisely, it is converted relative to the day before the beginning of the specified month.

The fourth field, RELDT2, is a date (acceptable to the convert_date_to_binary_ subroutine) that is converted relative to the date indicated by the RELDT1 of the third field. It specifies the date selected for the insertion of the TEXT.

The fifth field, TEXT, is arbitrary text to be inserted in the calendar if the date ultimately calculated from MONTHNO, RELDT1, and RELDT2 falls in the month for which the calendar is being printed.

An example of the rel operation code

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

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

THE REPEAT OPERATION CODE

The repeat operation code inserts a note into the boxes for several days that are separated by a constant interval of time. Its purpose is to enter notations for a series of repeating events, such as regular meetings. The syntax is as follows:

```
repeat,STARTDT,END_OR_COUNT,INTERVAL,TEXT
```

The first field is the repeat operation code itself, repeat.

The second field, STARTDT, is the date on which the series of events starts. It is a date acceptable to the convert_date_to_binary_ subroutine, or 0. The date is converted relative to the day before the beginning of the month to be printed. A STARTDT of 0 indicates that the series starts on the first day of the printed month.

The third field, END_OR_COUNT, is the end date (last day on which the event may potentially occur), or 0, or a count of the number of events in the series. A date (acceptable to the convert_date_to_binary_ subroutine) is converted relative to the day before the beginning of the month to be printed. An END_OR_COUNT of 0 indicates that the series continues throughout the entire month being printed. An integer number, rather than a date or 0, gives the number of events in the series.

The fourth field, INTERVAL, is any offset acceptable to the convert_date_to_binary_ subroutine, or 0. If the INTERVAL is an offset, it is truncated to an integral number of days; but if it is less than one day, it is treated as if it were 1 day. Thus, 75 hours gives an interval of 3 days, 5 hours gives an interval of 1day. If the INTERVAL is given as 0, it indicates an interval of 1 day.

The fifth field, TEXT, is arbritrary text to be placed in the box of each day in the series.

Examples of the repeat operation code:

```
repeat,04/01/80,10,1week,Karate Lesson
```

places the text "Karate Lesson" in the calendar boxes for April 1, 8, 15, etc., through June 3, 1980.

```
repeat,08/04/80,08/08/80,1day,Hang Glider Meet
```

places the text "Hang Glider Meet" in every calendar box from August 4, 1980 to August 8, 1980.

THE EASTER OPERATION CODE

The easter operation code calculates the date for Easter and inserts its text in that date if it falls in the printed month. The syntax is:

```
easter,TEXT
```

There are only two fields, the operation code, easter, and the text to be placed in the box. An example:

```
easter,Family reunion
```

THE RENAME OPERATION CODE

The rename operation code allows you to change the names of days or months. Its syntax is

```
rename,OLDNAME,NEWNAME
```

The OLDNAME field gives the name of a day or month to be changed. If the name of that day or month was previously changed in the current invocation of the command, OLDNAME must be the current name. The NEWNAME field gives the name to replace the OLDNAME. For example, rename,Monday,Lundi.

EXAMPLES

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

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

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

```
* personal calendar info
*
date,05/21,Kirsten's Birthday
date,11/16,Mom's Birthday
repeat,Saturday,0,1week,Racquetball Fred
   .
   .
   .
```

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

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

This page intentionally left blank.

Name: calendar_clock

SYNTAX AS A COMMAND

```
calendar_clock {time_string} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[calendar_clock {time_string} {-control_args}]
```

FUNCTION

returns the complete clock value from the four-digit year down through the microsecond in a sequence that allows direct comparison, e.g., "1982-12-23_18:06:30.421857_gmt_Thu". The format string to produce this is "^9999yc-^my-^dm_^Hd:^MH:^99.(6)9UM_^za_^da".

ARGUMENTS

time_string
: indicates the date about which information is desired. If you supply no time_string, the current date is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-language STR, -lang STR
: STR specifies the language in which month names, day names, and zone names are to be expressed. (Default: English)

-zone STR
: STR specifies the zone to be used to express the result. (Default: Greenwich Mean Time, GMT)

NOTES

Use the print_time_defaults command to display the default language and zone. Use the display_time_info command to display a list of all acceptable language and zone values.

Name: cancel_abs_request, car

SYNTAX AS A COMMAND

```
car request_identifiers {-control_args}
```

FUNCTION

allows you to delete a request for an absentee computation that is no longer needed.

ARGUMENTS
you can chose request_identifiers from the following:

path
is the full or relative pathname for the absentee input segment of requests to be canceled. You can use the star convention.

-entry STR, -et STR
identifies requests to be canceled by STR, the entryname portion of the absentee input segment pathname. You can use the star convention.

-id ID
identifies one or more requests to be canceled by request identifier. You can give this identifier 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. It is incompatible with -queue.

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

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

-queue N, -q N
specifies that queue N of the request type contains the requests to be canceled, where N is a decimal integer specifying the number of the queue. If you omit it, all the queues are searched.

-sender STR
specifies that only requests from sender STR should be canceled. You must also give one or more request identifiers. In most cases the sender is an RJE station identifier.

-user User_id
specifies the name of the submitter of the request to be canceled if it is not the same as the group identifier of the process. You can give the User_id as 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

You need o extended access to the queue to cancel your own request and r and d extended access to cancel somebody else's request.

NOTES

If you supply -id, only one path or -entry is allowed. If you give -id in addition to a path or -entry, they must match the same request. If you provide any path or -entry STR request identifiers, only one -id ID request identifier is accepted and it must match any requests selected by path or entryname.

You can supply multiple -id ID identifiers in a single command invocation only if you give no path or entry request identifiers. 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 canceled. 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 canceled 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>CPEBach>dump>translate
```

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

The command line

```
! car >udd>Demo>LNTolstoy>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>LNTolstoy>doc directory.

Name: cancel_cobol_program, ccp

SYNTAX AS A COMMAND

```
ccp names {-control_arg}
```

FUNCTION

causes one or more programs in the current COBOL run unit to be canceled.

ARGUMENTS

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

CONTROL ARGUMENTS

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

NOTES

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

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

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

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

Name: cancel_daemon_request, cdr

SYNTAX AS A COMMAND

```
cdr request_identifiers {-control_args}
```

FUNCTION

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

ARGUMENTS

request_identifiers can be chosen from the following:

path
identifies a request to be canceled 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 canceled 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 canceled by request identifier. You can use this identifier 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. It is incompatible with -queue.

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

-queue N, -q N
specifies that queue N of the request type contains the requests to be canceled, where N is a decimal integer specifying the number of the queue. If you omit it, all the queues are searched.

-request_type STR, -rqt STR
indicates that the request to be canceled is to be found in the queue for the request type identified by STR. If you don't supply -request_type, the default is "printer." The print_request_types command lists the request types.

-user User_id
specifies the name of the submitter of the request to be canceled if it is not the same as 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

You need o extended access to the queue to cancel your own request and r and d extended access to cancel somebody else's request.

NOTES

If you supply -id, only one path or -entry is allowed. If you give -id in addition to a path or -entry, they must match the same request. If you provide any path or -entry STR request identifiers, only one -id ID request identifier is accepted and it must match any requests selected by path or entryname.

You can specify multiple -id ID identifiers in a single command invocation only if you give no path or -entry request identifiers.

When you don't use star names and a single request identifier matches more than one request in the queue(s) searched, none of the requests are canceled; 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. You are 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 dprint and enter_output_request commands) are ignored by the system.

See the dprint, dpunch, and enter_output_request commands.

EXAMPLES

The command line

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

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

The command line

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

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

The command line

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

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

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 canceled 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 canceled 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 canceled by the request identifier. You can use this identifier 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. It is incompatible with -queue.

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

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

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

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

-queue N, -q N
specifies that queue N of the request type contains the requests to be canceled, where N is a decimal integer specifying the number of the queue. If you omit it, all the queues are searched.

-request_type STR, -rqt STR
indicates that the requests to be to be canceled is to be found in the queue for the request type identified by STR (see "Notes").

-user User_id
specifies the name of the submitter of the request to be canceled if it is not the same as 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

You need o extended access to the queue to cancel your own request and r and d extended access to cancel somebody else's request.

NOTES

You can specify multiple -id ID identifiers in a single command invocation only if you give no path or -entry request identifiers.

If you give any path or -entry STR request identifiers, only one -id ID request identifier is accepted and it must match any requests selected by path or entryname.

When you don't use star names and a single request identifier matches more than one request in the queue(s) searched, none of the requests are canceled; 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. You are 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 dprint and enter_output_request commands) are ignored by the system.

The -print, -punch, -plot, and -request_type control arguments are mutually exclusive. If you supply none, then cor searches the default request type used by eor -print (as displayed by the print_request_types command).

See the dprint, dpunch, and enter_output_request commands.

Name: cancel_resource, cnr

SYNTAX AS A COMMAND

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

FUNCTION

cancels reservations made with reserve_resource using the reservation identifier obtainable from list_resources.

ARGUMENTS

reservation_id
must be present and is the reservation identifier of the reservation to be canceled.

CONTROL ARGUMENTS

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

Name: cancel_retrieval_request, crr

SYNTAX AS A COMMAND

```
crr request_identifiers {-control_args}
```

FUNCTION

allows you to delete a request for a volume retrieval that is no longer needed.

ARGUMENTS
request_identifiers can be chosen from the following:

path
identifies a retrieval request to be canceled by the full or relative pathname of the segment or subtree. The star convention is allowed.

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

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

CONTROL ARGUMENTS

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

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

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

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

ACCESS REQUIRED

You must have o extended access to the queue to cancel your own requests. You must have r and d extended access to cancel a request entered by another user.

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.

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 canceled. However, a message is printed telling how many matching requests there are.

EXAMPLES

The command line:

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

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

Name: canonicalize, canon

SYNTAX AS A COMMAND

```
canon path1 {path2} {-control_args}
```

FUNCTION

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

ARGUMENTS

path1
is the pathname of the input segment.

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

CONTROL ARGUMENTS

-force, -fc
overwrites the output file without querying.

-input_tabs -every X, -itabs -ev X
replaces tabs with the appropriate number of spaces, assuming that tabs are at 1+n*X (where n = 1, 2, 3,...). (Default: every 10)

-input_tabs n1,n2,...,n20, -itabs n1,n2,...,n20
replaces tabs with the appropriate number of spaces, assuming that tab stops are as specified. If tabs are found in the input segment beyond the stops specified, every 10 is assumed.

-no_force, -nfc
queries before overwriting an existing segment. (Default)

-no_output_tabs, -notabs
does not place tabs in the output. (Default)

-output_tabs -every X, -otabs -ev X
inserts tabs at 1+n*X.

-output_tabs n1,n2,...,n20, -otabs n1,n2,...,n20
inserts tabs at the tab stops specified. You can give up to 20 tab stops. No spaces are allowed in the list.

NOTES

The command ensures that all characters in a print position are sorted in the proper order and removes all ASCII carriage-return (015) characters. If you use -otabs, canonicalize replaces blank spaces with the appropriate tab stops. If you use -notabs, canonicalize replaces horizontal tab stops with the correct number of blank spaces.

EXAMPLES

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

```
canon my_prog -otabs 7,21,35
```

To canonicalize the same segment, rename it to "new_prog", and set up tab stops at 15-space intervals, type:

```
canon my_prog new_prog -otabs -ev 15
```

To canonicalize the segment "new_prog", which already contains tab stops that are now to be replaced with blank spaces, type:

```
canon new_prog -itabs -ev 15
```

This page intentionally left blank.

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

```
canon old_prog
```

Name: canonicalize_mailbox

SYNTAX AS A COMMAND

```
canonicalize_mailbox path {-control_args}
```

FUNCTION

converts the messages in a mailbox into their canonical form as defined by the MR10.2 mail system.

ARGUMENTS

path
is the pathname of the mailbox whose messages are to be converted. The suffix "mbx" is supplied if needed. The star convention is accepted.

CONTROL ARGUMENTS

-force, -fc
temporarily alters your access to the mailbox when necessary to convert the messages in the mailbox (see "Access Required" below).

-no_force, -nfc
never alters your access to the mailbox. (Default)

-privilege, -priv
uses privileges to bypass the restrictions on the canonicalization process introduced by the Access Isolation Mechanism (see "Notes on AIM" below).

-no_privilege, -npriv
does not use privileges. (Default)

ACCESS REQUIRED

You must have status (s), modify (m), and append (a) access to the directory containing the mailbox. In addition, if -force is not specified, you must have read (r), add (a), and delete (d) extended access to the mailbox itself.

If -privilege is specified, you must have execute (e) access to the system_privilege_ gate. In addition, your maximum process authorization must be system_high.

NOTES

The canonical form of a message is similar to the text of the printed representation of that message when formatted using the default formatting modes.

Messages stored in mailboxes prior to MR10.2 were not stored in their canonical form. Unless these messages are converted to their canonical form, subsystems, such as read_mail, that provide the option to select messages by content are required to format the messages during the search. This formatting while searching severely affects the performance of the selection process.

Messages stored in a mailbox after the installation of MR10.2 are stored in canonical form and will not affect the performance of context searches.

This command needs to be used only once on any given mailbox. Preferably, use this command on any large mailbox (e.g., logboxes or saveboxes containing more than fifty messages).

This command first creates a new mailbox in the same directory as the mailbox whose messages are to be converted. The messages are then read from the original message, canonicalized, and stored in the new mailbox. Next, the names, access control list, maximum length, and safety switch setting of the original mailbox are moved to the new mailbox. Finally, the original mailbox is deleted.

If the directory containing the original mailbox has insufficient quota for the new mailbox, the original mailbox is left intact and an error message is printed.

The record of any process accepting messages on the original mailbox is lost during the canonicalization process. You must reissue the accept_messages command, if needed, for each mailbox that is processed by canonicalize_mailbox. Because of the nature of accept_messages, the explicit pathname of your default mailbox (>udd>Project_id>Person_id>Person_id.mbx) must be supplied if that mailbox is canonicalized in order to reaccept messages.

After a mailbox has been canonicalized, all messages in the mailbox are owned by the user who issued canonicalize_mailbox. If you originally placed a message in the mailbox, you cannot delete it if you have own (o) extended access on the mailbox. Normally this canonicalization's side effect is invisible for logboxes and saveboxes as only the creator of the logbox or savebox has access on that mailbox.

NOTES ON AIM

If the Access Isolation Mechanism (AIM) is in force at a site, several restrictions are placed on the use of canonicalize_mailbox. These restrictions are eliminated through the use of -privilege provided that you have sufficient access to it (see "Access Required" above).

To use canonicalize_mailbox, your process authorization must be equal to the access class of the directory containing the mailbox whose messages are to be converted.

Unlike ordinary segments, the access class of a mailbox may be greater than that of its containing directory. Each message in a mailbox has its own access class; the access class of the mailbox specifies the maximum access class for any message that may be added to the mailbox.

If the access class of a mailbox is greater than your process authorization, it may contain messages that you cannot read. If you were to canonicalize that mailbox, any such messages would be lost. Therefore, canonicalize_mailbox queries for permission to continue if asked to process a mailbox whose access class is greater than the process authorization. Unless you are quite certain that there are no upgraded messages in the mailbox, answer "no" to this query and ask a system administrator to canonicalize the mailbox using -privilege. The canonicalized mailbox created by this command has an access class equal to your maximum process authorization. If this access class is less than the previous one, a warning is issued. If the new access class is insufficient (e.g., a mailbox shared by several users with different maximum authorizations), ask a system administrator to reclassify the mailbox via the reclassify_sys_seg command.

Name: ceil

SYNTAX AS A COMMAND

```
ceil num
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ceil num]
```

FUNCTION

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

EXAMPLES

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

Name: change_default_wdir, cdwd

SYNTAX AS A COMMAND

```
cdwd {path}
```

FUNCTION

records a specified directory as your default working directory for the duration of the current process or until you issue the next change_default_wdir command.

ARGUMENTS

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

NOTES

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

The original default working directory is your home directory upon logging in.

See also the descriptions of the change_wdir and print_default_wdir commands.

Name: change_error_mode, cem

SYNTAX AS A COMMAND

```
cem {-control_args}
```

FUNCTION

controls the amount of information printed by the default handler for system conditions.

CONTROL ARGUMENTS

-brief, -bf
prints only the condition name.

-long, -lg
prints more complete messages. In particular, if the condition was detected in a

support procedure, the name of that procedure is printed besides the name of the most recent user procedure. If a segment that signaled a condition (or caused it to be signaled) is bound, both the offset relative to the base of the procedure and the offset relative to the base of the segment are printed.

NOTES

If you don't issue cem or issue it with no control arguments, you receive default length error messages. These messages are intermediate in length between the brief and long messages.

For a complete discussion of conditions and their handling see the Programmer's Reference Manual. See the reprint_error command for a similar, but more selective, capability.

Name: change_wdir, cwd

SYNTAX AS A COMMAND

```
cwd {path}
```

FUNCTION

changes your working directory to the directory specified as an argument.

ARGUMENTS

path
is the pathname of a directory. If you supply no path, the default working directory is assumed.

ACCESS REQUIRED

You must have s permission on the directory containing path.

NOTES

A working directory is a directory in which your activity is centered. Its pathname is remembered by the system so that you need not type the absolute pathname of segments inferior to that directory.

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

You don't need access to path to use change_wdir. However, once the working directory has been changed, you can proceed only according to your access to path. That is, to effectively use path as a working directory, you must have sma access permission for path. However, restricted uses are possible in accordance with the access mode attributes on the directory. For example, you must have at least status permission to list the directory.

See also the change_default_wdir and print_default_wdir commands.

Name: check_file_system_damage, cfsd

SYNTAX AS A COMMAND

```
cfsd {path} {-control_args}
```

FUNCTION

finds damaged segments and connection failures.

ARGUMENTS

path
: is a pathname specifying what is to be checked. It can be a star name, an absolute or relative pathname, or -working_dir (-wd). If you provide -subtree, path cannot be a star name (i.e., it must be a directory). If you give no path, you must supply -pathname.

CONTROL ARGUMENTS

-brief, -bf
: suppresses error messages about incorrect access to directories and no star name matches. (Default: to print these messages)

-call STR
: executes "STR path damaged" for each damaged segment and "STR path connection_failure" for each connection failure. STR is a command to be executed for each damaged segment. (Default: if you don't give -call, to print an error message for each damaged segment and each connection failure)

-depth N, -dh N
: looks only N directories down; if you supply it, -subtree is implied. (Default: to search downwards in all directories that are eligible for searching)

check_file_system_damage

-multisegment_file, -msf
checks components of MSFs. (Default)

-no_multisegment_file, -no_msf
does not check components of MSFs.

-pathname path, -pn path
specifies that the next argument is to be used as a pathname rather than as a control argument.

This page intentionally left blank.

-subtree, -subt
checks all segments in, and all directories below, the specified directory.

EXAMPLES

The command line

```
! cfsd >udd>Proj>CLDCarrol
```

checks the directory >udd>Proj>CLDCarrol to see if the directory is damaged.

The command line

```
! cfsd >udd>Proj>GJCasanova -subt -msf
```

starts at >udd>Proj>GJCasanova looking for damaged segments, directories, or MSF components, continuing down the directory hierarchy until the bottom is reached.

The command line

```
! cfsd >udd>Proj -dh 2
```

checks, only for damaged segments, the project directory >udd>Proj and the directories directly underneath it.

Name: check_iacl

SYNTAX AS A COMMAND

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

FUNCTION

lists segments whose access control lists (ACLs) disagree with the initial ACL for segments (for a description of ACLs, see the Programmer's Reference Manual).

ARGUMENTS

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

CONTROL ARGUMENTS

-all, -a
lists User_ids in a segment ACL excluded from the initial ACL and User_ids included in the initial ACL but omitted from a segment ACL. If you give no -all, only User_ids in addition to those in the initial ACL are listed.

-exclude User_id, -ex User_id
excludes the specified User_id from the comparison. You can supply up to 10 -exclude control arguments. You can use the star convention.

EXAMPLES

```
! check_iacl

   oldMap.com.runoff
     ACL added: rew James.Demo.*
     ACL added: rew Stevenson.Work.*

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

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 either the "No change" message or, if used with -call, the names of changed info segs. Don't use -brief with the 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. Don't use -call with the active function.

-date DT, -dt DT
uses the date DT instead of the date in your default value segment. The date in the value segment is not updated. See Section 1 for a description of valid DT values.

-long, -lg
prints the date-time-entry-modified as well as the segment name. Don't use -long with the active function.

-no_update, -nud
does not update the date in your value segment.

-pathname star_path, -pn star_path
checks all segments that match star_path, which is a pathname with a star name in the entryname portion. You can supply more than one -pathname. If you give none, the directories in the "info_segments" ("info_segs", "info") search list are used; **.info is used as the entryname.

-time_checked, -tmck
prints the date-time that is stored in your default value segment indicating from when checking of modified info segments occurs if -date is not given. This control argument is incompatible with all others when used with the active function. It does not update the time in your value segment when it is the only control argument.

NOTES

The first time you invoke cis, it sets the date in your default value segment. The value segment is created if one does not exist and is normally

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

but can be changed by the value_set_path command.

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

Zero-length info segments are not reported as being modified.

The cis active function returns entrynames of selected info segments separated by spaces. If you give -absolute_pathname, it returns full pathnames of info segments separated by spaces.

WARNING

Since the cis active function also sets the date in your 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: clock

SYNTAX AS A COMMAND

```
clock time_format {time_string} {control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[clock time_format {time_string} {control_args}]
```

FUNCTION

returns a string whose content is entirely controlled by specifications in the time_format string.

ARGUMENTS

time_format
: an ioa_-like control string describing the desired result in terms of literal characters and/or date/time selectors. (See Section 1 for a description of time_format.)

time_string
: indicates the date about which information is desired. If you supply no time_string, the current date is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-language STR, -lang STR
: STR specifies the language in which month names, day names, and zone names are to be expressed. (Default: the process default)

-zone STR
: STR specifies the zone to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default language and zone. Use the display_time_info command to display a list of all acceptable language and zone values.

Name: close_file, cf

SYNTAX AS A COMMAND

```
cf {filenames} {-control_args}
```

FUNCTION

closes the specified files belonging to the specified language(s).

ARGUMENTS

filenames
are the names of files opened by the specified language(s).

CONTROL ARGUMENTS

-all, -a
closes all open files that belong to the specified language(s). In this case no filename appears.

-language STR, -lang STR
specifies that files belonging to language STR are to be closed. STR is the unabbreviated name of the language's compiler (e.g., pascal) and can be any language that supports the file-closing subroutine interfaces. If you give no -lang, it closes all open FORTRAN, Pascal, and PL/I files.

NOTES

The format of a FORTRAN file name is fileNN, where NN is a two-digit number other than 00; e.g., file05. PL/I and Pascal file names are selected by you and can have any format.

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

For each filename, all files of that name belonging to the specified language(s) are closed.

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

Name: cobol

SYNTAX AS A COMMAND

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

FUNCTION

invokes the COBOL compiler to translate a segment containing the text of a COBOL source program into a Multics object segment. A listing segment can also be produced. These segments are placed in your working directory. You can't call this command recursively.

ARGUMENTS

path
is the pathname of a COBOL source segment to be translated by the COBOL compiler. If path does not have the .cobol suffix, one is assumed; however that suffix must be the last component of the name of the source segment.

CONTROL ARGUMENTS

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

-card
meaningless trailing blanks are removed from a standard fixed format COBOL source program in card image format. Characters in the identification field (columns 73-80) are ignored.

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

-expand, -exp
a standard fixed format COBOL source program that possibly contains COPY and REPLACE statements is translated into an equivalent source program that does not contain these statements.

-format, -fmt
pseudo-free form COBOL source program is translated into a standard fixed format COBOL source program. For details concerning pseudo-free format see the expand_cobol_source command.

-levelNM, -levNM
causes L-type diagnostics at severity M to be written to the user_output switch whenever a COBOL source line contains a language construct outside the subset specified by N. The value M can be one through three and specifies the severity of the diagnostic. If M is omitted, severity 3 is assumed. The value N can be one through five, corresponding to the four levels specified by the Federal Information Processing Standards Publication, December 1, 1975 (FIPS PUB 21-1) and to the extended version of COBOL supported by Multics. These values are

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

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

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

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

-no_table, -ntb
suppresses the generation of a full symbol table for use by symbolic debuggers.

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

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

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

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

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

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

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

NOTES

The only result of invoking the cobol command without control arguments is to generate an object segment containing a full symbol table.

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

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

The control arguments -format, -card and -expand cause the source program to be pre-translated before compilation. The transformations available are a subset of the transformations available by using the expand_cobol_source command. The translated source program is placed in your process directory with the suffix ex.cobol. Thus compiling name.cobol produces the segment [pd]>name.ex.cobol. If the segment being compiled has the suffix ex.cobol then these control arguments are ignored.

The control arguments -format and -card are inconsistent but either can be used in combination with the control argument -expand. The control argument -expand must be used if the source program contains COPY and REPLACE statements.

For information on COBOL, see the *Multics COBOL User's Guide* (AS43) and the *Multics COBOL Reference Manual* (AS44). See also the description of the profile command.

NOTES ON ERROR DIAGNOSTICS

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

1 Warning only. Compilation continues without ill effect.

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

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

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

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

An example of an error message in its long form is

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

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

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

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

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

NOTES ON LISTING

The listing created by the cobol command is a line-numbered image of the source segment with diagnostics interspersed. This is followed by a cross-reference table of all the names defined within the program. Following the cross-reference table is the object code map, which gives the starting location in the text segment of the instructions for each statement in the program. The map is sorted by ascending storage locations. Finally, the listing contains an assembly-like list of the object code produced. The executable instructions are grouped under an identifying header, which contains the source statement that produced the instruction. Opcode, pointer-register, and modifier mnemonics are printed alongside the octal instruction. If the address field of the instruction uses the IC (self-relative) modifier, the absolute text location corresponding to the relative address is printed on the remarks field of the line.

Name: cobol_abs, cba

SYNTAX AS A COMMAND

```
cba paths {-cobol_args} {-dp_args} {-control_args}
```

FUNCTION

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

ARGUMENTS

paths
are the pathnames of segments to be compiled.

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

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

CONTROL ARGUMENTS

-queue N, -q N
is the priority queue of the request. The default queue is defined by your site. (See "Notes" for a description of the interaction with the dprinting of output files.)

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

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

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

NOTES

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

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

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

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

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

If the -queue control argument is not specified, the request is submitted into the default absentee priority queue defined by the site and, if requested, the output files will be dprinted in the default queue of the request type specified on the command line. (If no request type is specified, the "printer" request type is used.)

If the -queue control argument is specified, and, if requested, the output files will be dprinted in the same queue as is for the absentee request. If the request type specified for dprinting does not have that queue, the highest numbered queue available for the request type is used and a warning is issued.

Name: collate

SYNTAX AS A COMMAND

```
collate
```

SYNTAX AS AN ACTIVE FUNCTION

```
[collate]
```

FUNCTION

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

Name: collate9

SYNTAX AS A COMMAND

```
collate9
```

SYNTAX AS AN ACTIVE FUNCTION

```
[collate9]
```

FUNCTION

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

Name: comp_dir_info

SYNTAX AS A COMMAND

```
comp_dir_info path1 path2 {-control_args}
```

FUNCTION

compares two directory information segments created by save_dir_info and reports on the differences.

ARGUMENTS

path1
: is the pathname of the old directory information segment. If the dir_info suffix is not supplied, it is assumed.

path2
: is the pathname of the new directory information segment. If the dir_info suffix is not supplied, it is assumed.

CONTROL ARGUMENTS

-brief, -bf
: compares and prints minimum information.

-long, -lg
: compares and prints all information.

-verbose, -vb
: compares and prints almost all information. (Default)

NOTES

Output from the comp_dir_info command is written on the user_output I/O switch.

Unless the -brief control argument is specified, a form feed character is transmitted and then a heading is printed that identifies the directories being compared and the times the information was saved.

Output is in three sections:

modified entries
deleted entries
added entries

and is identified by entry type (dir, seg, or link) and the entryname. For deletions and additions, a heading of the form:

deleted: entry entryname

is printed, followed by a listing of the attributes of the deleted or added entry, in the format:

item_name: value

For segments that have been modified, a heading of the form:

modified: entry entryname

is printed, followed by a line of the following formats:

item_name changed from value1 to value2
item_name added: value

(The second format is used to report the addition or deletion of names, ACL entries, etc.)

The list below shows the output items according to the control argument and entry type. The control arguments are listed in order of their verbosity; i.e., the -brief (-bf) control argument prints out the least information, the -verbose (-vb) control argument prints out more information (including the "-bf" items), and the -long (-lg) control argument prints out all of the items listed.

segments:

```
-bf names              -vb safety switch              -lg date modified
    ring brackets          copy switch                    volume
    damaged switch         tpd switch                     bit count
    property list          no_complete_dump_switch        entry point bound
    deletion of ACL        no_incremental_dump switch
    truncation             security OOS switch
                           author
                           bit count author
                           ACL
                           audit flag
                           multiclass switch
                           access class
                           date branch modified
                           records used
                           max length
```

directories:

```
-bf names              -vb safety switch            -lg date branch modified
    ring brackets          copy switch                  date modified
    damaged switch         tpd switch
    property list          no_complete_dump_switch
    deletion of ACL        no_incremental_dump switch
    sons volume            security OOS switch
    master dir             author
    quota                  bit count author
    MSF indicator          ACL
                           audit flag
                           multiclass switch
                           access class
                           initial seg ACL
                           initial dir ACL
```

links:

```
-bf names              -vb date link modified       -lg link dumped
    type
    link target
```

When looking for a match between the old and new dir_info segments, the comp_dir_info command looks first for a match on the unique ID item. If no match is found, it looks for any entry with a name matching the primary name of the old entry.

If a match is found, the comp_dir_info command checks a set of items (depending on the specified control argument) to determine whether to report the entry as modified.

The names item is always checked. The date dumped and date used items are never compared. Other checking is dependent upon the control argument.

If comp_dir_info completes a pass without finding any modifications, deletions, or additions, it prints "Identical". Invoking the command with a more verbose control argument can detect some changes.

Name: compare

SYNTAX AS A COMMAND

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

SYNTAX AS AN ACTIVE FUNCTION

```
[compare path1{|offset1} path2{|offset2} {-control_args}]
```

FUNCTION

compares two files (segments, multisegment files, or archive components) and lists their differences. The comparison is a word-by-word check and can be made with a mask so that only specified parts of each word are compared. The active function returns true if the compared portions are identical, false otherwise.

ARGUMENTS

path1, path2
: are the pathnames of the files to be compared. The equal convention is allowed for path2. Either can be an archive component pathname.

offset1, offset2
: are octal offsets within the files if they are segments or archive components. If you omit them, the entire contents are compared. The comparison begins at the word specified or at the first word of the segment if you specify no offset.

CONTROL ARGUMENTS

-brief, -bf
: prints only the first and last words of each block of discrepancies that is four or more words in length (see "Notes").

-inhibit_error, -ihe
: causes the active function to return "false" rather than produce an error if one of the files to be compared does not exist. An error still occurs if neither file exists.

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

-long, -lg
: prints all discrepancy words (see "Notes"). (Default)

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

-no_inhibit_error, -nihe
suppresses the effect of -inhibit_error.

-short, -sh
prints a single line for each block of discrepancies:

```
 120 words at:  1631
1100 words at: 33404
```

(See "Notes.")

-totals, -tt
prints a single line for the entire comparison:

```
17 differences, 3140 words total.
```

NOTES

The maximum number of words to be compared is the word count of the first segment minus its offset or the word count of the second segment minus its offset, whichever is greater. If you supply -length, comparison stops after the specified number of words. If the segments are of unequal length, the remaining words of the longer segment are printed as discrepancies. The word count of a segment is computed by dividing the bit count plus 35 by 36. If the word count minus the offset is less than zero, an error message is printed and the command is aborted. Any discrepancies found by the command are listed in the following format:

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

To compare segments containing only ASCII character string data, use the compare_ascii command.

Multisegment files (MSFs) are compared component by component, with headers of the form "Component <n>:". Excess components of the longer MSF are listed, the same as for excess words in a longer segment. When a segment is compared to an MSF, a header of the form "Segment/component 0:" or "Component 0/segment:" is printed at the beginning.

You can't use -brief, -long, and -short in the active function.

Name: compare_ascii, cpa

SYNTAX AS A COMMAND

```
cpa paths {-control_args}
```

FUNCTION

compares ASCII segments and prints any differences.

An exec_com tool called compare_pl1 compares PL/I source segments of dissimilar formats via the format_pl1 command (see compare_pl1).

ARGUMENTS

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

CONTROL ARGUMENTS

-extend
: when -output_file is given, the output is appended to the output file if it already exists. (Default)

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

-minchars NN
: specifies the minimum number of characters that must be identical for compare_ascii to assume that it has found the end of a difference (see "Notes"). (Default: 20)

-minlines NN
: specifies the minimum number of lines that must be identical for compare_ascii to assume that it has found the end of a difference (see "Notes"). (Default: 2)

-no_header, -nhe
: does not print the header information. (Default)

-no_numbers, -nnb
: does not print identifying letter and line numbers preceding the lines from the segments being compared. (Default: to print them)

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

-no_output_file, -nof
specifies that output is to be printed on the terminal. (Default)

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

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

-output_file path, -of path
directs the output of the comparison to the file specified by path. The equal convention is allowed, and is applied to the original path.

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

-totals, -tt
prints only the totals line, giving the number of differences and the number of changed lines. (Default: to print discrepancies and the totals line)

-truncate, -tc
specifies that the output file be truncated before the comparison is written into it.

NOTES

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

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

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

EXAMPLES

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

```
yesterday.menu                    today.menu

Breakfast Menu:                   Breakfast Menu:
     Juice                             Juice
     Toast                             Toast
     Eggs                              Eggs
Luncheon Menu:                    Luncheon Menu:
     Hot dogs                          Hamburger
     Milk                              Milk
     French fries                      Salad
Supper Menu:                           French fries
     Steak                        Supper Menu:
     Baked potato                      Chicken
     Coffee                            Rice
                                       Coffee
```

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

```
cpa yesterday.menu today.menu

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

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

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

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

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

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

A6          Hot dogs
Changed by B to:
B6          Hamburger

Inserted in B:
B8          Salad
Preceding:
A8          French fries

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

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

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

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

Hamburger
Salad
Chicken
Rice
```

Name: compare_configuration_deck

SYNTAX AS A COMMAND

```
compare_configuration_deck path1 {path2} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[compare_configuration_deck path1 {path2}]
```

FUNCTION

compares either a saved copy of the configuration deck or the configuration deck for the running system to a saved copy.

As an active function, returns either "true" or "false" to indicate whether the two configuration decks are equivalent.

ARGUMENTS

path1
is the pathname of a saved copy of the configuration deck.

path2
is the pathname of a copy of the configuration deck to be compared against path1. If not supplied, >sl1>config_deck (the configuration deck for the running system) is used.

CONTROL ARGUMENTS

-brief, -bf
suppresses informational messages and printing of the identifying headers.

-label, -lbl
displays cards with mnemonic labels for each field.

-long, -lg
prints all output. (Default)

-no_label, -nlbl
does not display field labels. (Default)

OUTPUT FORMAT (-LONG MODE)

The long output format consists of up to four sections, each of which is printed with an identifying header if it is not empty. The four sections are added cards, deleted cards, changed cards, and mem cards. The section for mem cards is printed only if the order or number of mem cards in the two decks differs; otherwise only changed mem cards are printed. The changed cards are listed in pairs, such as:

compare_configuration_deck

```
Was:    mem  a  123.  on
        mem  a  123.  off
```

The first line (prefaced by Was:) is the card from the saved deck and the second is the current card. If the two decks are different in order or number, this is announced and both decks are printed entirely.

OUTPUT FORMAT (-BRIEF MODE)

The brief output format omits the section headings and the message "The two decks are identical." Cards are identified by preface--added cards are prefaced by "New:" and deleted cards by "Old:". Changed cards are listed in pairs, in the same format as in the long output mode. If the mem cards section is printed, it is the last section. The mem cards are listed in two groups, with the first card in each group prefaced by Was: for the first group or Now: for the second group, and all the other cards in the group are listed with no preface.

NOTES

This command is fairly accurate when identifying "changed" cards--it knows about the cards (such as mem, cpu, etc.) that may appear several times and may specify multiple items and identifies them by their operands as well as by name. It decides that the two decks are completely different if there appear to be more than 32 differences between them.

EXAMPLES OF LONG MODE OUTPUT

Note that because the mem cards have been reordered the changed card for mem A is not listed in the changed cards section.

```
Cards added in current deck:
  parm  chwm  dirw  ttyb  7000.
  salv  pdlv  1

Cards deleted from old deck:
  intk  warm  3  star

Changed cards:
Was:  cpu   b  6  off
      cpu   b  6  on

mem cards are reordered:
Was:  mem   a  258.  on
      mem   c  258.  on
      mem   b  258.  on
Now:  mem   c  258.  on
      mem   a  258.  off
      mem   b  123.  on
```

EXAMPLES OF BRIEF MODE OUTPUT

This output is equivalent to the sample output for the long mode shown above.

```
Old:  salv  pdlv  1
New:  intk  warm  3  star
Was:  cpu   b  6  off
Now:  cpu   b  6  on
Was:  mem   a  258.  on
      mem   c  258.  on
      mem   b  258.  on
Now:  mem   c  258.  on
      mem   a  258.  off
      mem   b  123.  on
```

Name: compare_dump_tape

SYNTAX AS A COMMAND

```
compare_dump_tape -control_args
```

FUNCTION

This command compares Multics storage system hierarchy dump data on two sets of input tape; a master set and a copy set. Options allow selective comparing based upon pathname specifications in a selection file, and comparing using a storage system file containing an image of a set of dump tapes, rather than tapes.

CONTROL ARGUMENTS

-abort
indicates that comparing of the master with the copy should stop when the first discrepancy is found.

-copy_file OUT_PATH, -cf OUT_PATH
gives the pathname of a copy file to be compared with the master data.

-copy_volume VOLNAMES, -cvol VOLNAMES
gives a list of tape volume names. The master data is compared with this copy tape volume set. The names are separated from one another by a blank. Up to 20 volume names can be given. This control argument may be followed by the control arguments described below in "Control arguments for volume attributes".

-master_file IN_PATH, -mf IN_PATH
gives the pathname of a file containing an image of the backup dump tape. This file must have been created by a prior invocation of copy_dump_tape. It contains the master data to be copied.

-master_volume VOLNAMES, -mvol VOLNAMES
gives a list of tape volume names containing the master data to be copied. The names are separated from one another by a blank. Up to 20 volume names can be given. This control argument may be followed by the control arguments described below in "Control arguments for volume attributes".

-maximize_devices, -maxdv
indicates that all tape drives reserved by the process or assigned to the process are to be used equally (round-robin) when mounting tapes.

-no_abort, -nabort
indicates that comparing master and copy should continue when errors are encountered, until 20 discrepancies are found. This is the default.

-no_maximize_devices, -nmaxdv
allows RCP to select which tape drives to use when reading tapes. This is the default.

-no_select, -nslct
indicates that all master data is to be compared with copy data. This is the default.

-no_trace, -ntrace
prevents tracing information from being printed. This is the default.

-select SELECT_PATH, -slct SELECT_PATH
gives the pathname of a file similar to a standard backup_dump control file. This file gives paths of master files to be selected for comparison. See "Notes on control file."

-trace {TYPE}
controls printing of trace information while comparing. This information is primarily used for debugging compare_dump_tape. See "List of trace types".

CONTROL ARGUMENTS FOR VOLUME ATTRIBUTES

The following control arguments define attributes of tape volumes given in preceding -master_volume or -copy_volume control argument.

-density DEN, -den DEN
gives a tape density. DEN may be 800, 1600 or 6250. The input tapes are mounted on a tape drive capable of reading density DEN. However, the actual density at which the input tapes are written determines the density used for reading. The default density is 1600 BPI (bits per inch).

-track TK, -tk TK
mounts tapes on a tape drive capable of handling tapes containing TK tracks. TK may be 7 or 9. The default track size is 9.

LIST OF TRACE TYPES

One of the following trace types may be given as an operand with the -trace control argument. These arguments control the type of trace information printed. If any tracing is enabled, then attach descriptions are printed in addition to the segment information described below.

compare, cmp
during the compare operation, trace master segments selected by paths in the -select file. This is the default if -trace is specified without a TYPE operand.

off
turn off tracing. This is equivalent to -no_trace.

rejects, reject, rej
print master segments unmatched or rejected by paths in the -select file.

LIST OF SEVERITY VALUES

compare_dump_tape sets an external variable to indicate the success or failure of copy and compare operations. This variable may be examined using the severity command/active function. For example:

```
&goto RESULT_&[severity compare_dump_tape]
```

The following severity values can be returned.

0
The compare operation completed successfully.

2
The compare operation completed successfully, but one or more paths given in the -select file were not matched by master segments. These pathnames are listed in a message printed by compare_dump_tape.

3
The compare operation found discrepancies between master and copy segments.

4
The compare operation failed, due to fatal errors. These errors are listed in error messages printed by compare_dump_tape.

NOTES

Either -master_file or -master_volume must be given to specify the source of master input data. Either -copy_file or -copy_volume must be given to specify the source of copy input data.

NOTES ON CONTROL FILE

The control file specified by -select is an ASCII segment containing pathnames of entries (segments, MSFs, and directory subtrees). Each pathname must be given on a separate line. Absolute pathnames must be given, with each entryname of the path being a primary name (the first name of the entry). Master entries matching one of the paths are compared. Master entries which are superior to one of the paths are also compared. If a path identifies a directory, master entries inferior to that directory are compared. A pathname preceded by a circumflex (^) character identifies entries which are NOT to be compared, unless later entries in the control file override the rejection.

For example--

```
   >library_dir_dir>hardcore
  ^>library_dir_dir>hardcore>info
   >library_dir_dir>hardcore>info>hardcore.header
```

selects all entries in the subtree below >library_dir_dir>hardcore, except those in the info directory. However, the hardcore.header entry in the info directory is selected.

compare_entry_names

Name: compare_entry_names, cen

SYNTAX AS A COMMAND

```
cen path1 path2
```

FUNCTION

compares the names on two entries in the storage system and prints information about the differences.

ARGUMENTS

path1
is the pathname of the first entry. You can't use the star convention.

path2
is the pathname of the second entry. You can use the equal convention.

Name: compare_object, cob

SYNTAX AS A COMMAND

```
cob oldpath newpath {-control_args}
```

FUNCTION

compares two object segments and optionally prints out the changes made to the segment specified by oldpath to yield the segment specified by newpath. The assumption is that the first segment is older than the second and that they were both produced from the same source segment but, potentially, by different versions of a language processor.

ARGUMENTS

oldpath
is the pathname of the first segment.

newpath
is the pathname of the second segment. You can use the equal convention.

CONTROL ARGUMENTS

-all, -a
compares the text, definition, and linkage section of the two segments. If the segments have separate static sections, these are compared also. (Default)

-brief, -bf
prints out by section a summary of discrepancies in the object segments, suppressing detailed listing of the discrepancies.

This page intentionally left blank.

-defs
compares the definition sections of the two segments.

-link, -lk
compares the linkage sections of the two segments.

-static
compares the static section of two segments with separate static; otherwise compares the linkage sections.

-text
compares the text sections of the two segments.

NOTES

If no control arguments are specified, the text, definition, and linkage sections of the two segments are compared.

In comparing the lengths of the symbol sections of the two segments, the command uses a heuristic to determine whether a discrepancy is serious or trivial (e.g., caused by differences in pathnames of include files). This heuristic errs in the direction of caution and tends to be inaccurate for large object segments.

Name: compare_pl1, cpp

SYNTAX

```
ec >t>cpp path1 path2
```

FUNCTION

The compare_pl1 exec_com compares two PL/I programs of dissimilar formats.

ARGUMENTS

path1, path2
are the relative or absolute pathnames of the source programs to be compared. The .pl1 suffix is assumed. The star convention is not allowed; the equal convention is allowed for path2. Archive component pathnames are allowed.

NOTES

All format_pl1 control comments are removed from both programs. Then, format_pl1 is used to put both programs into a canonical style. The compare_ascii command is used to see how the source programs differ. Vertical white space inserted or deleted between statements is not ignored. The line numbers in the compare_ascii output is not accurate due to possible white space or statements broken over lines.

Name: component

SYNTAX AS A COMMAND

```
component path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[component path]
```

FUNCTION

returns the archive component name portion of path after it has been expanded into an absolute pathname. If you don't supply an archive component pathname, then this command/active function is equivalent to entry.

ARGUMENTS

path
is the pathname whose archive component name portion is to be returned.

NOTES

Since the pathname is returned in quotes, the command processor treats it as a single argument regardless of special characters in the name.

EXAMPLES

```
! component >udd>Multics>Library>Source>bound_command_demos_.s::program.pl1
  program.pl1

! component >udd>Proj>Myname>start_up.ec
  start_up.ec
```

Name: connect

SYNTAX AS A COMMAND

```
connect channel {destination} {-control_args}
```

FUNCTION

permits you to access a remote system through a dial-out channel (see the dial_out command).

Name: contents

SYNTAX AS A COMMAND

```
contents path {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[contents path {-control_args}]
```

FUNCTION

prints or returns the contents of a segment or archive component as a character string. Newline characters in the segment are changed to blanks in the string.

ARGUMENTS

path
is an absolute or relative pathname to the segment or archive component to be processed

CONTROL ARGUMENTS

-exclude /REGEXP/, -ex /REGEXP/
does not print lines containing a string matching the regular expression REGEXP (see the qedx command).

-exclude STR, -ex STR
does not print lines containing STR. Exclusion is done after matching; thus, "-match A -ex B" prints all lines with an A except those with a B.

-from /REGEXP/, -fm /REGEXP/
begins with the first line matching REGEXP.

-from N, -fm N
begins printing from the Nth line. (Default: 1)

-match /REGEXP/
prints only lines containing a string matching REGEXP.

-match STR
prints only lines containing STR.

-newline, -nl
leaves newline characters in the segment unchanged.

-no_newline, -nnl
changes newline characters in the segment to blanks in the string. (default)

-requote_line, -rql
requotes each line in the segment and changes newline characters in the segment to blanks.

-to /REGEXP/
stops printing with the first line matching REGEXP. The search for REGEXP begins after the first line printed.

-to N
stops printing with line number N. (Default: to print all lines)

maintains a from_string and a to_string that define the conversion to be made. The converted segment is the same as the original except that every instance of the i'th character of from_string present in the original segment is replaced by the i'th character of to_string.

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

ARGUMENTS

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

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

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

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

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

LIST OF KEYWORDS

lc
converts alphabetic characters to lowercase.

uc
converts alphabetic characters to uppercase.

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

bcd
converts BCD special characters to ASCII/EBCDIC equivalents.

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

LIST OF KEYWORDS

bcd
converts BCD special characters to ASCII/EBCDIC equivalents.

lc
converts alphabetic characters to lowercase.

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

uc
converts alphabetic characters to uppercase.

This page intentionally left blank.

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

^	'
-	=
>	"
+	<
=	>
'	:
{	+
"	?
?	

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

NOTES

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

Name: convert_ec, cvec

SYNTAX AS A COMMAND

```
cvec path {-control_args}
```

FUNCTION

Converts an exec_com from one version to another. By default, it converts a Version 1 (old version) exec_com to Version 2, inserting the line "&version 2" at the beginning.

ARGUMENTS

path
is the pathname of an exec_com or absin segment. The ec suffix is added if neither suffix is present. The star convention is allowed.

CONTROL ARGUMENTS

-chase
finds and chases links matching path if path is a starname. (Default: -chase if path is not a starname, -no_chase if path is a starname)

-check, -ck
prints warning and error messages but does not change the segment or produce an output file.

-force, -fc
in the absence of -output_file and -check, forces the original segment to be overwritten even if errors occur. (Default: to create a copy in the process directory if errors occur)

-no_chase
does not operate on links. (Default: -chase if path is not a starname, -no_chase if path is a starname)

-no_check, -nck
converts the segment in addition to printing warning and error messages. (Default)

-no_force, -nfc
does not replace the original segment or create an output file with -output_file if errors (as opposed to warnings) occur. (Default)

-output_file path, -of path
places the converted segment in path instead of the original segment specified by path. The equal convention is allowed in path. If the output segment already exists, it is overwritten. If errors occur, the converted segment is placed instead in the process directory.

-severity N, -sv N
where N is a number from 0 to 3, suppresses warnings/errors with severities lower than N. The default is -sv 2. Severities are as follows:

0 warnings requiring no conversion.
1 warnings (nonstandard but valid syntax), such as unrecognized &strings converted to &&string.
2 errors that can be converted, such as unrecognized &string at the beginning of a line converted to a comment.
3 errors that cannot be converted.

ACCESS REQUIRED

Read access on path1, write access on the output file or append on the parent of the output file if the output file does not exist.

NOTES

Use of -output_file is recommended rather than overwriting the original segment, so that original and converted copies can be compared. The simple conversion rules can, in complicated cases, change the intent of expressions. Therefore, a copy of the original should be kept until the converted exec_com has been shown to operate correctly.

LIST OF CONVERSIONS (V1 -> V2)

leading and trailing whitespace -> literals such as &SP
this conversion is performed because Version 2 strips leading and trailing whitespace from lines.

&<whitespace> -> &-
new comment sequence.

&...& -> &&...&&
strings of two or more ampersands are doubled.

&(...) -> &&(...)
unrecognized by Version 1, the construct on the left is used in Version 1 ec's to pass &(...) parameters to other programs.

&NN -> &(NN)
Version 2 requires parameters with two or more digits to have the digits enclosed in parentheses.

&0, &q0 -> &ec_path
&r0 -> "&ec_path"
new construct to get the expanded, suffixed pathname of the ec.

&if [...] -> &if [...]
the &[...] construct is uniformly required to expand active functions in control lines.

&command_line...-> &trace &command...
&comment_line...-> &trace &comment...
&control_line...-> &trace &control...
&input_line...-> &trace &input...
new tracing statement &trace.

&unrecognized (beginning of line) -> &-&unrecognized
comments entire line if begins with unrecognized keyword.

&unrecognized -> &&unrecognized
all other unrecognized Version 1 &keywords are converted to literals.

EXAMPLES

The command line

```
! cvec >udd>m>Vivaldi>collect -of collect.v2
```

converts the exec_com >udd>m>Vivaldi>collect.ec into Version 2 and places the Version 2 copy in collect.v2.ec in the working directory.

Name: copy, cp

SYNTAX AS A COMMAND

```
cp path1 {path2...path1N path2N} {-control_args}
```

FUNCTION

copies specified segments, multisegment files (MSFs), data management (DM) files, and extended entries in the specified directories with the specified names. Optionally it copies access control lists (ACLs), ring brackets, and multiple names.

ARGUMENTS

path1
: is the pathname of a segment, MSF, DM file, and extended entry to be copied. If it is the name of a link, the command copies the target of the link. The star convention is allowed. (See "Notes" below.)

path2
: is the pathname of a copy to be created from path1. If you don't give path2, the copy is placed in your working directory with the entryname of path1. The equal convention is allowed.

CONTROL ARGUMENTS

-acl
: copies the ACL.

-all, -a
: copies multiple names, ACLs, and ring brackets.

-brief, -bf
: suppresses warning messages (see "Notes").

-chase
: copies the targets of links that match path1 (see "Notes" for the default).

-extend
appends the contents of path1 to the contents' end of path2. An error occurs if path2 does not already exist.

-force, -fc
with -extend or -update, forces writing of the file contents regardless of whether you have write access to path2; in all other cases (replacing the entire file), forces deletion of an existing path2.

-long, -lg
prints warning messages. (Default)

-name, -nm
copies multiple names.

-no_acl
does not copy the ACL. (Default)

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

-no_force, -nfc
does not force write without write access or force deletion of an existing path2. (Default)

-no_name, -nnm
does not copy multiple names. (Default)

-no_ring_brackets, -no_rb, -nrb
does not copy ring brackets. (Default)

-replace, -rp
replaces the entire file path2, rather than modifying its contents as is done by -extend and -update. (Default)

-ring_brackets, -rb
copies the ring brackets of path1 to path2. It is incompatible with -extend and -update.

-update, -ud
replaces the contents of path2 with those of path1 without deleting path2 or changing any of its attributes. An error occurs if path2 does not already exist.

ACCESS REQUIRED

You need read access for path1; write access for path2, unless you use -force with -extend or -update; status permission for the directory containing path1 if you supply -acl, -all, or -name; modify permission for the directory containing path2 if you give -acl, -all, or -name; append permission for the directory containing path2 if you select neither -extend nor -update.

NOTES

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

The default for chasing links depends on path1: if it is not a starname, links are chased by default; if it is a starname, links are not chased.

The initial ACL of the target directory doesn't affect the ACL of the segment or multisegment file being copied. The AIM access class of a segment is not copied by -acl.

Since two entries in a directory cannot have the same entryname, copy takes special action if the name of path1 already exists in the directory specified by path2: if path1 has an alternate name, the entryname that would have resulted in a duplicate name is removed, you are informed of this action, and the copying operation takes place; if path1 has only one entryname, the entry that already exists in the directory must be deleted to remove the name, you are asked if the deletion should be done, and the copying operation does not take place if you answer "no."

This command prints a warning message if the bit count of path1 is less than its current length ("Bit count inconsistent with current length...") or if the current length is greater than the number of records used ("Current length is not the same as records used...").

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_acl

SYNTAX AS A COMMAND

```
copy_acl path1 path2...path1N {path2N}
```

FUNCTION

copies the access control list (ACL) from one segment, directory, multisegment file, data management file, or extended entry to another, replacing the current ACL if necessary. (For a description of ACLs, see the Programmer's Reference Manual.)

ARGUMENTS

path1
is the pathname of a file or directory whose ACL is to be copied. You can specify your working directory with -working_directory (-wd). The star convention is allowed.

path2
is the pathname of a file or directory into which the initial ACL is to be copied. You can specify your working directory with -working_directory (-wd). The equal convention is allowed.

ACCESS REQUIRED

You require status permission for the containing directory of path1 and modify permission for the containing directory of path2.

Name: copy_cards, ccd

SYNTAX AS A COMMAND

```
ccd deck_name {new_deck_name}
```

FUNCTION

copies specified card image segments from system pool storage into your directory. The segments to be copied must have been created using the Multics card input facility.

ARGUMENTS

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

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

NOTES

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

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

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

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

EXAMPLES

The command line

```
! ccd my_deck
```

copies your card image segment named my_deck from the card pool storage into your current working directory.

Name: copy_characters, cpch

SYNTAX AS A COMMAND

```
cpch STR N
```

SYNTAX AS AN ACTIVE FUNCTION

```
[cpch STR N]
```

FUNCTION

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

EXAMPLES

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

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. You can also specify which 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.)

-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. Chasing the links eliminates link translation. (Default: not to chase links)

-force
: executes the command, when target_dir already exists, without asking you. If -force is not selected, you are queried.

-no_link_translation, -nlt
: copies links with no change. 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. (Default: to translate links being copied)

-primary, -pri
copies only primary names. If -primary is not given, 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 nonexistent or empty, this control argument has no effect. (Default: 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 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 copy_dir. Modify and append permission are required on target_dir if it already exists. This command does not force access.

If -acl 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, you are 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, you are queried whether processing should continue. If you answer yes, the contents of the directory are copied (appended) but none of the attributes of that directory are copied; if you answer no, the directory and its subtree is skipped. If name duplication occurs between segments, you are asked whether to delete the existing one in target_dir

If you give -replace 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: copy_dump_tape

SYNTAX AS A COMMAND

```
copy_dump_tape -control_args
```

FUNCTION

This command copies Multics storage system hierarchy dump data from a set of input (master) tapes to a set of output (copy) tapes. Options allow comparing master and copy tapes after the copy operation; selective copying based upon pathname specifications in a selection file; and copying/comparing from or to a storage system file containing an image of a set of dump tapes, rather than tapes.

CONTROL ARGUMENTS

-abort
indicates that comparing of the master with the copy should stop when the first discrepancy is found.

-compare, -cmp
indicates that master and copy should be compared after the copy is generated. Any discrepancies are reported to the user.

-input_file IN_PATH, -if IN_PATH
gives the pathname of a file containing an image of the backup dump tape. This file must have been created by a prior invocation of copy_dump_tape. It contains the master data to be copied.

-input_volume VOLNAMES, -ivol VOLNAMES
gives a list of input tape volume names containing the master data to be copied. The names are separated from one another by a blank. Up to 20 volume names can be given. This control argument may be followed by the control arguments described below in "Control arguments for volume attributes".

-map {MAP_PATH}
controls the generation and naming of a dump map. If -map is given with a MAP_PATH, then a dump map listing the copied files is generated in that file. A suffix of map is assumed if not supplied. If no MAP_PATH is given, the map is generated in a file in the working directory. The file name is derived from other control arguments, as follows. If -select SELECT_PATH is given, then the map file name is the final entryname from SELECT_PATH. If -select is omitted but -ovol VOLNAME is given, the map file is called VOLNAME.map. If -of OUT_PATH is given, the map file is the final entryname from OUT_PATH. Otherwise, the map file name is a unique character string (returned by the unique_chars_ subroutine).

-maximize_devices, -maxdv
indicates that all tape drives reserved by the process or assigned to the process are to be used equally (round-robin) when copying from or to tape, and that during comparison, tape volumes are to be mounted on a different tape drive than was used during copying. This helps detect tape failures caused by reading or writing on a poorly calibrated tape drive.

-no_abort, -nabort
indicates that comparing master and copy should continue when errors are encountered, until 20 discrepancies are found. This is the default.

-no_compare, -ncmp
indicates that master and copy are not to be compared after the copy operation. This is the default.

-no_map, -nmap
indicates that no backup map of the copied data is to be produced. This is the default.

-no_maximize_devices, -nmaxdv
allows RCP to select which tape drives to use when reading or writing tapes. This is the default.

-no_select, -nslct
indicates that all master data is to be copied and compared. This is the default.

-no_trace, -ntrace
prevents tracing information from being printed. This is the default.

-output_discard, -od
indicates that no output copy is to be generated. This is useful in conjunction with -map to produce a map of the master data, or in conjunction with -trace for debugging purposes.

-output_file OUT_PATH, -of OUT_PATH
gives the pathname of a copy file into which the master data is copied.

-output_volume VOLNAMES, -ovol VOLNAMES
gives a list of output tape volume names. The master data is copied onto this copy tape volume set. The names are separated from one another by a blank. Up to 20 volume names can be given. This control argument may be followed by the control arguments described below in "Control arguments for volume attributes".

-select SELECT_PATH, -slct SELECT_PATH
gives the pathname of a file similar to a standard backup_dump control file. This file gives paths of master files to be selected for copying. See "Notes on control file."

-trace {TYPE}
controls printing of trace information while copying and comparing. This information is primarily used for debugging copy_dump_tape. See "List of trace types".

CONTROL ARGUMENTS FOR VOLUME ATTRIBUTES

The following control arguments define attributes of tape volumes given in preceding -input_volume or -output_volume control argument.

-density DEN, -den DEN
gives a tape density. DEN may be 800, 1600 or 6250. If given for input tapes, the tapes are mounted on a tape drive capable of reading density DEN. However, the actual density at which the input tapes are written determines the density used for reading. If given for output tapes, the tapes are written at density DEN. The default density is 1600 BPI (bits per inch).

-track TK, -tk TK
mounts tapes on a tape drive capable of handling tapes containing TK tracks. TK may be 7 or 9. The default track size is 9.

LIST OF TRACE TYPES

One of the following trace types may be given as operand with the -trace control argument. These arguments control the type of trace information printed. If any tracing is enabled, then attach descriptions are printed in addition to the segment information described below.

all, a
during both copy and compare operations, trace master segments selected by paths in the -select file.

compare, cmp
during the compare operation, trace master segments selected by paths in the -select file. No segments are traced during the copy operation.

copy, cp
during the copy operation, trace master segments selected by paths in the -select file. No segments are traced during the compare operation. This is the default if -trace is specified without a TYPE operand.

off
turn off tracing. This is equivalent to -no_trace.

rejects, reject, rej
print master segments unmatched or rejected by paths in the -select file.

LIST OF SEVERITY VALUES

copy_dump_tape sets an external variable to indicate the success or failure of copy and compare operations. This variable may be examined using the severity command/active function. For example:

```
&goto RESULT_& [severity copy_dump_tape]
```

The following severity values can be returned.

0
Both copy and compare operations completed successfully.

2
The copy and compare operations completed successfully, but one or more paths given in the -select file were not matched by master segments. These pathnames are listed in a message printed by copy_dump_tape.

3
The copy operation completed successfully, but the compare operation found discrepancies between master and copy segments.

4
Either copy or compare operations failed, due to fatal errors. These errors are diagnosed in error messages.

NOTES

Either -input_file or -input_volume must be given to specify the source of master input data. -output_discard, -output_file, or -output_volume must be given to specify the target for copied data.

NOTES ON CONTROL FILE

The control file specified by -select is an ASCII segment containing pathnames of entries (segments, MSFs, and directory subtrees). Each pathname must be given on a separate line. Absolute pathnames must be given, with each entryname of the path being a primary name (the first name of the entry). Master entries matching one of the paths are copied and compared. Master entries which are superior to one of the paths are also copied/compared. If a path identifies a directory, then master entries inferior to that directory are copied/compared. A pathname preceded by a circumflex (^) character identifies entries which are NOT to be copied or compared, unless later entries in the control file override the rejection.

For example--

```
>library_dir_dir>hardcore
^>library_dir_dir>hardcore>info
>library_dir_dir>hardcore>info>hardcore.header
```

selects all entries in the subtree below >library_dir_dir>hardcore, except those in the info directory. However, the hardcore.header entry in the info directory is selected.

Name: copy_file, cpf

SYNTAX AS A COMMAND

```
cpf in_control_arg out_control_arg {-control_args}
```

FUNCTION

copies records from an input file to an output file that has been restructured for maximum compactness. The input and output file records must be structured (see "Notes on Unstructured Files" below). The input file can be copied either partially or in its entirety.

ARGUMENTS

in_control_arg
the input file from which records are read can be specified by either of the following:

-input_switch STR, -isw STR
specifies the input file by means of an already-attached I/O switch name, where STR is the switch name.

-input_description STR, -ids STR
specifies the input file by means of an attach description STR. STR must be enclosed in quotes if it contains spaces or other command language characters.

out_control_arg
the output file to which the records are written can be specified by either of the following:

-output_switch STR, -osw STR
specifies the output file by means of an already-attached I/O switch name, where STR is the switch name.

-output_description STR, -ods STR
specifies the output file by means of an attach description STR. STR must be enclosed in quotes if it contains spaces or other command language characters.

CONTROL ARGUMENTS

-all, -a
copies until the input file is exhausted. (Default)

-brief, -bf
suppresses a message indicating the number of records or lines actually copied.

-count N, -ct N
copies until N records have been copied or the input file is exhausted, whichever occurs first, where N is a positive integer. (Default: to copy until the input file is exhausted)

-from N, -fm N
copies records beginning with the Nth record of the input file, where N is a positive integer. (See "Notes.") (Default: to begin copying with the "next record")

-keyed
copies both records and keys from a keyed sequential input file to a keyed sequential output file. (See "Notes on Keyed Files.") (Default: to copy records from an input file, keyed or not, to a sequential output file)

-long, -lg
prints a message indicating the number of records or lines actually copied: "345 records copied". (Default)

-start STR, -sr STR
copies records beginning with the record whose key is STR, where STR is 256 or fewer ASCII characters. (Default: to begin copying with the "next record")

-stop STR, -sp STR
copies until the record whose key is STR has been copied or the input file is exhausted, whichever occurs first, where STR is 256 or fewer ASCII characters. This control argument can be given without specifying -start. However, if -start is supplied, the STR used with -stop must be greater than or equal to (according to the ASCII collating sequence) the STR given with -start.

-to N
copies until the Nth record has been copied or the input file is exhausted, whichever occurs first, where N is a positive integer greater than or equal to the N given with -from. If you use -to, you must give -from.

NOTES

If either the input or output specification is an attach description, it is used to attach a uniquely named I/O switch to the file. The switch is opened, the copy performed, and then the switch is closed and detached. Alternately the input or output file can be specified by an I/O switch name. Use either io_call or iox_ to attach the file prior to the invocation of copy_file.

If the input file is specified by an I/O switch name and the switch is not open, copy_file opens it for (keyed_)sequential_input, performs the copy, and closes it. If the switch is already open when copy_file is invoked, the opening mode must be sequential_input, sequential_input_output, keyed_sequential_input, or keyed_sequential_update. The switch is not closed after the copy has been performed.

The "next record" must be defined if neither -start nor -from is specified as the absolute starting position within the input file. If the I/O switch is opened by copy_file, the next record is the first record of the file; otherwise the next record is the one at which the file is positioned when copy_file is invoked.

If the output file is specified by an I/O switch name and the switch is not open, copy_file opens it for (keyed_)sequential_output, performs the copy, and closes it. If the switch is already open when copy_file is invoked, the opening mode must be sequential_output, sequential_input_output, keyed_sequential_output, keyed_sequential_update, direct_output, or direct_update. (In update mode, output file records with keys that duplicate input file records are rewritten.) The switch is not closed after the copy has been performed.

The following control arguments are mutually exclusive: -from and -start; -to, -stop, -count, and -all; -brief and -long.

NOTES ON UNSTRUCTURED FILES

This command operates by performing record I/O on structured files. If you want to copy from/to an unstructured file, you can use the record_stream_ I/O module:

```
! cpf -ids "record_stream_ -target vfile_ pathname" -osw OUT
```

which takes lines from the file specified by pathname via the vfile_ I/O module, transforms them into records via the record_stream_ I/O module, and copies them to the I/O switch named OUT.

NOTES ON KEYED FILES

The command can copy a keyed sequential file to produce an output file that has been restructured for maximum compactness as a keyed file or as though it were sequential. By default it copies only records and does not place keys in the output file. To copy the keys, use -keyed. When you select -keyed the input file must be a keyed sequential file. Whether keys are copied or not, choose control arguments to delimit the range of records to be copied (e.g., -start, -stop). Copying is always performed in key order.

If the keyed file has keys but no records (e.g., a dictionary file), the file, its keys, and its associated record descriptors are copied.

EXAMPLES

To copy an entire file from an already-attached file to the segment in_copy, type

```
cpf -isw in -ods "vfile_ in_copy"
```

To print the first 13 records of a tape file:

```
cpf -ct 13 -ids "tape_ansi_ 887677 -name TEST21 -ret all"
   -ods "record_stream_ user_output"
```

To copy 13 records from an already-attached file to another already-attached file, starting with the 56th record of the input file:

```
cpf -isw in -osw out -from 56 -ct 13
```

To copy records 43 through 78 from an already-attached file to an already-attached file:

```
cpf -isw in -osw out -from 43 -to 78
```

To copy all but the first seven records from segment testdata.11 to an already-attached file:

```
cpf -ids "vfile_ testdata.11" -osw out -fm 8
```

To copy an entire keyed sequential file with keys:

```
cpf -isw in -osw out -all -keyed
```

To copy 13 records of a keyed sequential file starting with the record whose key is ASD66 to a sequential output file, the following line is typed. (No keys are copied.)

```
cpf -isw in -osw out -sr ASD66 -ct 13
```

To copy the records and keys from a keyed sequential file up to and including the record whose key is bb"bb, type

```
cpf -keyed -isw in -osw out -sp "bb""bb"
```

Name: copy_iacl_dir

SYNTAX AS A COMMAND

```
copy_iacl_dir path1 path2 {...path1N path2N}
```

FUNCTION

copies the initial access control list for directories (directory initial ACL) of one directory to another, replacing the current directory initial ACL if necessary. (See the Programmer's Reference Manual for a description of initial ACLs.)

ARGUMENTS

path1
: is the pathname of a directory. You can specify your working directory with -working_directory (-wd). The star convention is allowed.

path2
: is the pathname of the target directory. You can specify your working directory with -working_directory (-wd). The equal convention is allowed.

ACCESS REQUIRED

You need status permission on path1 and modify permission on path2.

Name: copy_iacl_seg

SYNTAX AS A COMMAND

```
copy_iacl_seg path1 path2 {...path1N path2N}
```

FUNCTION

copies a segment initial access control list (initial ACL) from one directory to another, replacing the current initial ACL if necessary. (See the Programmer's Reference Manual for a description of initial ACLs.)

ARGUMENTS

path1
: is the directory from which the initial ACL is to be copied. You can specify your working directory as -working_directory (-wd). You can use the star convention.

path2
: is the directory into which the initial ACL is to be copied. You can specify your working directory as -wd. You can use the equal convention.

ACCESS REQUIRED

You need status permission on path1 and modify permission on path2.

Name: copy_names

SYNTAX AS A COMMAND

```
copy_names path1 {path2...path1N path2N}
```

FUNCTION

copies all the names of one entry--directory, segment, multisegment file (MSF), data management (DM) file, extended entry, or link--to another.

ARGUMENTS

path1
: is the pathname of the entry whose names are to be copied. You can use the star convention.

path2
is the pathname of the entry to which all names are copied. If you omit path2N, names are copied into an entry in you working directory with the same entryname as path1N. You can use the equal convention.

NOTES

All names are left on the original entry. The two entries cannot reside in the same directory because duplicate names are not allowed in a directory.

Only one matching name per entry is used when resolving the equal name. This is the first matching name on that entry (in the order returned by hcs_$star_) for which the specified equal name exists.

Name: create, cr

SYNTAX AS A COMMAND

```
cr paths {-control_args}
```

FUNCTION

creates a storage system entry for an empty segment in any directory.

ARGUMENTS

paths
are pathnames of segments to be created.

CONTROL ARGUMENTS

-max_length N, -ml N
sets the maximum length of the created entry to N. Used with -msf, -ml sets future MSF components to N words long.

-multisegment_file, -msf
creates an MSF with one empty component, instead of an empty segment. When you foresee that you need much storage, creating an MSF prevents the expensive copying occurring when a segment is converted to an MSF.

-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 your validation level.

ACCESS REQUIRED

You must have m access to a directory to create the segment, and you are given rw to it.

NOTES

If there is a one-name segment with an identical name to the segment you are creating, you are asked whether to delete the old segment. If it has multiple names, the conflicting one is removed and a message is issued to you. In either case, since the directory is being changed, you must also have modify permission for the directory.

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 create_dir and link commands for the creation of directories and links.

EXAMPLES

The command line

```
! cr first_class_mail >udd>Demo>IEBunin>alpha>beta
```

creates the segment first_class_mail in the working directory and the segment beta in the directory >udd>Demo>IEBunin>alpha.

Name: create_area

SYNTAX AS A COMMAND

```
create_area virtual_pointer {-control_args}
```

FUNCTION

creates an area and initializes it with user-specified area management control information.

ARGUMENTS

virtual_pointer
: is a virtual pointer specifier to the area to be created (see Section 1 for a description of virtual pointers). If the segment already exists, the specified portion is still initialized as an area.

CONTROL ARGUMENTS

-dont_free
is used during debugging to disable the free mechanism. This does not affect the allocation strategy.

-extend
causes the area to be extensible, i.e., span more than one segment. This feature should be used only for perprocess, temporary areas.

-id STR
specifies a string to be used in constructing the names of the components of extensible areas.

-no_freeing
allows the area management mechanism to use a faster allocation strategy that never frees.

-size N
specifies the octal size, in words, of the area being created or of the first component, if extensible. If this control argument is omitted, the default size of the area is the maximum size allowable for a segment. The minimum area is forty octal words.

-zero_on_alloc
instructs the area management mechanism to clear blocks at allocation time.

-zero_on_free
instructs the area management mechanism to clear blocks at free time.

Name: create_data_segment, cds

SYNTAX AS A COMMAND

```
cds path {-control_arg}
```

FUNCTION

translates a create_data_segment (CDS) source program into an object segment. A listing segment is optionally created. These results are placed in your working directory. This command cannot be called recursively.

ARGUMENTS

path
is the pathname of a CDS segment. If path does not have a cds suffix, one is assumed; however the cds suffix must be the last component of the name of the source segment.

CONTROL ARGUMENTS

-list, -ls
produces a source listing of the CDS program used to generate the data segment followed by object segment information (as printed by the print_link_info command) about the actual object segment created.

NOTES

The source for create_data_segment programs is standard PL/I with the restriction that the program include a call to the create_data_segment_ subroutine. The create_data_segment_ subroutine creates a standard object segment from PL/I data structures passed to it as parameters. These data structures can be initialized with arbitrarily complex PL/I statements in the CDS program.

Since the create_data_segment command invokes the PL/I compiler to first compile the CDS segment, any errors that the compiler finds are reported by its standard technique. If any errors with a severity greater than 2 occur, the CDS run is aborted and an object segment is not created.

Name: create_dir, cd

SYNTAX AS A COMMAND

```
cd paths {-control_args}
```

FUNCTION

creates a specified directory branch in a specified directory or in your working directory; that is, it creates a storage system entry for an empty subdirectory.

ARGUMENTS

paths
are pathnames of directories to be created.

CONTROL ARGUMENTS

-access_class STR, -acc STR
applies to each pathi and upgrades each directory created to the specified access class. You can give the access class with either long or short names.

-account STR, -acct STR
specifies the volume quota account from which the created master directory is to draw its quota, where STR must match an existing quota account on the given logical volume. If omitted, an account that matches the owner User_id is used (if any). You can supply -account only if you select -logical_volume.

-dir_quota N
specifies the directory quota to be given to the directory when it is created, where N must be a positive integer and applies to each pathi. If omitted, the directory is given zero directory quota.

-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.

-owner USER_ID, -ow USER_ID
specifies the owner of the created master directory. You can supply -owner only if you select -logical_volume. (Default: your User_id)

-quota N
specifies the segment quota to be given to the directory when it is created, where N must be a positive integer and applies to each pathi. You must provide -quota if you use either -access_class or -logical_volume. If omitted, the directory is given zero segment quota.

-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

You must have a access to a directory in order to create a subdirectory in that directory.

The -account and -owner control arguments are allowed only for volume administrators (i.e., only those who have e access to the volume).

NOTES

If you specify a directory or segment quota and the directory you are creating is not a master directory, the containing directory must have sufficient directory or segment quota to move quota to the directory being created (see move_quota).

If the creation of a new subdirectory introduces a duplication of names within the directory and if the old entry has only one name, you are asked whether to delete the old entry. If the old entry has multiple names, the conflicting name is removed and a message is issued to you. You are given sma access on the created subdirectory.

All superior directories specified in pathi must already exist. That is, you can only create a single level of storage system directory hierarchy in a single invocation of create_dir.

To create a master directory, you must have a quota account on the logical volume with sufficient volume quota to create the directory. If you are not a volume administrator, you can create a master directory only if the administrator has created a quota account that matches your User_id. A master directory must always have a nonzero quota; therefore you must always give -quota when creating a master directory. You can create a master directory 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 you supply -access_class, 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.

See the create and link commands for the creation of segments and links.

EXAMPLES

The command line

```
! cd sub >my_dir>alpha>new
```

creates the directory sub immediately inferior to your current working directory and the directory new immediately inferior to the directory >my_dir>alpha. 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, immediately inferior to your working directory, with an access class of a,c1,c2 and a quota of five pages. (The access class names a, c1, and c2 used in the example represent possible names defined for your site. See print_auth_names for details on access class names.)

The command line

```
! cd subB -logical_volume volz -quota 100
```

creates a master directory subB immediately inferior to your 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: create_dm_file

SYNTAX AS A COMMAND

```
create_dm_file path {-control_args}
```

FUNCTION

creates an unpopulated data management (DM) file for use with Multics data management (see the Programmer's Reference Manual). Files created in this manner would be used primarily for test purposes or in applications calling the file_manager_ subroutine directly.

ARGUMENTS

path
is the pathname of the DM file to be created.

CONTROL ARGUMENTS

-concurrency, -conc
provides automatic concurrent access protection to a protected DM file by enforcing locking conventions on get and put operations to the file (see "Notes"). (Default)

-no_concurrency, -nconc
turns concurrency protection off for a protected DM file, saving on overhead when locking is unnecessary (see "Notes").

-no_rollback, -nrlb
turns the rollback capability off for a protected DM file, saving on overhead when rollback is unnecessary (see "Notes").

-protected, -prot
creates a protected DM file, which means the file is entitled to the protection features provided by the integrity services of data management. You can access a protected DM file only within the context of a transaction. (See "Notes.") (Default)

-ring_brackets W {R}, -rb W {R}
sets the write ring bracket to W and the read ring bracket to R. If you don't specify the read ring bracket, it defaults to the value given for the write ring bracket. Both W and R must be greater than or equal to your validation level.

-rollback, -rlb
provides an automatic rollback capability for a protected DM file by logging before images of modifications made to the file. These images are used in the event of transaction, process, or system failure to restore the file to its original state. (See "Notes.") (Default)

-unprotected, -unprot, -not_protected, -nprot
creates an unprotected DM file, without the benefit of integrity services. You can access an unprotected DM file outside a transaction. (See "Notes.")

ACCESS REQUIRED

You need sma access on the directory in which the DM file is created and s access on the directory containing that directory.

NOTES

The -unprotected control argument is mutually exclusive with -concurrency, -no_concurrency, -no_rollback, -protected, and -rollback. If you use two mutually exclusive control arguments, the rightmost option in the command line takes precedence.

This command is part of the command level interface to Multics data management.

Name: cross_reference, cref

SYNTAX AS A COMMAND

```
cref library_descriptions {-control_args}
```

FUNCTION

creates a cross-reference listing of any number of object programs. The listing contains information about each object module encountered, including the location of each program, its entry points and definitions, any synonyms, and which other modules encountered reference each entry point or definition. It also optionally supplies a cross-reference listing of include files used by the modules encountered.

This page intentionally left blank.

ARGUMENTS

library_descriptions
can be chosen from the following:

paths
are the pathnames of segments to be examined and cross-referenced. The star convention is allowed.

-library library_name {-all} paths, -lb library_name {-all} paths
specifies that all modules represented by paths are treated by the cross-referencer as if they were in a common library of that name. The library_name argument can be any identifier you choose. If -library contains -all, all the module names encountered are considered external (see "Resolving References.") This control argument is generally used only for cross-references of the Multics hardcore libraries.

CONTROL ARGUMENTS

-brief, -bf
suppresses nonfatal error messages. It does not affect the reporting of error messages to the output file.

-first
specifies, with -input_file, that once any instance of a particular module has been located, the cross-referencer need not search the remaining directories for other instances of modules with the same name. If omitted, the cross-referencer searches all libraries in the search list for each module name supplied.

-include_files, -icf
cross-references include files used by all modules examined.

-input_file path, -if path
uses a control file describing the modules to be cross-referenced instead of the library descriptions. If the crl suffix is not part of the supplied filename, it is assumed. If -input_file is given, no library descriptions are allowed.

-line_length N, -ll N
formats lines in the output file to the given line length. (Default: 132)

-output_file path, -of path
creates the cross-reference list in a segment of the specified name. If the cross-reference suffix is not part of the supplied filename, it is assumed. If -output_file is not selected, but -input_file is, the output file takes its name from the input file, with the suffix ".crossref" replacing the suffix ".crl"; otherwise, the output file is named "crossref.crossref".

-short, -sh
does not include in the output referenced modules that are not included in the scope of any library_descriptions. With -short, the output reflects only the interrelationships among the modules in the libraries specified.

MODULE EXAMINATION

Module examination is performed in two passes: the first defines all the segment names, synonyms, and definitions; the second examines external references and attempts to resolve them with existing definitions.

Segments encountered fall into four classes: nonobject, bound segments, stand-alone modules, and archives.

When a nonobject segment is encountered, a warning message is printed and the segment is included in the results of the cross-reference.

When a bound segment is found, a warning message is printed and the segment is ignored. Bound segments are useless to the cross-referencer, since information necessary to determine which components use which external reference links is no longer available due to the binding process. Use instead the object archive from which it was bound.

When a stand-alone segment is met, it is analyzed for entry points, definitions, and external references. All additional names on the segment are entered as synonyms for the module. This information is then included in the results of the cross-reference.

When an archive is encountered, each component is analyzed for entry points, definitions, and external references. If a bindfile exists, synonyms for each component are derived from "synonym" statements in the bindfile, when they exist. This information is then included in the results of the cross-reference.

Modules are also identified by the segment in which they are found (either themselves, for a stand-alone segment, or the containing archive, for an archive) and by the library_name of the directory in which they are found. If the directory is given without a library_name, the pathname of the directory is used as the library_name. This allows having multiple occurrences of segments with the same name, as long as they differ by at least one of these identification criteria.

RESOLVING REFERENCES

When a module is examined by the cross-referencer, its name and synonyms are classified as "internal" or "external" by the following criteria:

1. If the module is stand alone, its name and synonyms are external.

2. If the module is archived and the library description contained -all , its name and all its synonyms are considered external.

3. If the module is archived and the library description did not contain -all, its name and each of its synonyms are external only if they appear in the "Addname:" statement of the bindfile. If no bindfile exists, the name and synonyms are considered internal.

The cross-referencer tries to resolve external references on a best-match basis by using the following criteria:

1. If the reference can be satisfied by a definition in the same module, that definition is used.

2. If the referencing module is part of a bound segment and can be satisfied by a definition in the same bound segment, that definition is used.

3. If the reference can be satisfied by an external definition in the same library_name, that definition is used.

4. Otherwise, the first external definition found that satisfies the reference is used. If more than one such definition exists, a warning message is printed.

FORMAT OF A DRIVING FILE

If -input_file is given, the cross-referencer takes its input from a special file.

The first lines of the file must contain the names of one or more directories to be searched. They are specified in the following manner:

```
-library:       (OR  -library -all:)
pathname_1           library_name_a
pathname_2           library_name_b
   ....
pathname_N           library_name_z;
```

Each pathname_i specifies a directory to be searched. When present, a library name (which can contain spaces) is used to describe the preceding directory name. (See "Module examination" above.) The tokens "-wd" or semicolon ends the search list.

The next information in the file is a list of the segments to be examined. They must appear one to a line.

If you wish to define explicitly synonyms for any modules that would not otherwise be generated (e.g., a nonapparent reference name by which a segment is sometimes initiated), they can be included in this section with one or more lines of the form:

```
modulename syn1 syn2 ... synN
```

These lines do not by themselves cause the cross-referencer to search for the module "modulename", since it may not be a freestanding segment. Any synonyms defined in this manner are considered external.

A file can consist of several repetitions of the format described above; that is, a search list, segment names, another search list, more segment names, etc. Whenever a new search list is found, it replaces the old search list. If a driving file is to be used, make it consist of multiple occurrences of a one-directory search list followed by the segments contained in that directory.

Here is example of a control file constructed to cross-reference a student subsystem:

```
-library:
>udd>Class>systemdir>object CLASS SUBSYSTEM;

class_login_responder.archive
class_tests.archive
student_grades_database
audit_procedure
class_utilities.archive
unallowed_compiler_stub fortran pl1
unallowed_compiler_stub
```

SPECIAL CASES

Segments with unique names and with single-digit last components are ignored, since these are conventions used by the system library tools to denote segments to be deleted shortly.

Archives whose names are identical with the exception of a different numeric next-to-last component are considered the same archive.

Definitions or entry points in archive components that masquerade as segment names by an added name on the bound segment, without being defined as a synonym for their containing component, are not cross-referenced satisfactorily.

INCLUDE FILES

The cross-reference listing of include files, when requested, is appended to the regular output of the cross-referencer. Each include file met is classified by its entryname and its date/time modified. This ensures that modules that use different versions of the same include file are apparent.

EXAMPLES

The following command produces a cross_reference listing of the Standard Service System in the file "standard.crossref":

```
!  cref -library STANDARD >ldd>sss>o>** -of standard
```

To produce a cross-reference listing of the hardcore library, you can use

```
!  cref -library HARD -all >ldd>h>x>* >ldd>h>o>*.archive -of hard
```

OUTPUT EXAMPLE

Entries are separated by dashed lines in the output listing. The following is a sample entry:

```
----------*****bound_x_ in SSS *****---------------
sample_segname        SYNONYM:  one syn, another_syn
 one_entrypoint       program_a program_b
 second_entrypoint    program_a program_c
 unused_entrypoint
 undefined_ent (?)    program_d
```

The entry shown is for segment "sample_segname", which is a component of bound_x_ in the library specified as SSS. It possesses three entry points: "one_entrypoint", "second_entrypoint", and "unused_entrypoint". The information shows that "sample_segname$one_entrypoint" is called by module "program_a" and module "program_b". The question mark after entry point "undefined_ent" signifies that this entry point is an implicit definition: the module "program_d" refers to "sample_segname$undefined_ent", but that entry point does not exist. (A diagnostic is printed when this situation is encountered.)

All error messages produced during the run, including warning messages that may not have been printed at the terminal because of -brief, are appended to the end of the output file for reference.

Name: cumulative_page_trace, cpt

SYNTAX AS A COMMAND

```
cpt command_line {-control_args}
```

FUNCTION

accumulates page trace data so that the total set of pages used during the invocation of a command or subsystem can be determined. The command accumulates data from one invocation of itself to the next. The output is in tabular format showing all pages that have been referenced by your process. You can obtain the same trace as that produced by page_trace.

ARGUMENTS

command_line
: is a character string to be interpreted by the command processor as a command line. If this character string contains blanks, enclose it in quotes. All procedures invoked as a result of processing this command line are metered by cumulative_page_trace.

CONTROL ARGUMENTS

-count, -ct
prints the accumulated results, giving the number of each page and the number of faults for each page. Do not use -count with -print or -total (see "Notes" below).

-flush
clears primary memory before each invocation of the command line and after each interrupt. This helps you determine the number of page faults but increases the cost.

-interrupt N, -int N
interrupts execution every N virtual CPU milliseconds for page fault sampling. (Default: 500 CPU milliseconds)

-long, -lg
produces output in long format, giving full pathnames.

-loop N
calls the command to be metered N times.

-print, -pr
prints the accumulated results, giving the number of each page referenced. Do not use -print with -count or -total (see "Notes.")

-print_linkage_faults
prints all accumulated linkage faults and calls the hcs_$make_ptr entry point.

-reset, -rs
resets the table of accumulated data. If the table is not reset, data from the current use of cumulative_page_trace is added to that obtained earlier in the process.

-short, -sh
formats output for a line length of 80.

-sleep N
waits for N seconds after each call to the command being metered.

-temp_dir path, -td path
creates temporary segments, used for flushing main memory, in the directory identified by path. Use -temp_dir with -flush. (Default: to create them in the process directory)

-timers
includes all faults between signal and restart.

-total, -tt
prints the total number of page and segment faults and the number of pages referenced for each segment. Do not use -total with -count or -print (see "Notes.")

-trace_linkage_faults
accumulates linkage, page, and segment fault information.

-trace path
writes the trace on the segment "path" using an I/O switch named "cpt.out" (see "Examples"); cumulative_page_trace attaches and detaches this switch.

NOTES

This command operates by sampling and reading the system trace array after invocation of a command and at repeated intervals.

At least one of three generic operations must be requested. They may all be combined and, if so, are performed in the following order: resetting the table of accumulated data, calling the command to be metered, applying the specified control arguments, and printing the results in the specified format. If 500 milliseconds is too long, messages indicate that some page faults may have been missed; choose then a smaller value, but the cost of a smaller value is high and may cause additional side effects. If the command or subsystem to be metered includes the taking of CPUT interrupts, then supply -timers, which includes some of the page faults of the metering mechanism as well.

You can give only one of -count, -print, or -total. Each of these control arguments produces printed output in a different format. If you want more than one format, invoke the command once for each format.

For -flush to operate correctly, the directory used for temporary segments must have sufficient quota for as many pages as there are in main memory.

EXAMPLES

The command line

```
! cpt "pl1 test" -interrupt 400 -trace trace_out
```

calls the pl1 command to compile the program named "test," requesting an interrupt every 400 milliseconds to obtain page trace information. Trace information is placed in a segment named "trace_out."

The command line

```
! cpt "list -pn >udd>Multics" -loop 2 -sleep 10
```

calls the list command twice and sleeps for 10 seconds between calls.

The command line

```
! cpt -print
```

prints the accumulated results of previous metering. The command line

```
! cpt "dco ls" -trace cpt.trace
```

produces the following output:

```
LINKAGE FAULT BY      2.05   674        bound_metering_: cpt|2136
RESOLVED LINK TO      3.14   256        bound_library_1_$cp
MAKE-PTR-CALL        10.18
                      5.18   310    7   s
.                      .      .     .   .
.                      .      .     .   .
.                      .      .     .   .
```

where the first column is the event, what is happening (if it is blank, it means that there is a page fault); the second is the CPU time; the third, the segment number; the fourth, the page number; and the fifth, the name of the segment.

Name: cv_ttf

SYNTAX AS A COMMAND

```
cv_ttf path {-control_arg}
```

FUNCTION

compiles a terminal type file (TTF) into a terminal type table (TTT) for installation.

ARGUMENTS

path
: is the pathname of the TTF to be compiled. It must have the ttf suffix. The resulting TTT is placed in your working directory; its entryname is the same as the entryname of the TTF with the ttt suffix added.

CONTROL ARGUMENTS

-brief, -bf
: prints all error messages produced by cv_ttf in short form.

-long, -lg
: prints all error messages produced by cv_ttf in long form.

-severity N, -sv N
does not write error messages whose severity is less than N (where N is 0, 1, 2, 3, or 4) to the user_output switch. If not given, a severity level of 0 is assumed; i.e., all error messages are written to the user_output switch. (See "Notes on Severity Values.")

NOTES

If neither -brief nor -long is selected, the first instance of a given error produces a long message and all subsequent instances of that error produce short messages.

NOTES ON SEVERITY VALUES

This command associates the following severity values to be used by the severity active function:

Value	*Meaning*
0	No compilation yet or no error
1	Warning
2	Correctable error
3	Fatal error
4	Unrecoverable error
5	Could not find source

Name: date

SYNTAX AS A COMMAND

```
date {time_string} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[date {time_string} {-control_arg}]
```

FUNCTION

returns a date of the form "mm/dd/yy" (e.g., "12/23/82"). The format string to produce this is "^my/^dm/^yc".

ARGUMENTS

time_string
indicates the date about which information is desired. If you supply no time_string, the current date is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-zone STR
STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default zone. Use the display_time_info command to display a list of all acceptable zone values.

Due to exec_coms, etc., that have been built around the expected date format, this command does not honor the process date format (set by set_time_default). You are encouraged to use "clock date" instead of date to get the proper default handling.

Name: date_compiled, dtc

SYNTAX AS A COMMAND

```
dtc path {-control_arg} {components}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dtc path {-control_arg} {components}]
```

FUNCTION

prints the date and time compiled and the compiler identifier for an object file or an archive of object segments. For a bound object file, it prints the date and time compiled for each component. The active function returns the first line of output that would be printed if you invoked it as a command.

ARGUMENTS

path
is the pathname of an object segment, bound object segment, bound object multisegment file (MSF), or an archive of object segments.

components
are names of components in a bound object file or archive of object segments. If you supply component names, information on only these components is listed.

CONTROL ARGUMENTS

-brief, -bf
lists only the date and time compiled (see "Examples").

-long, -lg
lists the date and time compiled, the file name, your User_id, and the long form of the compiler identifier (see "Examples").

NOTES

If an archive is listed, the bind file is ignored.

If you give neither control argument, dtc lists the date and time compiled, the file name, your User_id, and the short compiler identifier (see "Examples").

EXAMPLES

To check the compilation date of a private version of the list command in your working directory, type

```
! dtc list -bf
  04/11/83 0922.2
```

To check information on the latest compilation of a Multics-system-installed command that is unbound, such as demo_command, type

```
! dtc >system_library_standard>demo_command
  03/09/83 1615.2 demo_command ARomanov.SysMaint.a PL/I
```

To get compilation information on an entire bound object segment that is part of the standard Multics system, type

```
! dtc >sss>bound_binder_

  Bound 10/26/83 1337.4 bound_binder_ ARomanov.SysMaint.a binder
  07/26/82 1048.4 bind ARomanov.SysMaint.a PL/I
  10/26/83 1328.2 bx_ ARomanov.SysMaint.a cds
                        .
                        .
                        .
  12/27/82 1354.3 old_make_bindmap_ BDerek.SysMaint.a PL/I
```

To get detailed information on one component of a bound object segment (in this case, bind in bound_binder_), type

```
! dtc >sss>bound_binder_ -lg bind

  07/26/83  1048.4 bind ARomanov.SysMaint.a Multics PL/I
    Compiler, Release 25q, of May 22, 1983
```

Name: date_deleter

SYNTAX AS A COMMAND

```
date_deleter dir_path cutoff {star_names} {-control_args}
```

FUNCTION

deletes segments and multisegment files (MSFs) older than a specified number of days or older than a given date-time.

ARGUMENTS

dir_path
: is the pathname of the directory in which the deletions are to occur; dir_path can be -working_directory (-wd) to indicate the working directory.

cutoff
: is a positive integer number of days. If it is an integer N, files with a date more than N days old are deleted; if it is a date-time DT, files with a date earlier than DT are deleted. (See Section 1 for a description of valid DT values.)

star_names
: are the optional starnames of files to be deleted. If you supply none, all files older than the specified number of days are deleted; otherwise only files matching one or more of the starnames, and older than the specified number of days, are deleted.

CONTROL ARGUMENTS

-absolute_pathname, -absp
: prints the entire pathname of the entries listed by -long, -query_all, and -query_each. (Default: to print entrynames)

-all, -a, -branch, -br
: deletes directories, segments, and multisegment files.

-date_time_contents_modified, -dtcm
: uses the date/time value specified in the dtcm attribute to calculate the deletion date. (Default)

-date_time_dumped, -dtd
: uses the dtd of each entry instead of the dtcm.

-date_time_entry_modified, -dtem
: uses the dtem of each entry instead of the dtcm.

-date_time_used, -dtu
: uses the dtu of each entry instead of the dtcm.

-directory, -dr
deletes directories only.

-entryname, -etnm
prints only the entrynames of the files listed by -long, -query_all, and -query_each rather than the entire pathname. (Default)

-file, -f
deletes segments and multisegment files. (Default)

-long, -lg
prints a message of the form "Deleted <type> <path>" for each entry deleted.

-multisegment_file, -msf
multisegment files only.

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

-query_all, -qya
lists all entries to be deleted and queries whether they should be deleted or not.

-query_each, -qye
queries for every entry to be deleted.

-segment, -sm
deletes segments only.

EXAMPLES

The command line

```
! date_deleter >ldd>old 7
```

deletes all files in >ldd>old last modified more than a week ago.

The command line

```
! date_deleter >udd>Proj>listing_pool 2 **.list
```

deletes all listing files in the directory >udd>Proj>listing_pool that are more than two days old.

Name: date_time

SYNTAX AS A COMMAND

```
date_time {time_string} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[date_time {time_string} {DT} {-control_args}]
```

FUNCTION

returns a date and time value for a specified date-time or the current date-time consisting of a date, a time from 0000.0 to 2359.9, a time zone, and a day of the week. The date and time value is returned as a single quoted string of the form "mm/dd/yy hhmm.m zzz www" (e.g., "06/01/84 0840.9 mst Fri"). The format string to produce this is "^my/^dm/^yc ^Hd^99v.9MH ^xxxxza^xxxda".

ARGUMENTS

time_string
: indicates the date_time about which information is desired. If you supply no time_string, the current date and time are used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-language STR, -lang STR
: STR specifies the language in which month name, day names, and zone names are to be expressed. (Default: the process default)

-zone STR
: STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default language and zone. Use the display_time_info command to display a list of all acceptable language and zone values.

Due to exec_coms, etc., that have been built around the expected date_time format, this command does not honor the process date_time format (set by set_time_default). You are encouraged to use "clock date_time" in place of date_time to get the proper default handling.

Name: date_time_after, dtaf

SYNTAX AS A COMMAND

```
dtaf A B {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dtaf A B {-control_arg}]
```

FUNCTION

returns "true" if the date-time A is later than the date-time B.

ARGUMENTS

A and B
: indicates the date and time about which information is desired (see Section 1 for a description of valid time_string values).

CONTROL ARGUMENTS

-date, -dt
: compares only the date portions of A and B as represented in GMT. If the day that A falls on is after the day that B falls on, the value returned is "true".

Name: date_time_before, dtbe

SYNTAX AS A COMMAND

```
dtbe A B {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dtbe A B {-control_arg}]
```

FUNCTION

returns "true" if the date-time A is earlier than the date-time B.

ARGUMENTS

A and B
: indicates the date and time about which information is desired (see Section 1 for a description of valid time_string values).

CONTROL ARGUMENTS

-date, -dt
compares only the date portions of A and B as represented in GMT. If the day that A falls on is before the day that B falls on, the value returned is "true".

Name: date_time_equal, dteq

SYNTAX AS A COMMAND

```
dteq A B {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dteq A B {-control_arg}]
```

FUNCTION

returns "true" if the date-time strings A and B are equivalent.

ARGUMENTS

A and B
indicates the date and time about which information is desired (see Section 1 for a description of valid time_string values).

CONTROL ARGUMENTS

-date, -dt
compares only the date portions of A and B as represented in GMT. If A and B fall on the same day, the value returned is "true".

Name: date_time_interval, dti

SYNTAX AS A COMMAND

```
dti {time_string1} time_string2 {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dti {time_string1} time_string2 {-control_args}]
```

FUNCTION

returns the difference between two date values, relative to the first, in offset terms: "0 yr 0 mo -2 da -6 hr 0 min -4.64 sec". You are able to specify that the result be only in terms of certain units.

ARGUMENTS

time_string1
is the beginning of the interval. If not specified, the current time is used (see "Notes").

time_string2
is the end of the interval. If the end is earlier than the beginning, all numbers are preceeded by a minus sign (see "Notes").

CONTROL ARGUMENTS

-brief, -bf
specifies that the units displayed are in the abbreviated form (Default).

-fractional_digits {N}, -fd {N}
specifies the maximum number of fractional digits to be included on the smallest unit. The value being formatted is rounded to the number of digits specified. All trailing zeros are removed and then the decimal point if it is last. N can't exceed 20. If you supply no N, the maximum is used. (Default: 2)

-zero_units, -zu
specifies that all units are output even if their value is zero (e.g., "2 da 0 hr 0 min 4.2 sec".

-language STR, -lang STR
STR specifies the language in which the result is to be expressed. This can be in any of the languages known to the date/time system. If STR is "system_lang", the system default is used. If you choose no -language or it is present with STR being "", the per-process default is used. Use the display_time_info command to obtain a list of acceptable language values.

-long, -lg
specifies that the units displayed are in the singular/plural form.

-no_zero_units, -nzu
specifies that any unit that has a value of zero are not included in the output; however if all units are zero, the smallest is shown with the value of "0". Example: "2 da 4.2 sec". (Default)

-units STRs
specifies that the result is to be expressed in terms of a given set of units. All arguments following -units on the command line are taken as the set of units to use; therefore make -units, if given, the last control argument. You can enter the units in any language available on the site and in any order. All units, however, must be in the same language. These are the units that you can specify: year, month, week, day, hour, minute, second, and microsecond. The output appears in that order.

NOTES

When you specify no units, this set is used: years, months, days, hours, minutes, seconds. A default result could look like this: "-2 da -6 hr -4.05 sec"; but if the arguments given were: -fd -units hr min, the same interval could be: -54 hr -0.0676252166666666666 min. Note that there is a truncation in the first instance to two decimal places with the corresponding loss of accuracy.

See Section 1 for a description of valid time_string values.

Name: date_time_valid, dtv

SYNTAX AS A COMMAND

```
dtv time_string
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dtv time_string]
```

FUNCTION

returns "true" if the argument is a valid date-time string, "false" otherwise.

ARGUMENTS

time_string
is a string that is checked. Since the concatenation of all the arguments is checked, the argument need not be quoted if it contains white space. (See Section 1 for a description of valid time_string values.)

EXAMPLES

```
! dtv foo

returns "false".

! format_line [dtv 12/31/83 8pm]

returns "true".
```

Name: day

SYNTAX AS A COMMAND

```
day {time_string} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[day {time_string} {-control_arg}]
```

FUNCTION

returns a one- or two-digit number of a day of the month, from 1 to 31. The format string to produce this is "^Z9dm".

ARGUMENTS

time_string
: indicates the day about which information is desired. If you supply no time_string, the current day is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-zone STR
: STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default zone. Use the display_time_info command to display a list of all acceptable zone values.

*

Name: day_name

SYNTAX AS A COMMAND

```
day_name {time_string} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[day_name {time_string} {-control_args}]
```

FUNCTION

returns the full name of a day of the week for a specified date or the current date. The format string to produce this is "^dn".

ARGUMENTS

time_string
: indicates the day_name about which information is desired. If you supply no time_string, the current day_name is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-language STR, -lang STR
: STR specifies the language in which month name, day names, and zone names are to be expressed. (Default: the process default)

-zone STR
: STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default language and zone. Use the display_time_info command to display a list of all acceptable language and zone values.

Name: debug, db

SYNTAX AS A COMMAND

debug

FUNCTION

is an interactive debugging aid to be used in the Multics environment. It allows you to look at or modify data or code. You can stop execution of a program and examine its state by inserting "breakpoints" in the program before and/or during execution. A concise syntax for user requests, coupled with a complete system of defaults for unspecified items, allows you to make many inquiries with little effort. Symbolic references permit you to retreat from the machine-oriented debugging techniques of conventional systems and to refer to variables of interest directly by name.

The debug command uses a segment in the home directory to keep track of information about breaks. This segment is named Person_id.breaks, where Person_id is your login name. The break segment is created if not found. If the segment cannot be created, the break features of debug are disabled and unusable.

Users who do not need the sophisticated machine level debugging provided by this command should refer to the probe command in this manual.

With the debug command you can

Look at data or code;

Modify data or code;

Set a break;

Perform (possibly nonlocal) transfers;

Call procedures;

Trace the stack being used;

Look at procedure arguments;

Control and coordinate breaks;

Continue execution after a break fault;

Change the stack reference frame;

Print machine registers;

Execute commands.

These functions are provided by two types of debug requests: data requests and control requests. The first five functions above are performed by data requests; the others, by control requests. Multiple debug requests (either data or control) can be placed on a line separated by semicolons (;).

NUMBER REPRESENTATION CONVENTIONS

Debug uses both octal and decimal representation of numbers. In general, machine-dependent numbers such as pointers, offsets, and registers are assumed to be octal, while counting arguments (e.g., specifying a source line number, printing the first 20 lines) and variables referenced by name are assumed to be decimal.

A decimal default can be changed to octal by preceding the number with the escape sequence "&o". An octal default can be changed to decimal by preceding the number with "&d".

Example	*Description*
x = 8	assign the value 8 to the program variable x. Program variables referenced by name are assumed to be decimal; if octal representation is preferred, type: x = &o10
$q = 77	assign the value of 77 to the q-register. Register values are machine dependent and assumed to be octal; if decimal representation is preferred, type: $q = &d63
/test/&a19	print line 19 of the source segment for test.
&a19,s8	print 8 source lines, beginning at line 19.

DATA REQUESTS

Data requests consist of three fields and have the following format:

```
<generalized address> <operator> <operands>
```

The generalized address defines the actual data or code of interest. It is ultimately reduced to segment number and offset by debug before being used. The operator field indicates to debug which function to perform, e.g., print or modify the data referenced by the generalized address. The operands field may or may not be necessary, depending on the operator. When these fields are specified, they are separated by blanks or commas.

When debug decodes a data request, it parses the generalized address and generates a pointer to the data being referenced. This pointer, called the working pointer, is changed whenever the generalized address is changed. It points into either the working segment, its stack frame, or its linkage section. The actual segment depends on the most recent specification in a generalized address. The form for a generalized address is as follows:

```
/<segment name>/<offset><segment ID><relative offset>
```

where each of the four fields is optional. The segment name is either a pathname, a reference name, or a segment number, and defines what is called the working segment. The segment ID specifies which of the data bases associated with the working segment is to be used in setting the working pointer. The segment ID can be one of the following:

&s refers to the stack frame if the working segment is a procedure segment with an active stack frame.
&l refers to an active linkage section (i.e., one with an entry in the linkage offset table (LOT) for your ring).
&t refers to the working segment itself.
&a refers to the source program for the working segment.
&p refers to the parameters of an active invocation of a procedure.
&i refers to an active internal static section (i.e., one with an entry in the internal static offset table (ISOT) for your ring).

The offset field is used as an offset within the segment referenced by the working pointer. For the working segment, this offset is relative to the base of the segment. If the working pointer points into an active stack frame, the offset is relative to the base of that frame. If the working pointer points into an active linkage section, the offset is relative to the beginning of that linkage section.

The offset can be either a number or a symbolic name. If a symbolic name is specified, a symbol table must exist for the working segment. See the translator commands for descriptions of symbol table creation. If a symbolic name begins with a numeric character, the escape characters &n (for name) must precede the name, to avoid interpreting the name as a number. For example:

```
/test/&n10&t
```

can be used in a debug request to specify the location associated with FORTRAN line number (i.e., label) 10.

The relative offset field allows you to relocate the working pointer by a constant value or register. For example, if you wish to reference the fourth word after the stack variable i, you could use

```
/test/i+4
```

as the generalized address. The relative offset can also assume the value of a register. For example, if the a-register contains the value 4 at the time of a break, then:

```
/test/100&s$a
```

sets the working pointer to offset 104 from the base of the stack frame. It is important to note that a + sign is not present when a register is used. (See "Registers" below.)

The three most common values for the segment ID field are &t, &s, and &l. These designate that the working pointer is to refer to, respectively, the working segment itself, its active stack frame, or its active linkage section. In addition, two other possible values of segment ID allow alternate methods of referring to locations in either the working segment or its stack frame.

A segment ID of &a refers to the ASCII source program for the working segment. Associated with this segment ID is a decimal line number, which must immediately follow the &a. This line number is used to generate a working pointer to the first word of code compiled for that line. A relative offset can follow the line number. Note that the line-number/code-location association can only be determined if a symbol table exists for the working segment. This example:

```
/test_seg/&a219+36
```

generates a working pointer that points to the thirty-sixth (octal) word in the text after the first word of code generated for line 219 in the source for the segment test_seg. If an offset field is given before &a, the offset is ignored. The offset of the working pointer is generated solely from the line number and the relative offset.

A segment ID of &p refers to the parameters of an active invocation of a procedure. If the current defaults specify an active stack frame, a number following the &p specifies the parameter that is to be addressed. The offset field is ignored, but a relative offset can be specified. This example:

```
/test_seg/&s;&p4+36,a14
```

causes the stack frame for test_seg to be the working segment, and the first 14 characters of the data contained at a location 36 words after the beginning of the fourth parameter are printed in ASCII format.

It is not necessary to specify all four fields of a generalized address. In fact, every field is optional. If a field is not specified, a default value is assumed that is frequently the last value that the field had. For example:

```
/test_seg/line&s+3
```

followed by the generalized address

```
+4
```

is acceptable. The latter request is equivalent to

```
/test_seg/line&s+7
```

One time that the defaults assumed are not the values of the previous data request is when a symbolic variable name or label is specified that causes some field to change. If this is the case, debug might recognize that the segment ID, for example, of the previous data request is not valid and set it appropriately. For example:

```
/test_seg/760&s
```

followed by

```
regp
```

would cause the defaults to be changed to

```
/test_seg/140&l
```

if regp is found at a relative offset of 140 (octal) in the linkage section. Note that the segment ID is changed to &l where it remains until explicitly or implicitly changed again.

Defaults are also reset to values different from the previous values when the segment name field is specified in a generalized address. In this case, the following actions are taken:

1) If the segment name begins with &n, take the rest of the characters composing the segment name and go to step 3 below, treating the string as a name. This convention allows the use of debug on segments whose names are composed of numeric characters.

2) If the segment name is really a segment number, this number is used in a search of all active stack frames to see if one exists for this segment. The search is from the highest stack depth (deepest in recursion) to the base of the stack so that if an active stack frame is found, it is the one most recently used. If an active stack frame is found, the generalized address defaults are set as follows:

working segment	the one specified by the given segment number.
offset	zero.
segment ID	&s, i.e., the working pointer points into the latest stack frame for the working segment.
relative offset	zero.

   If no active stack frame is found, the defaults are set as above except that the segment ID is &t instead of &s, i.e., the working pointer points into the working segment itself.

3) If the segment name is a reference name known in this ring, the segment number for the segment being referenced is found, and then the defaults are calculated as if this segment number were given directly.

4) If the segment name is a pathname, the specified segment is initiated (it can already have been known) and the returned segment number is used as above.

5) If the segment name is of the form segname$entname, the stack is searched from the highest active frame (as in step 2) for the most recent frame associated with the entry point entname in the segment segname. The working segment becomes segname, and the remaining defaults are set as described in step 2.

The entire set of defaults that apply to a debug data request can be determined at any time by issuing the .d control request to print defaults. For the format and use of this request, see the description under "Control Requests" below.

OPERATOR FIELD OF DATA REQUESTS

After decoding the generalized address and determining the working pointer, debug checks the operator. The following five operators are recognized:

, (comma)	print
=	assign
<	set a break
>	alter program control (i.e., "go to")
:=	call a procedure

If a debug request is terminated before an operator is encountered either by a semicolon or a newline character, the default operator used is ",", i.e., print. The one exception is that a blank line is ignored. The first, second, and fifth operators above have operands.

PRINT REQUEST

For the print request, there are three optional operands. They are a single character specifying the output mode desired; a number indicating how much output is being requested; and a number in parentheses indicating the size of the output. The size has two meanings that are dependent on the output mode being used.

1) If the mode is comp-8 or comp-5, the size is the number of digits plus the sign, if present.

2) If the mode is not comp-8 or comp-5, the size is the number of bits to use in printing one item.

The size specification is permitted for the following modes: o, h, d, e, f, p, comp-5, comp-8. It is ignored for the following modes: i, l, a, b, comp-6, comp-7. All of the arguments are optional and spaces can appear between arguments. For example:

```
142&s,o(18)12
```

requests that 12 (decimal) half words starting at 142 (octal) in the stack be printed in octal format.

The following output modes are available for print requests (see "Output Modes" below for a full description):

a	ASCII	f	floating point
b	bit string	fl	long floating point
comp-5	COBOL	g	graphic
comp-6	COBOL	h	half-carriage octal
comp-7	COBOL	i	instruction
comp-8	COBOL	l	code for line number
d	decimal	n	no output
e	floating point with exponent	o	octal
		p	pointer
el	long floating point with exponent	s	source statement

The request

```
+36,a14
```

requests that 14 (decimal) characters starting at 36 (octal) words after the current working pointer be printed in ASCII format. The output might be

```
1416 1416 ">user_dir_dir>"
```

The two numbers printed in most output modes should be interpreted as follows:

1) If the data is from a stack frame, the first number is the relative offset from the base of the stack segment and the second number is the relative offset within the stack frame. If the second number is negative, the variable does not exist in the current stack frame and is a parameter or a global variable.

2) If the data is from a linkage section, the first number is the offset within the combined linkage segment and the second number is the offset within the linkage section.

3) For all other segments, both numbers are the same and represent the offset within the segment.

If a mode is not specified for output, the last specified mode is used unless debug realizes another mode is more appropriate (e.g., when a symbol specifies a variable of a different type). If the amount of output is not specified, it is assumed to be one unit, i.e., one word for octal output, one line for source output, one character for ASCII output, etc.

ASSIGN REQUEST

When modifying data or code, the operands (at least one is expected) specify the new values to use. For example:

```
i = 8; p(1) = 206|10, 206|32
```

assigns the decimal value 8 to i and the values 206|10 and 206|32 to p(1) and p(2), respectively. (It is assumed that both are variables that are defined for the current working segment.) If more than one operand is specified in an assignment request, consecutive words starting at the working pointer are changed. This is illustrated by the assignment to the pointer array p.

There are nine acceptable forms for assignment operands:

1. octal number
2. decimal number
3. character string
4. register value (see "Registers" below)
5. instruction format input
6. floating point number
7. pointer
8. bit string
9. variable

Whether a number is assumed to be octal or decimal on input depends on the target. A variable referenced by name is assumed to be decimal unless overridden by "&o". Assignment to a location specified by offset is assumed to take an octal value unless overridden by "&d".

```
x = 99 (decimal)
+2 = 77 (octal)
```

Character strings being input must be bracketed by quote characters ("). Bit strings being input must be bracketed by quote characters and followed by a b. Floating point numbers must not have exponents.

The word-offset portion of a pointer value being input can optionally be followed by either a decimal bit offset in parentheses, a ring number in square brackets, or both. If both a bit offset and a ring number are specified, the ring number must follow the bit offset, with no intervening blanks. For example:

```
p = 206|25(29); q = 252|104[5]; rp = 211|200(3)[4]
```

The format for instruction input is

```
(opcode address,tag)
```

The address can specify a base register or a number. For example:

```
/test/lab2 = (lda pr6|20) (sta pr0|2,*0) (nop 0)
```

Some value must be given for the address field. The zero opcode is specified by the opcode arg.

Input of bit strings and character strings changes only those bits or characters specified, i.e., a full word might not be completely changed.

Several types of input can be interspersed in the same assignment request. For example:

```
/145/13000 = "names" &d16 126
```

When different types of input are specified in one request, you should be aware that the bit offset of the temporary working pointer might be ignored for certain types of input. In the example above, the ASCII for "name" is placed at 145|13000 and the ASCII for "s" is placed in the first character position of 145|13001. The next assignment argument (&d16) fills in 145|13001 with the decimal 16 and hence overwrites the "s" of the previous argument.

In order to better specify more complicated assignments, a repetition factor is provided. If a single number (decimal) appears in parentheses in an assignment, the next data item is assigned repeatedly (i.e., the specified number of times), updating the working pointer each time. An example of this is

```
string = (32)" " "alpha"
```

which results in string being modified so that the first 32 (decimal) characters are blanks, and the 33rd through the 37th contain the string "alpha".

SET BREAK REQUEST

A breakpoint is a special modification to the code of a program that, when executed, causes control to pass to debug. You are then free to examine and change the states of variables, set other breaks, continue execution, etc. When setting a break, the working pointer is used directly unless it points into the stack. In that case, the working pointer is temporarily forced to the text. To set a break at the label

loop_here in the program parse_words, type:

```
/parse_words/loop_here<
```

You can also type

```
/parse_words/loop_here+23<
```

to set the breakpoint 23 (octal) locations after the first word of code for the statement labelled loop_here in the text segment.

You can also set a break by specifying a line number. For example:

```
/rand/&a26<
```

sets a break at the first word of code generated for line 26 (decimal) of the source program.

The break number printed by debug when setting a breakpoint is used as the name of the break when referring to breaks. After a break is reset, the break number is reused. (Resetting a break restores the code to its previous value.)

Once a break has been set at a given location, another break cannot be set there. The list breaks control requests .bl and .bgl can be used to find out which breaks are set.

ALTER PROGRAM CONTROL REQUEST

To alter program control by issuing an explicit transfer, type:

```
/216/2176>
```

causing debug to search the stack for an active stack frame for the segment 216 (octal) and set the stack pointer to this frame. It then transfers to 2176 (octal) in the text associated with this stack frame.

If no active stack frame is found, debug prints a message and waits for further requests.

CALL A PROCEDURE REQUEST

You can cause debug to call a specified procedure and return values into specified locations. This is done by specifying := as the operator in a data request. This operator expects one operand that is a procedure name with its associated arguments. There are two slightly different ways to invoke this feature: first, to invoke a procedure as a function call (with the argument n+1 being the returned value); and second, to explicitly call a procedure. When a procedure is invoked as a function reference, the current working pointer is used as the last argument in the argument list and, hence, the procedure returns a value into wherever the working pointer is pointing. For example:

```
/test/fi := sqrt_(2.0)
```

causes the sqrt_ function to be called with the first argument 2.0 and the return argument of fi; debug converts the 2.0 into a floating point number before the call. If no fields are present before the := is encountered, debug does not specify a return argument in the call. (The := can be thought of as "call" in a PL/I program.) For example:

```
:= who
```

sets up a call to who$who with no arguments. The call

```
:= rename ("foo","moo")
```

and

```
..rename foo moo
```

are functionally equivalent. (See Multics command execution under "Control Requests" below.)

The method debug uses in setting up the call is to use ten temporary storage areas, one for each of ten possible arguments. debug converts the arguments appropriately and stores the values in these areas. Each area starts on an even location and consists of eight words. These temporary storage areas can be looked at or altered with standard data requests. They are named %1, ..., %10. For example:

```
:= cpu_time_and_paging_(0,0,0)
%1,d
%2,d
%3,d
```

prints three decimal numbers, all being return values from hcs_$usage_values. The actual call that debug made had three arguments that were all 0. (The first words of the first three storage areas were zeroed out prior to the call.) The above call can also be made as follows:

```
%3 := cpu_time_and_paging_(0,0)
```

If this is done, the third argument is not zeroed before the call.

Variables can also be used as arguments. For example:

```
sum := sqrt_(n)
```

No conversion is done by debug if n is fixed and sqrt_ expects a floating argument.

The above mentioned temporaries can be used to do simple mode conversion. For example, to get the floating point representation of 3.7 (in octal), type:

```
%1 = 3.7; ,o
```

To find the ASCII value for 137 (octal), type:

```
%1 = 137137137137 ; ,a4
```

A reference to one of these storage areas causes the working segment to be changed to the stack segment.

If one of the arguments in a procedure call is the character %, the temporary storage for that argument is not changed (e.g., overwritten with the usual argument value). Results from some previous work can be passed in that argument position. For example:

```
%2 := sqrt_(2.0)
:= ioa_("^e",%)
```

REGISTERS

The hardware registers at the time of a fault (in particular a break fault) are available to you for inspection or change. These registers are referenced by preceding the register name immediately by a dollar sign ($). The register can be looked at by merely typing the register name. For example:

```
$a
```

prints the contents of the a-register at the time of the last fault. The value in the a-register can be changed to octal 146 by typing

```
$a = 146
```

Decimal input is allowed also:

```
$a = &d19
```

The value in a pointer register, the tpr, or the ppr can be set to a pointer:

```
$pr0 = 254|1173
$ppr = 512|0
```

The value assigned must always be a pointer.

No assignment may be made to the registers regs, ind, scu, all, or prs. The register name must always be given on the input line. You are warned that the working pointer is not set or referenced by register operations.

The predefined register names used by debug are

a	a-register.
all	all machine conditions.
aq	the a- and q-registers considered as a single register.
eaq	the exponent, a- and q-registers in floating point format.
even	even instruction of Store Control Unit (SCU) data.
exp	exponent register.
ind	indicator register.
odd	odd instruction of SCU data.
ppr	procedure pointer register.
prN	pointer register N where N can be 0 through 7.
prs	all pointer registers.
q	q-register.
ralr	ring alarm register.
regs	all registers x0, a, q, aq, exp, tr, and ralr.
scu	all SCU data.
tpr	temporary pointer register.
tr	timer register.
xN	index register N where N can be 0 through 7.

You can change the above registers at will (with the exception of ind and eaq) with the understanding that if execution continues after the break or transfers directly (via > in a data request), the values of the hardware registers are set to those of the above registers.

The values in the registers are automatically filled in by debug (when it is called or faulted into) with those values associated with the last fault found in the stack. You can override these values with the fill registers (.f) and crawlout registers (.C) control requests. See "Control Requests" below.

You can also define registers and use them as a small symbolic memory. For example:

```
$sta1 = 600220757100; $nop = 11003
```

allows you to later specify

```
/test/210&t = $sta1 $nop $nop
```

To print out the contents of all user-defined registers, type:

```
$user
```

The setting and displaying of registers follows the syntax of data requests. However, only the register name and a possible new value can appear in a register request. Registers can be specified in a general data request only in the relative offset field and as operands in assignment requests. Register names must be less than or equal to four characters in length. Some examples of the use of registers follow:

```
/test/i =$q
/test/0 = $x0
/test/46$x0,a5
```

CONTROL REQUESTS

Control requests provide you with useful functions not necessarily related to any specific data. The format for a control request is

```
.<request name>
```

Control requests and data requests can be freely mixed on a command line if separated by semicolons. However, certain control requests use the entire input line and hence ignore any semicolons found therein. Spaces are not allowed in most control requests.

The following is a list of all control requests and the functions they perform. See "Summary of Data and Control Requests" below for a complete review of all requests.

Trace Stack

The general form is `.ti,j`

The stack is traced from frame i (counting from 0 at the base of the stack) for j frames, where i and j are decimal integers. If i is less than 0, tracing begins at 0; if i is greater than the last valid frame, then only the last frame is traced. If i is not specified, it is assumed to be 0; if j is not specified, all valid stack frames from i on are traced. The name printed in the stack trace is the primary segment name unless the segment is a PL/I or FORTRAN program in which case it is the entryname invoked for the stack frame (i.e., the label on the entry or procedure statement).

Examples:

```
.t2,3
.t100
```

Pop Or Push Stack

The general form is `.+i` or `.-i`

The working segment is changed by moving up or down the stack i frames, where i is a decimal integer. For example, if the working segment's active stack frame is at depth 4 in the stack, then:

```
.+3
```

changes the working segment to the segment whose stack frame is at depth 7 in the stack. The defaults for working pointer, segment ID, and offset are reinitialized to the base of the stack frame, &s, and 0, respectively.

Set Stack

The general form is `.i`

The working segment is set to that of stack frame i (starting at 0), where i is a decimal integer. The defaults are set as in pushing or popping the stack.

Execute Multics Command

The general form is `..<Multics command line>`

The input line is interpreted as a standard Multics command line and is passed to the standard command processor with any preceding characters blanked out. Any valid Multics command line can be given. When setting breaks, the program being debugged must be called in this manner because debug sets up a condition handler (for break faults) that is active only as long as debug's stack frame is active.

Print Defaults

The general form is `.d` or `.D`

The output might look like

```
.vm 6,3,2,20

3 /test_seg/14(0)&t,i 212
```

or

```
3 />udd>m>foo>test_seg/14(0)&t,i 212
```

The first number (3 above) is the stack frame depth in decimal, unless there is no stack frame for the working segment, in which case the number is -1. The name of the working segment appears between the slashes (test_seg above); if `.D` is used, the full pathname occurs here. The offset appears next (14 above); the bit offset (in decimal) of the working pointer appears next; the segment ID (`&t` above) appears next; the operator appears next (, for print); the output mode appears next (i for instruction); finally the segment number of the working segment appears (212 above). To find the name/segment number association for a given segment, for example segment number 206, type:

```
/206/,n;.d
```

yielding

```
60 /test_caller/0(0)&s,o 206
```

Knowing the name, you can obtain the same output by typing

```
/test_caller/,n;.d
```

Continue Execution After a Break

The general forms are `.c,i .ct,i .cr,i`

If i is not specified, it is assumed to be 0. If i is specified, the next i break faults for the current break are skipped. The first instruction executed upon continuation is the instruction on which the break occurred. If a t follows the c, debug continues in temporary break mode (see "Break Requests" below). If an r follows the c, debug resets the mode to normal (not temporary).

Examples:

`.c`	continue execution.
`.c,3`	continue execution, but skip the next three break faults for the current break.
`.ct`	continue execution in temporary break mode.

Quit

The general form is %.fnt typ%.q%.fnt%

This request returns from debug to its caller. Note that if debug was entered via a break, typing `.q` returns to the last procedure that explicitly called debug.

Change Output Mode

Requests pertaining to debug's terminal output begin with `.m`.

1) Enter brief output mode: `.mb`

This request places debug in brief output mode, which is somewhat less verbose than its normal output mode. In particular, assignment requests and the resetting of breaks are not acknowledged on your terminal; the column headings are not printed for a stack trace; the printing of register contents is somewhat more compact; some error messages are abbreviated.

2) Enter long output mode: `.ml`

This returns debug to long output mode, which results in fuller and more explicit terminal output. Long mode is the initial default.

Set I/O Switch Names

These requests allow you to debug a program that is run with file output because it generates extensive output or a program that is run from within an exec_com after `&attach` because it requires much input. The general form is

```
.si switch_name
.so switch_name
```

where switch_name identifies the switch_name to use for input (.si) or output (.so). The named switch must be attached by you before the request is made. If no switch name is given, debug creates one (either debug_input or debug_output).

1) User makes a switch request but does not give a switch name:

```
.si
.so
```

debug creates a switch named debug_input or debug_output and attaches it to the user_i/o switch. This is the usual request for debugging programs that require the user_input or user_output switches to be attached to a file instead of to user_i/o. Debug detaches the debug_input and debug_output switches when you quit debug.

2) User makes a switch request and gives the switch name:

```
.si input_switch
.so output_switch
```

You must attach the switch_name before making the request. This can be used when you want to read debug requests from a file. The switches can be restored by typing

```
.si user_input
.so user_output
```

Examples:

You have directed the output switch named user_output to a segment, but wants debug diagnostics to be printed on the terminal. This can be done by typing

```
debug
.so
```

Since a switch name is not given with the request, debug sets up a new I/O switch named debug_output as a synonym for user_i/o, which is the terminal in this case. When you quit debug, the switch named debug_output is detached.

You want to debug a procedure that uses the user_input switch and has a set of debug requests in another segment named debug_macro. An input switch named macro has been attached to the segment of debug requests. You type

```
debug
.si macro
```

and debug takes requests from the switch named macro and does not detach the switch when you exit debug. An attempt by debug to read beyond the end of the macro input stream results in an exit from debug.

Break Requests

The following control requests are specific to breaks and begin with `.b`. Reference is made to the default object segment, which is merely that segment that debug is currently working with when performing break requests. The default object segment is generally specified implicitly when a break is set or hit. It can be changed and determined upon request. The default object segment used for break requests is not necessarily the same as the segment addressed by the working pointer used in data requests.

Breaks are numbered (named) sequentially starting at 1 but the numbers are unique only for the object segment in which the break resides. You can have several breaks with the same number defined in different object segments.

There are two types of global requests that can be performed on breaks. The first, or subglobal requests, refer to all breaks within the default object segment. The second, or global requests, refer to all breaks set by you (as determined from the break segment in the home directory). The subglobal request is specified by omitting the break number in a break request. The global request is specified by a "g" immediately after the "b" of all break requests (see below).

The general form of all break requests is `.bgxi args`

where the "g", the number i, and the arguments are optional. The "x" is replaced by the control character for the break request desired. The following break requests are currently defined:

1) Reset a break (or breaks). The forms of the requests are

`.bri` to reset break i of the default object segment.
`.br` to reset all breaks of the default object segment.
`.bgr` to reset all breaks known to debug.

2) List (print information about) a break. The forms of the request are

.bli to list break i of the default object segment.
.bl to list all breaks of the default object segment.
.bgl to list all breaks known to debug.

3) Execute a debug request at break time. The forms for this request are

```
.bei   <rest of line>
.be    <rest of line>
.bge   <rest of line>
```

Specifying the above request causes <rest of line> to be interpreted as a debug input line whenever the appropriate break(s) is encountered. If <rest of line> is null, the specified breaks have this execute feature reset to normal.

4) Disable a break (or breaks). The forms of this request are

.boi disable (turn off) break i of the default break segment.
.bo disable all breaks in the default break segment.
.bgo disable all breaks known to debug.

Disabling a break has the effect of preventing the break from being taken without discarding the information associated with it. You can disable a break rather than reset it if the break is to be needed again in the future. A disabled break can be eliminated altogether (reset) by the .br request, or reenabled by the .bn request. If the break has already been disabled, these requests have no effect.

5) Enable a break or breaks. The forms of this request are

.bni enable (turn on) break i of the default break segment.
.bn enable all breaks in the default break segment.
.bgn enable all breaks.

This request restores a previously disabled break. If the break was not disabled, the request has no effect.

6) Establish a temporary command line to be executed whenever breaks are encountered. This request is of the form:

```
.bgt <rest of line>
```

This causes <rest of line> to be executed as a debug request whenever any break is encountered during the current process. The difference between this request and .bge is that when .bge is typed, the associated line remains associated with all breaks until they are reset, or until they are changed by .be requests. It is possible to have a temporary global command without removing request lines associated with individual breaks. If <rest of line> is null, a previously-established temporary command line is disestablished.

7) Break conditionally. The following requests allow you to change a break into a conditional break, i.e., a break that stops only if a certain condition is met.

```
.bci arg1 <rel> arg2
.bc arg1 <rel> arg2
```

arg1 and arg2 can be constants or variables; <rel> can be = or ^=. Whenever a specified break is encountered, a test is made to see if the equality exists and breaks according to whether you specified = or ^= in setting up the conditional break. For example:

```
.bc3 i ^= 0
```

causes break 3 to fault whenever it is encountered and the value of i is nonzero. Another example:

```
.bc3 i = j
```

causes break 3 to fault whenever it is encountered and the value of i is the same as the value of j. The comparison is a bit by bit comparison with the number of bits to compare being determined by the size and type of the second argument.

If no arguments are given to a set conditional request, the specified break is set back to a normal break. For example:

```
.bc
```

causes all breaks of the default object segment to fault normally.

8) Specify the number of times a break should be ignored (skipped). The general form is

```
.bsi N
```

This causes the number of skips to be assigned to break i of the default object segment to be N.

9) Print or change the default object segment. The form for this request is

```
.bd name
```

where name is the (relative) pathname, reference name, or segment number of the segment to become the default object segment. If name is not specified, the pathname of the default object segment is printed.

10) List the current segments that have breaks. The form for this request is

```
.bp
```

This request merely interprets the break segment in the initial working directory.

Print Arguments

The general form is `.ai,m`

Argument i for the current stack frame is printed in the mode specified by m. If i is not specified, all arguments are printed. If m is not specified, debug decides the output mode. Valid values for m are

```
o      full word octal
p      pointer
d      decimal
a      ASCII
b      bit string
l      location of argument
e,f    floating point
?      debug decides (the default value for m)
```

Examples:

```
! .a3
  ARG 3:  ">user_dir_dir"
! .a3,o
  ARG 3: 076165163145
```

Get Fault Registers

The general form is `.f`

For register requests debug uses the machine registers of the last fault found in the stack starting at the frame currently being looked at. (This is the default when debug is entered as a result of a break fault.)

Crawlout Registers

The general form is `.C`

For register requests debug uses the fault data associated with the last crawlout (abnormal exit from an inner ring).

PROGRAM INTERRUPT FEATURE

You can interrupt debug by pressing the quit button at any time, in particular during unwanted output. To return to debug request level (i.e., to where debug waits for a new request), type:

```
! pi
```

which is the standard program interrupt manager. (See the description of the program_interrupt command.)

TEMPORARY BREAK MODE

When debug is in temporary break mode (placed there via a `.ct` control request), the following actions are taken automatically:

1) When you continue any break, another (temporary) break is set at the first word of code for the next line of source code after the source statement containing the break being continued. If debug cannot determine the location of the next line of source code, the temporary break is set at the word of object code immediately following the break being continued.

2) A temporary break is restored automatically whenever it is continued. A temporary break must be explicitly reset by you only when it is not continued.

Since temporary breaks are set sequentially in a program (i.e., at the next statement in the source program), any transfers within a program can either skip a temporary break or cause code to be executed that was stopped earlier with a temporary break. Temporary break mode is designed to be used in programs that are fairly uniform and sequential in their flow of control. You should list breaks after using temporary break mode to see if any breaks remain active.

INDIRECTION

It is quite often desirable to reference the data pointed to by the pointer that is pointed to by the working pointer, i.e., to go indirect through the pointer. You can instruct debug to do this by typing * instead of the segment name, offset, and segment ID in a generalized address. For example:

```
/test/regp
```

might print

```
1260 110 214|2360
```

To find what two octal words begin at 214|2360, type:

```
*,o2
```

This causes the working pointer to be set to 214|2360 and not necessarily point into the same segment as before the request.

IMPLEMENTATION OF BREAKPOINTS

Breakpoints are implemented by using a special instruction (mme2) that causes a hardware fault whenever it is executed. When the fault is first encountered in a process using the standard process overseer, a static handler for the fault is set up that passes control to debug. When debug is entered via a break, it does the following:

1) fills the registers with those of the break fault;

2) prints the location of the break fault;

3) waits for requests.

When continuing after a break fault, debug changes the control unit information so that when it is restarted, it executes the instruction that used to exist where the break word was placed.

The debug command keeps track of a default object segment. All break requests made are relative to the default object segment. For example, any reference to break 3 really means break 3 of the default object segment. To change (or find out) the value of the default object segment, the `.bd` request should be used.

VARIABLE NAMES FOR PL/I AND FORTRAN PROGRAMS

If a symbol table was created for a PL/I or FORTRAN program using the table option, then names of labels, scalars, structures, and arrays can be used. The only restrictions are

1) that the entire structure name must be specified;

2) the only expressions that are allowed for subscripts are of the form:

```
variable +/- constant
```

where `variable` can be an arbitrary reference as above;

3) all subscripts must appear last. If a variable is based on a particular pointer, that pointer need not be specified. Some examples of valid variable references are

```
p-> a.b.c(j,3)
a.b
p(3,i+2)  -> qp.a.b(x(x(4)+1))->j.a
```

BIT ADDRESSING

When a working pointer is generated to a data item that is based on or is a part of a substructure, a bit offset may be required. This bit offset is indeed kept and used. When making references to data relative to a working pointer with a bit offset, the relocated addresses can still contain a bit offset. For example, if the working pointer

has the value

```
151|3706(13)
```

then the request

```
+16,b3
```

sets the working pointer to

```
151|3724(13)
```

and prints the three bits at this location.

OUTPUT MODES

The following output modes are acceptable to debug:

o The data pointed to by the working pointer is printed in full word octal format, eight words per line.

h Half carriage octal: the data is printed as in o format except that only four words per line are printed.

d The data is printed in decimal format, eight words per line.

a a
The data is interpreted as ASCII and printed as such. No more than 256 characters are printed in response to a single request.

i i
The data is printed in instruction format.

p The data is printed in pointer format, i.e., segment number and offset (and bit offset if nonzero).

s One or more source statement lines are printed starting with the line of source code that generated the code pointed to by the working pointer (assumed to be pointing into the text). For example:

```
/test/loop_here+32,s2
```

prints two lines of source code starting with the line that generated the code, 32 (octal) words after the label loop_here. Another example:

```
/test/&a219,s
```

prints line number 219 (decimal) of test.lang where lang is the appropriate language suffix. Note that if there was no code generated for the specified line, debug prints a message, increments the line number, and tries again for up to 10 lines.

l The code associated with the specified line number is printed in instruction format. The line number is determined as in s type output. For example:

```
/test/&a27,l
```

prints the code generated for line 27 (decimal) of test.lang.

n No output. This is used to suppress output when changing defaults.

e Floating point with exponent (single precision)

el Long floating point with exponent (double precison).

f Floating point (single precision).

fl Long floating point (double precision).

b The data is printed as if it were a bit string. No more than 72 bit positions are printed in response to a single request.

g The specified number of characters are interpreted as Multics standard graphics code. The type and value of each recognizable item is printed to the terminal. (Refer to the *Multics Graphics System Manual*, AS40, for details.)

comp-5, comp-6, comp-7, comp-8
The data is printed as if it were a COBOL data type. If the size field is used for comp-5 or comp-8, it is the number of digits plus sign to use in printing the data.

```
comp-5 byte-aligned packed decimal
comp-6 full-word binary integer
comp-7 half-word binary integer
comp-8 digit-aligned packed decimal
```

Name: decat

SYNTAX AS A COMMAND

```
decat strA strB C
```

SYNTAX AS AN ACTIVE FUNCTION

```
[decat strA strB C]
```

FUNCTION

performs operations on bit or character strings. These operations are specified by a three-digit bit string given last in the usage line.

ARGUMENTS

strA, strB
: are character strings or bit strings entered as 0 and 1 characters.

C
: is any three-digit bit string expressed as 0 and 1 characters such as 000,001,...,111.

NOTES

The first occurrence of strB found in strA divides strA into three parts: part prior to strB, part matching strB, and part following strB. The digits given in C correspond to these three parts. The return string contains the parts of strA whose corresponding bit in C is 1. All three parts are returned in their original order of appearance in strA.

EXAMPLES

```
! decat abcdef123ghi 123 110
  abcdef123
! string [decat decat.incl.pll .incl 101]
  decat.pll
```

Name: decimal, dec

SYNTAX AS A COMMAND

```
dec values
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dec values]
```

FUNCTION

returns one or more values in decimal.

ARGUMENTS

value
: is a value to be processed. The last character of the value indicates its type. Acceptable types are binary (b), quartenary (q), octal (o), hexadecimal (x), or unspec (u).

Any valid PL/I real value is allowed. The absence of any specifier means decimal. The unspec value is limited to eight characters.

EXAMPLES

```
! dec 110.1b
  6.5

! string [dec abcu]
  25478243
```

Name: decode

SYNTAX AS A COMMAND

```
decode path1A {path2A ...  path1N path2N} {-control_arg}
```

FUNCTION

reconstructs an original segment from an enciphered segment according to a key that is not stored in the system. The encode command is used to encipher segments.

ARGUMENTS

path1A
is the pathname of an enciphered segment. The code suffix should not be specified because the command attaches the code suffix to the path1 argument (e.g., if you type alpha_x.code as the path1 argument, the command attaches the suffix and looks for a segment named alpha_x.code.code). The star convention is allowed.

path2A
is the pathname of the deciphered segment to be produced. If the last path2 argument is missing, the command constructs a pathname from the path1 argument (see "Notes" below). The equal convention is allowed.

CONTROL ARGUMENTS

-key STR
specifies the encipherment key STR on the command line and does not query for one. This control argument is useful in exec_com's for multiple invocations of the command with the same key.

NOTES

The decode command requests the key from the terminal only once. All segments specified in an invocation of decode are deciphered with the same key.

If the last path2 argument is not given, the command places the deciphered segment in a segment whose name is the path1 argument, minus the code suffix.

EXAMPLES

If you type the command line

```
! decode alpha_x
```

the command looks for an enciphered segment named alpha_x.code and places the deciphered segment produced in a segment named alpha_x.

Name: decode_access_class, dac

SYNTAX AS A COMMAND

```
dac STR
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dac STR]
```

FUNCTION

decodes a character string produced by the encode_access_class command or the convert_access_class_$encode subroutine to return the authorization or access class.

ARGUMENTS

STR
is the encoded access class string to be decoded. The null string is converted to the string "system_low."

Name: default

SYNTAX AS A COMMAND

```
default STRA {STRB}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[default STRA {STRB}]
```

FUNCTION

supplies default arguments to commands and can override this default when desired. Use this command with the abbrev and do commands.

NOTES

If you provide no STRB or it is the null string, STRA is returned (see "Examples").

EXAMPLES

In the first example, you set up an abbreviation using the default active function to automatically compile a program with the -map and -table control arguments. You can override the defaults by specifying more than one argument when using the abbreviation. Assume that comp_pl1 is an abbreviation for

```
do "pl1 &1 [default ""-map -table"" &2] &f3"
```

Thus typing "comp_pl1 test" is the same as typing "pl1 test -map -table"; typing "comp_pl1 test -list -profile" is the same as typing "pl1 test -list -profile".

The next example shows the null input feature of the default active function. Assume that my_dp is an abbreviation for

```
do "dp -he [string [default [entry &1] &r2]]
   -q [default 3 &3] &f4 &1"
```

When you type the command line

```
! my_dp >udd>Demo>Bach>design_memo.runout "" 2 -dl
```

the null input for the second argument means that default uses the default value for this argument (in this case, the entryname portion of the pathname). Thus the expansion of the command line is

```
dp -he design_memo.runout -q 2 -dl >udd>Demo>Roy>design_memo.runout
```

Name: default_wdir, dwd

SYNTAX AS A COMMAND

```
dwd
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dwd]
```

FUNCTION

returns the pathname of the default working directory of the process in which you invoke it, as set by the change_default_wdir command.

Name: defer_messages, dm

SYNTAX AS A COMMAND

```
dm {mbx_specification}
```

FUNCTION

suspends printing of messages.

ARGUMENTS

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

LIST OF MBX SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table.

STR
is any noncontrol argument and is first interpreted as -mailbox STR; if no mailbox is found, STR is then interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

NOTES

Deferred messages stay in your mailbox until you issue print_messages. The immediate_messages command restores printing of messages as you get them.

For a description of mailbox creation and characteristics see accept_messages.

Name: delete, dl

SYNTAX AS A COMMAND

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

FUNCTION

deletes the specified segments, multisegment files (MSFs), data management (DM) files, and/or extended entries. Use delete_dir to delete directories, unlink to delete links.

ARGUMENTS

paths
: are the pathnames of segments, MSFs, DM files, or extended entries. You can use the star convention.

CONTROL ARGUMENTS

-absolute_pathname, -absp
: prints the entire pathname of entries listed by -lg, -qya, and -qye.

-brief, -bf
: does not print an error message if a segment, MSF, or DM file to be deleted is not found.

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

-entryname, -etnm
: prints only the entrynames of the entries listed by -lg, -qya, and -qye, rather than the entire pathname. (Default)

-force, -fc
: deletes the specified entries, whether or not they are protected, without querying.

-interpret_as_extended_entry, -inaee
: interpret the selected entry as an extended entry type.

-interpret_as_standard_entry, -inase
: interpret the selected entry as a standard entry type.

-long, -lg
: prints a message of the form "Deleted file <path>" for each entry deleted.

-name STR, -nm STR
specifies a nonstandard entryname STR (e.g., an invalid star name such as **.**.compout or a name containing <).

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

-query_all, -qya
lists all segments to be deleted and queries whether they should be deleted or not. Unless you give -fc, an individual query is given for protected segments.

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

ACCESS REQUIRED

You must have modify permission on the containing directory.

NOTES

You must supply at least one path or -name STR.

To delete a segment or MSF the entry must have both its safety switch and its copy switch off. If either is on, you are interrogated whether to delete the entry.

You can't delete DM files if a transaction is still pending.

Name: delete_acl, da

SYNTAX AS A COMMAND

```
da path {User_ids} {-control_args}
```

FUNCTION

removes entries from the access control lists (ACLs) of nonlink entries in a directory. (For a description of ACLs see the Programmer's Reference Manual.)

ARGUMENTS

path
is the pathname of an entry. If it is -working_directory (-wd), your working directory is assumed. The star convention is allowed.

User_ids
are access control names of the form Person_id.Project_id.tag. All ACL entries with matching names are deleted. If you give no User_ids, your 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 star name.

-directory, -dr
affects only directories. (Default: segments, multisegment files, and directories)

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

-segment, -sm
affects only segments and multisegment files.

-select_entry_type STR, -slet STR
affects only entries of the entry type selected by STR, which is a comma-delimited list of file system entry types. Use the list_entry_types command to obtain a list of valid entry type values. Example: da ** -slet mbx,segment.

ACCESS REQUIRED

You need modify permission on the containing directory.

Name: delete_dir, dd

SYNTAX AS A COMMAND

```
dd {paths} {-control_args}
```

FUNCTION

deletes the specified directories and any segments, links, multisegment files, data management files, and extended entries they contain. All inferior directories and their contents are also deleted. Use the delete command to delete segments and the unlink command to delete link entries.

ARGUMENTS

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

CONTROL ARGUMENTS

-absolute_pathname, -absp
prints the entire pathname of the entries listed by -long, -query_all, and -query_each.

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

-entryname, -etnm
prints only the entrynames of the entries listed by -long, -query_all, and -query_each. (Default)

-force, -fc
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 entryname STR which begins with a hyphen or contains ASCII control characters or any of the nonstandard characters ", <, >, $, %, ?, *, =, (,), [,], ::.

-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. (Default)

ACCESS REQUIRED

You must have modify permission on both the directory and its superior directory.

NOTES

At least one path or -name must be given.

If -force is not supplied, delete_dir asks you whether to delete the specified directory; it is then deleted only if you type "yes." When deleting a nonempty master directory, or a directory containing inferior nonempty master directories, you 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.

When you are deleting a directory containing data management files, you can't delete those files if a transaction is still pending.

Name: delete_external_variables, dev

SYNTAX AS A COMMAND

```
dev names {-control_arg}
```

FUNCTION

deletes from your name space specified variables managed by the system for you. All links to those variables are unsnapped and their storage is freed.

ARGUMENTS

names
are the names of the external variables, separated by spaces, to be deleted.

CONTROL ARGUMENTS

-unlabeled_common, -uc
indicates unlabeled (or blank) common.

Name: delete_iacl_dir, did

SYNTAX AS A COMMAND

```
did path {User_ids} {-control_args}
```

FUNCTION

deletes entries from a directory's initial access control list (initial ACL) in a specified directory. A directory initial ACL contains the ACL entries to be placed on directories created in the specified directory. (For a description of initial ACLs, see the Programmer's Reference Manual.)

ARGUMENTS

path
specifies a pathname of the directory whose directory initial ACL should be changed. If path is -working_directory (-wd) or omitted, your working directory is assumed. The star convention is allowed.

User_ids
are access control names of the form Person_id.Project_id.tag. All entries in the directory initial ACL that match the User_ids are deleted (for a description of the matching strategy, see the set_acl command). If you give no User_ids, your Person_id and current Project_id are assumed.

CONTROL ARGUMENTS

-all, -a
deletes the entire directory initial ACL except an entry for *.SysDaemon.*.

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

-ring N, -rg N
identifies the ring number whose directory initial ACL is to be deleted. It can appear anywhere on the line and affects the whole line. If present, follow it by N (where 0 <= N <= 7). If not given, your ring is assumed.

ACCESS REQUIRED

You must have modify permission on the directory.

EXAMPLES

The command line

```
! did news .Faculty Dickinson..
```

deletes from the directory initial ACL of the news directory all entries ending in .Faculty.* and all entries with Person_id Dickinson.

The command line

```
! did -a
```

deletes all entries from the directory initial ACL of the working directory.

The command line

```
! did store Emerson -rg 5
```

deletes the entry for Emerson.*.* from the ring 5 directory initial ACL of the store directory.

Name: delete_iacl_seg, dis

SYNTAX AS A COMMAND

```
dis path {User_ids} {-control_args}
```

FUNCTION

deletes entries from a segment initial access control list (initial ACL) in a specified directory. A segment initial ACL contains the ACL entries to be placed on segments created in the specified directory. (For a discussion of initial ACLs, see "Access Control" in the Programmer's Reference Manual.)

ARGUMENTS

path
: specifies the pathname of a directory whose segment initial ACL is to be changed. If it is -working_directory (-wd) or omitted, your working directory is assumed. The star convention is allowed.

User_ids
: are access control names of the form Person_id.Project_id.tag. All entries in the directory initial ACL that match the User_ids are deleted. (For a description of the matching strategy, see the set_acl command.) If you give no User_ids, your Person_id and current Project_id are assumed.

CONTROL ARGUMENTS

-all, -a
: deletes the entire initial ACL except an entry for *.SysDaemon.*.

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

-ring N, -rg N
: identifies the ring number whose directory initial ACL is to be deleted. It can appear anywhere on the line and affects the whole line. If present, follow it by N (where 0 <= N <= 7). If not given, your ring is assumed.

ACCESS REQUIRED

Your must have modify (m) permission on the directory.

EXAMPLES

The command line

```
! dis news .Multics. JJoyce
```

deletes from the segment initial ACL of the news directory all entries with Project_id Multics and the entry for JJoyce.*.*.

The command line

```
! dis -a
```

deletes all entries from the segment initial ACL of the working directory.

The command line

```
! dis store Hawthorne.. -rg 5
```

deletes all entries with Person_id Hawthorne from the ring 5 segment initial ACL of the store directory.

Name: delete_message, dlm

SYNTAX AS A COMMAND

```
dlm msg_specs {mbx_specification} {-control_args}
```

FUNCTION

deletes any interprocess 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_messages" was in effect.

ARGUMENTS

msg_specs
: are one or more numbers or ranges. Numbers are as printed next to each message when "accept_messages -hold_messages" is in effect. Ranges are of the form N:M, where N<=M and both N and M are valid message numbers. You can use the keywords "first" (f) and "last" (l) as message numbers and the keyword "all" (a) as a range (equivalent to "f:l").

mbx_specification
: specifies the mailbox from which messages are to be deleted. If not given, the user's default mailbox (>udd>Project>Person>Person.mbx) is used.

LIST OF MBX SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table.

STR
is any noncontrol argument and is first interpreted as -mailbox STR; if no mailbox is found, STR is then interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

CONTROL ARGUMENTS

-after time_string
deletes messages sent after time_string only (see "Notes").

-all, -a
delets all messages, including those held by -hold_messages mode (see accept_messages).

-before time_string
deletes messages sent before time_string only (see "Notes").

-brief, -bf
suppresses an error message when no matching messages are found.

-comment STR, -com STR
deletes messages with comment fields containing STR only.

-exclude STR
deletes messages with text not containing STR only.

-force, -fc
deletes selected unseen messages.

-from STR, -fm STR
deletes messages sent from STR only. STR can be of the form Person.Project, Person, or .Project.

-long, -lg
    overrides -brief.

-match STR
    deletes messages with text containing STR only.

-messages, -msg
    deletes regular messages (nonnotifications) only.

-no_force, -nfc
    prevents deletion of unseen messages. (Default)

-no_messages, -nmsg
    suppresses -messages.

-no_notifications, -nnt
    suppresses -notifications.

-notifications, -nt
    deletes notifications only.

NOTES

If you supply no mailbox, your default one is assumed (for a description of the mailbox see accept_messages and print_mail).

See Section 1 for a description of valid time_string values.

Name: delete_name, dn

SYNTAX AS A COMMAND

```
dn {paths} {-control_args}
```

FUNCTION

deletes specified name(s) from segments, multisegment files (MSFs), links, directories, data management (DM) files, or extended entries that have multiple names.

ARGUMENTS

paths
    are the pathnames to be deleted. This argument can be "-name STR" to specify a nonstandard name, such as one beginning with a minus sign or containing * or >. The star convention is allowed, but does not apply to STR.

ARGUMENTS

paths
: are the pathnames of a segment, multisegment file, directory, extended entry, or link. This argument can consist of "-name STR" to specify a nonstandard entryname STR which already exists and which begins with a hyphen or contains ASCII control characters or any of the nonstandard characters ", <, >, $, %, ?, *, =, (,), [,], ::.

CONTROL ARGUMENTS

-brief, -bf
: suppresses error messages when entries are not found with specified pathnames.

-long, -lg
: prints error messages when entries are not found. (Default)

ACCESS REQUIRED

You need modify permission on the parent directory.

NOTES

Specify at least one path or "-name STR." The final portion of the relative or absolute pathname is deleted from the storage system entry it specifies, provided that doing so does not leave the segment or directory without a name, in which case an error message is printed.

See the add_name and rename commands.

EXAMPLES

The command line

```
! dn alpha >my_dir>beta
```

deletes the name alpha from the names of an entry in the current working directory and also deletes the name beta from the names of an entry in >my_dir.

Name: delete_search_paths, dsp

SYNTAX AS A COMMAND

```
dsp search_list search_paths {-control_arg}
```

FUNCTION

allows you 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. Quote it if it contains spaces or other command language characters.

search_paths
specifies a search path to be deleted. The search path must be an absolute pathname. Use the same name that appears when you invoke print_search_paths.

CONTROL ARGUMENTS

-all, -a
specifies that the search list itself is to be deleted. Any search paths specified are ignored. Use -all to delete all the search paths in a search list.

NOTES

For a complete list of the search facility commands see add_search_paths.

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").

NOTES

This command accepts no site-defined keywords and no home_dir and process_dir; add_search_rules does. Even though dsr accepts initiated_segments and referencing_dir, be careful because their deletion may lead to unpredictable results.

Name: delete_volume_quota, dlvq

SYNTAX AS A COMMAND

```
dlvq logical_volume account
```

FUNCTION

deletes a quota account for a logical volume; used by the volume executive (the owner or manager of logical volumes).

ARGUMENTS

logical_volume
: is the name of the logical volume from which quota is to be deleted.

account
: is the name of the quota account (in the form Person_id.Project_id.tag) to be deleted.

ACCESS REQUIRED

To use this command you must have e access to the logical volume. It is not necessary that the volume be mounted.

NOTES

You can't delete the quota account if there are still master directories whose quotas are charged against the account to be deleted. You must either delete such directories or transfer them to another account (see the set_mdir_account command).

describe_entry_type

Name: describe_entry_type, dset

SYNTAX AS A COMMAND

```
dset type {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dset type -control_arg]
```

FUNCTION

prints or returns information about a file system entry type.

ARGUMENTS

type
identifies a storage system entry type. Use the list_entry_types command to obtain a list of entry type values.

CONTROL ARGUMENTS

-all, -a
prints all information about the entry type, which includes name, plural name, access modes, supported attributes, and the default values and all names for switches. You can't use -all in the active function.

-attributes, -attr
prints or returns the names of the storage system attributes that this entry type supports. These are the attributes that can be copied or moved by the copy and move commands.

-default NAME
prints or returns the default value of the specified switch for this entry type. You can give only one -default argument. This control argument is incompatible with -all and -switches.

-extended_acl, -xacl
returns "true" if the entry type supports extended ACLs, "false" otherwise. You can use it only in the active function.

-info_pathname, -ipn
prints or returns the pathname of an info segment containing more information about the entry type, if such an info segment is available.

-modes
prints or returns the acceptable access modes for the specified entry type.

-name, -nm
prints or returns the name of an entry of the selected entry type.

-plural_name, -plnm
prints the plural name of the specified entry type.

-switches
prints the names and default values of all switches supported by the entry type given.

NOTES

When invoked with no control arguments, the command prints the name, plural name, modes, attributes, info seg pathname, switch names and default values.

Name: describe_psp

SYNTAX AS A COMMAND

```
describe_psp Marketing_Identifier Key
```

SYNTAX AS AN ACTIVE FUNCTION

```
[describe_psp Marketing_Identifier Key]
```

FUNCTION

returns various information about priced separate products (PSP) available to Multics sites.

ARGUMENTS

Marketing_Identifier
: is the Honeywell order number of PSPs (e.g., SGL6803 for COBOL). It can be entered in uppercase or lowercase.

LIST OF KEYS

sti
: returns the current software technical identifier for the product released at the installed release.

source
: returns the absolute path of the product's source archive.

object
: returns the absolute path of the product's object archive.

executable
: returns the absolute path of the product's executable segment.

title
: returns the official name by which the product is known.

name
: returns a short descriptive name by which the product is known.

Name: detach_audit, dta

SYNTAX AS A COMMAND

```
dta {switchname}
```

FUNCTION

removes audit_ from the specified switch (see attach_audit and the audit_ I/O module).

ARGUMENTS

switchname
: is the switch from which audit_ is to be removed. If switchname is not specified, user_i/o is assumed.

Name: detach_lv, dlv

SYNTAX AS A COMMAND

```
dlv volume_names
```

FUNCTION

detaches one or more logical volumes that have been attached for your process by the resource control package (RCP).

ARGUMENTS

volume_names
: specifies the volumes to be detached.

NOTES

You can detach all logical volumes by specifying "all", rather than any volume names.

The detaching involves telling the storage system that the logical volume is no longer attached for this process; it does not affect the attached/detached state of the logical volume for any other process.

Name: dial_manager_call

SYNTAX AS A COMMAND

```
dial_manager_call request {STR1 {STR2} {STR3}}
```

FUNCTION

provides a command interface to the answering service's dial facility. All functions that are available through the dial_manager_ subroutine interface are available through this command.

ARGUMENTS

request
maps into a call to an identically named entry in dial_manager_. Each request requires the use of a particular STR, which is listed in the request description. A request must be one of the following:

allow_dials STR, ad STR
requests that the answering service establish a dial line to allow terminals to dial to the calling process. STR must be a dial_qualifier as described below.

dial_out STR1 STR2 {STR3}, do STR1 STR2 {STR3}
requests that an auto call channel be dialed to a given telephone number and, if the channel is successfully dialed, that the channel be assigned to the requesting process. STR1 must be a channel_name and STR2 must be a dial_out_destination as described below. STR3, which can be omitted, is a reservation_string as described below.

privileged_attach STR, pa STR
allows a privileged process to attach any terminal that is in the channel master file, and is not already in use. (See the description of dial_manager_$privileged_attach for information on what access is required.) The effect is as if that terminal had dialed to the requesting process. STR must be a channel_name as described below.

registered_server STR, rs STR
requests that the answering service allow terminals to dial the calling process using only the dial qualifier. STR must be a dial_qualifier as described below.

release_channel STR, rc STR
requests the answering service to release the channel specified by channel_name. This channel must be dialed to the caller at the time of the request. If the channel was dialed, the channel is returned to the answering service and another access request may be issued. If the channel is a slave channel, the channel is hung up. STR must be a channel_name as described below.

release_channel_no_hangup STR, rcnh STR
is the same as release_channel except that this request does not hang up slave channels. STR must be a channel_name as described below.

release_dial_id STR, rdi STR
informs the answering service that your process wishes to prevent further dial connections, but that existing connections should be kept. You can release later any connections kept with the release_channel request. STR must be a dial_qualifier as described below.

shutoff_dials STR, sd STR
informs the answering service that your process wishes to prevent further dial connections and that existing connections should be terminated. STR must be a dial_qualifier as described below.

start_report, start
turns on the reporting feature (see "Notes" below).

stop_report, stop
turns off the reporting feature (see "Notes" below).

terminate_dial_out STR, tdo STR
requests that the answering service hang up an auto call line and unassign it from the requesting process. STR must be a channel_name as described below.

STR
depends on the request. STR is selected from the following list. (For details on the interpretation of the following qualifiers see the dial_manager_ subroutine.)

channel_name
is the name of a tty_channel.

dial_qualifier
is the name for which you are to be a dial server.

dial_out_destination
is the destination (e.g., phone number) of up to 32 characters.

reservation_string
is a dial_manager_ reservation string of up to 256 characters.

NOTES

This command establishes an event call channel for communication with the answering service. This event channel and its handler (which is an entry point in dial_manager_call) remain active after the command terminates. Any events following the command termination, such as channel hang-ups, dial-ups, and dial requests are decoded using convert_dial_message_ and reported on the user_output I/O switch when they happen. You can turn this reporting feature on (the default) and off by using the start_report and stop_report requests.

When the destination specifies an X.25 address you can optionally precede it with "*" or "x29," to indicate that an X.29 (PAD) call should be made. For example, a destination of

```
x29,3106:mitmul or
*3106:mitmul
```

specifies an X.29-type call on TYMNET.

Name: dial_out

SYNTAX AS A COMMAND

```
dial_out channel {destination} {-control_args}

connect channel {destination} {-control_args}
```

FUNCTION

permits you to access a remote system through a dial-out channel.

ARGUMENTS

channel
: is the name of the dial-out or slave channel to be used. The star convention is allowed, which means the answering service selects a channel that has a matching name or generic destination and matching attributes (if specified).

destination
: is the dial-out destination. This string is used when making the connection. If omitted, the channel is priv_attached rather than dialed.

CONTROL ARGUMENTS

-8bit
does not suppress the parity bit of characters from the foreign system.

-abbrev
enables abbreviation processing of request lines.

-brief, -bf
disables printing of informational messages.

-echo
locally echoes characters typed by you.

-escape CHAR, -esc CHAR
makes CHAR the escape character. (Default: !)

-modes MODES
allows you to select the initial values of dial_out's modes (see "List of Modes" below).

-no_start_up, -ns
disables execution of your start_up.dial_out. This is assumed for the connect command.

-profile PATH
establishes PATH as the abbrev profile to be used for request lines. (Default: your current profile).

-request REQUEST, -rq REQUEST
executes the given request before entering its interaction loop. The rightmost -request is the one that is executed. (See "List of Requests" below.)

-resource RSC_DESC, -rsc RSC_DESC
allows you to specify a resource description to be used when attaching the dial_out channel.

-terminal_type TYPE, -ttp TYPE
sets the terminal type of the remote connection to TYPE. You can use this for hosts with unusual communications requirements.

LIST OF REQUESTS

escape CHAR, esc CHAR
sets the escape character.

file_output PATH, fo PATH
starts copying output to a file.

interrupt, int, break, brk, ip
sends an interrupt signal (line break) to the foreign system.

modes STR
allows you to control operational modes.

revert_output, ro
finishes a previous file_output.

send STR
sends its arguments to the foreign system as if they were typed by you.

send_file PATH {-control_arg}
sends the contents of pathname PATH to the foreign system. Any characters sent from the foreign system during this time are discarded; thus foreign echo of the file being sent is rejected. (If the foreign echo is slow, you may still see some echo after this request has finished.)

Control arguments are:

-display_input, -dsin
displays characters sent from the foreign system during the transfer.

-no_display_input, -ndsin
does not display characters sent from the foreign system during the transfer.

switch_name
returns the name of the I/O switch used by dial_out.

wait {STR} {-control_args}, [wait {STR} {-control_args}]
waits for a specified string to come from the foreign system. With no arguments, this request waits until any characters are sent from the foreign system. Invoked with a string and/or -nl, it waits until the specified string is sent.

Control arguments are:

-nl
waits until the specified string is sent with a trailing new line.

-nnl
waits until the specified string is sent but without a trailing new line.

-no_timeout, -ntm
specifies that there is no limit to how much time can elapse between character transmissions from the foreign system before dial_out assumes that the foreign system has finished transmitting. (Default)

-timeout N, -tm N
specifies the maximum number of seconds that can elapse between character transmissions from the foreign system before dial_out assumes that the foreign system has finished transmitting.

As an active request (and if a timeout did not occur), all the text sent from the foreign system since the last text processed by dial_out, including the string specified, is returned as a quoted string.

LIST OF MODES

echo, ^echo
enables/disables local echoing of characters typed by you.

echo_lf, ^echo_lf
enables/disables echoing of LF after you type a CR.

line, ^line
enables/disables line-at-a-time mode as opposed to character-at-a-time mode for your terminal.

quit, ^quit
enables/disables transparent quit mode. In this mode the BREAK key performs an "interrupt" request, rather than the normal Multics function.

raw, ^raw
enables/disables direct transmission to the foreign system of characters typed. (Default: raw)

send_lf, ^send_lf
enables/disables transmission of a LF character as part of the new-line sequence. If you don't set this mode, the new-line sequence consists of CR only; if you set it, it is CR-LF. The new-line sequence is sent when you type the CR key or an NL is encountered in a file.

LIST OF SUBSYSTEM REQUESTS

.
prints a line of requests available in dial_out.

..
executes Multics command lines.

?
prints a list of requests available in dial_out.

abbrev, ab
controls abbreviation processing of requests lines.

answer
provides preset answers to questions asked by another request.

do
executes/returns a request line with argument substitution.

execute, e
executes a Multics command line.

exec_com, ec
executes a file of dial_out requests that can return a value.

help
prints information about dial_out requests and other topics.

if
conditionally executes/returns one of two request lines.

list_help, lh
displays the name of all dial_out info segments on given topics.

quit, q
exits dial_out.

ready, rdy
prints a Multics ready message.

subsystem_name
prints the name of this subsystem.

subsystem_version
prints the version number of this subsystem.

ACCESS REQUIRED

You must have the dialok attribute and rw access to the access control segment for the channel (type "user attributes" to determine what attributes you have).

NOTES

The dial_out command executes your start_up.dial_out exec_com on invocation, whereas the connect command executes an exec_com specific to the given destination (or channel if you give no destination). The .dial_out ec suffix for dial_out and connect exec_coms is assumed if you don't supply it.

The dial_out and connect commands acquire an appropriate communications channel to a foreign system, execute your dial_out start_up and initial request, and then enter an environment in which characters typed on your terminal are sent to the foreign system and in which characters sent from the foreign system become output at your terminal.

When the destination specifies an X.25 address, you can optionally precede it with "*" or "x29", to indicate that an X.29 (PAD) call should be made. For example, a destination of

```
x29,3106:mitmul or
*3106:mitmul
```

specifies an X.29-type call on TYMNET.

After the dial_out environment is entered, you can enter dial_out requests by preceeding them with the escape character (" " by default). Characters from the escape character to the next escape character or the end of the line are interpreted as a request. You must double the escape character to send it to the remote system.

EXAMPLES

```
! dial_out b.h218 9-482-5622 -echo
  Ready on tty_ b.h218 -destination 9-482-5622...

! dial_out tymnet x29,3106:p25
```

Name: directories, dirs

SYNTAX AS A COMMAND

```
dirs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dirs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of directories that match one or more star names.

ARGUMENTS

star_names
: are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
: returns absolute pathnames rather than entrynames. (Default: to return entrynames)

-chase
: processes the targets of links when you specify a starname.

-inhibit_error, -ihe
: returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
: does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
: signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: directory, dir

SYNTAX AS A COMMAND

```
dir path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[dir path]
```

FUNCTION

returns the directory portion of path after it has been expanded into an absolute pathname.

ARGUMENTS

path
is the pathname whose directory portion is to be returned.

NOTES

Since the pathname is returned in quotes, the command processor treats it as a single argument regardless of special characters in the name.

EXAMPLES

In the working directory >udd>Proj>Myname:

```
! dir start_up.ec
  >udd>Proj>Myname

! dir >udd>Multics>Library>Source>bound_command_demos_.s::program.pl1
  >udd>Multics>Library>Source
```

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. You need not quote it.

CONTROL ARGUMENTS

-output_switch STR, -osw STR
where STR is the name of an I/O switch. If you give no -osw, output on the user_output I/O switch is suppressed. You can specify -osw more than once.

NOTES

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

Name: disconnect

SYNTAX AS A COMMAND

```
disconnect
```

FUNCTION

disconnects the terminal from the current process, suspending that process if the user's disconnect_ok process attribute is on. If the trusted path login installation parm is off the terminal will remain dialed to the system. If the disconnect_ok process attribute is on and the trusted path login installation parm is on then a message will be displayed stating the line must be hung up followed by a hangup. If the disconnect_ok process attribute is off then an message will be displayed stating the process cannot be suspended and no further action will take place.

NOTES

If disconnect is called from an absentee or daemon process an error message will be returned without further action.

When the process is suspended after disconnection, it will remain that way for a period of time equal to or less than the site-specified inactivity time. Contact your project administrator for process attribute information.

Name: display_audit_file, daf

SYNTAX AS A COMMAND

```
daf {path} {-control_args}
```

FUNCTION

displays the file produced by the audit_ I/O module.

ARGUMENTS

path
is the pathname of the audit file to be displayed. If path is not indicated, the audit file associated with the user_i/o switch is assumed. If user_i/o is not being audited, the audit file currently in use is displayed.

CONTROL ARGUMENTS

-append_nl, -anl
appends new lines to the end of entries that do not end in a new line. It overrides -no_append_nl to the left in the command line. By default, new lines are appended if a "leader generating" control argument (i.e., -class_identifiers, -entry_numbers, or -metering) is present.

-class STR1 {...STRn}
prints the entries having a class identifier matching any of the STRs. Class identifiers are as follows:

EL, el (edit line)
IL, il (input line)
IC, ic (input characters)
OC, oc (output characters)
TM, tm (trace of modes operations)
TC, tc (trace of control operations)

If STR contains only one character, it is matched on the first character of the class identifiers; for example, if STR is I, entries having either IL or IC class identifiers are displayed. If -class is not given, the audit file is displayed without class identifiers.

-class_identifiers, -cli
displays the audit file with the class identifiers before each entry. If -metering is also specified, the metering information precedes class identifiers.

-entry_numbers, -etn
prints the entry numbers before each entry.

display_audit_file

-exclude STR1 {...STRn}, -ex STR1 {...STRn}
excludes any entries containing strings matching any of the STRs. If -exclude is not chosen, all selected entries are printed.

-from STR, -fm STR
specifies the first audit file entry to be displayed. If STR is a positive integer, it is interpreted as an entry number. If STR is a positive number containing a decimal point, it is interpreted as a time in 24-hour format. If it is neither, the audit file is displayed from the first entry that matches STR. If -from is not supplied, the audit file is displayed from the beginning.

-insert_nl, -inl
inserts new lines whenever an entry is over length (as determined by -line_length or the current line length for the switch). (Default)

-last STR
displays entries beginning at the point specified by STR, starting at the end of the audit file. If STR is in entry number format, the first entry displayed is STR entries back from the end of the file. If STR is in time format, the first entry is STR hours and minutes from the end of the file. If STR is a character string, the first entry contains a match for STR searching from the end of the file. If -last is not selected, the audit file is displayed from the beginning.

-line_length N, -ll N
inserts a new line after the character specified by N if a line of output is greater than N characters long. A continuation line is indented to allow for any entry descriptors produced by -metering, -entry_numbers, or -class_identifiers and is preceded by an "*" to indicate it is a continuation of the previous line.

-match STR1 {...STRn}
prints entries containing strings matching any of the STRs. If it is not specified, all selected entries are printed.

-metering, -mt
displays the audit file with metering information at the beginning of each line, preceding the class identifiers if -class or -class_identifiers is also used.

-next STR
displays a given number of entries from an explicit point in the file to the point specified by STR. If STR is in entry number format, the next STR entries are displayed. If STR is in time format, the entries within the next STR period of time after the beginning entry are displayed. If STR is a character string, the entries up to the next match of STR are displayed. If -next is not selected, all entries to the end of the file are displayed.

-no_append_nl, -nanl
does not append new lines to entries that do not end in a new line. This control argument overrides the appending of new lines because of "leader generating" control arguments or an occurrence (to the left in the command line) of -append_nl.

-no_insert_nl, -ninl
    does not insert new lines.

-output_file path, -of path
    displays the audit file into the segment named path.

-reverse
    prints the entries in reverse chronological order.

-string STR, -str STR
    uses STR as a character string with no special interpretation. This is useful for preventing STR from being interpreted as a control argument, a time, or an entry number. It can be given with -from, -to, -next, -last, -match, and -exclude, for example, "-from -string 81" (see "Notes").

-switch STR
    displays the audit file associated with the I/O switch specified by STR if the I/O switch is currently attached. If the I/O switch is not attached, an error message is printed. If -switch is not chosen, the audit file associated with the user_i/o switch is displayed.

-to STR
    stops the display of the audit file at the point specified by STR, where STR can have any of the values for -from. If -to is not specified, the audit file is displayed up to the end.

NOTES

You specify the format of the output, which entries are selected to be output, and the file to which the output is directed (see attach_audit and the audit_ I/O module).

The audit_meter mode must be on for there to be any metering information in the audit file; without this information, time arguments are invalid.

The -string control argument is useful in the following situations. To pass 1005.2 as a character string to be matched, rather than a time value for -from, type:

```
! daf -from -string 1005.2
```

To pass -last as a character string to -match, type:

```
! daf -match -string -last
```

Name: display_cobol_run_unit, dcr

SYNTAX AS A COMMAND

```
dcr {-control_args}
```

FUNCTION

displays the current state of a COBOL run unit.

CONTROL ARGUMENTS

-all, -a
prints information about all programs in the run unit, including those that have been cancelled.

-files
displays information about the current state of the files that have been referenced during the execution of the current run unit.

-long, -lg
displays more detailed information about each COBOL program in the run unit.

NOTES

The minimal information displayed tells which programs compose the run unit. Optionally, more detailed information can be displayed concerning active files, data location, and other aspects of the run unit. Refer to the run_cobol command for information concerning the run unit and the COBOL runtime environment.

See also the stop_cobol_run and cancel_cobol_program commands.

Name: display_component_name, dcn

SYNTAX AS A COMMAND

```
dcn path offsets
```

FUNCTION

converts an offset within a bound segment (e.g., bound_zilch_|23017) into an offset within the referenced component object (e.g., comp|1527). This command is especially useful when it is necessary to convert an offset within a bound segment (as displayed by a stack trace) into an offset corresponding to a compilation listing.

ARGUMENTS

path
is the pathname of a bound object segment or an octal segment number. You can specify a pathname that looks like an octal segment number by -name nnn.

offsets
are octal offsets within the text of the bound object segment specified by path.

EXAMPLES

The command line

```
! dcn bound_zilch_ 17523 64251
```

might respond with the following lines:

```
17523      component5|1057
64251      component7|63
```

If bound_zilch_ were known with segment number 532, the following command would generate the same output:

```
dcn 532 17523 64251
```

Name: display_entry_point_dcl, depd

SYNTAX AS A COMMAND

```
depd virtual_entry
```

SYNTAX AS AN ACTIVE FUNCTION

```
[depd virtual_entry]
```

FUNCTION

prints a PL/I declare statement describing a procedure entry point or other, system-wide external data constant. The command obtains the entry point declaration from data files that contain the declarations of all unusual procedure entry points (ALM segments, procedures written as subroutines but called as functions, etc.) and system-wide external data values (e.g., sys_info$max_seg_size, iox_$user_output, etc.) or from argument descriptors for the entry point parameters that are included with the procedure entry point itself.

display_entry_point_dcl

The active function returns the declaration as a quoted string.

ARGUMENTS

virtual_entry
a character string representation of the procedure entry point or external data value whose declaration is to be printed (see Section 1 for a description of virtual entries). You can use the archive component pathname convention.

NOTES

Most command and active function entry points do not declare arguments in their procedure statements since they accept a variable number of arguments. They also do not use the options(variable) attribute in their procedure statements. Therefore, when the depd command encounters a procedure entry point with no declared arguments and without options(variable), it assumes the options(variable) attribute required for commands and active functions and prints a declare statement of the form:

```
dcl help entry options(variable);
```

depd distinguishes between such assumed options(variable) entries and those that explicitly use the options(variable) attribute in their procedure statement by printing "entry" for the assumed case and "entry()" for the explicit case. Thus, it prints a declaration for depd, which explicitly uses options(variable), as

```
dcl depd entry() options(variable);
```

For procedures that use structures as arguments, certain structure declarations are inexactly returned as parameter declarations because the mechanism for encoding argument descriptors does not provide an adequate description of the alignment of a structure. The descriptor only determines whether the overall structure is packed or not and does not specify whether or not it was originally declared with the aligned attribute.

The following structures generate the same argument descriptors, even though PL/I treats the level 1 structures as having different attributes:

```
dcl 1 s2 structure aligned,
      2 e11 fixed bin aligned,
      2 e12 fixed bin aligned;

dcl 1 s2 structure,
      2 e11 fixed bin aligned,
      2 e12 fixed bin aligned;
```

depd reproduces the declaration for s2 when either s1 or s2 are used as parameters for an entry point. To bypass this problem, declare the subroutine properly in your personal .dcl segment (see "User-provided Data Files" below) and place this segment in your "declare" search paths.

NOTES ON SEARCH LIST

The depd command uses the "declare" search list, which has the synonym "dcl", to find data files describing unusual procedure entry points. The default search list identifies the data file:

```
>sss>pl1.dcl
```

For more information about search lists, see the descriptions of the search facility commands and, in particular, the add_search_paths command.

USER-PROVIDED DATA FILES

You can provide data files that redeclare standard system entry points (e.g., redeclaring a subroutine as a function) or that declare their own entry points or external data items. Data items have the general form of

```
virtual_entry declaration
```

For example:

```
ioa_ entry options(variable)
```

Note that the word "dcl" is not included in the data item, nor does the declaration end with a semicolon (;). External data values are declared in a similar fashion. For example:

```
iox_$user_output ptr external static
```

Name: display_mailing_address, dsmla

SYNTAX AS A COMMAND

```
dsmla {names}
```

FUNCTION

displays the specified mail table entries with default mailing address(es), which appear in the format used in message headers displayed by read_mail. In addition, if the mail table entry specifies an ACS segment to allow other maintainers to update it, this pathname is displayed.

ARGUMENTS

names
are the Person_ids or aliases of the user whose mailing address should be displayed, or the names or aliases of a mail table entry for a forum or mailing list. The command displays the mailing address for each one (printing a warning message for invalid ones). If you give none, the default is your Person_id.

Name: display_pl1io_error, dpe

SYNTAX AS A COMMAND

```
dpe
```

FUNCTION

describes the most recent file on which a PL/I I/O error was raised and displays diagnostic information associated with that type of error. This command is designed to be invoked after the occurrence of an I/O error signal during a PL/I I/O operation.

EXAMPLES

The command line

```
! dpe
```

might respond with the following display:

```
Error on file afile
Title: vfile_ afile
Attributes: open input keyed record sequential
Last i/o operation attempted: write from
Attempted "write" operation  conflicts with file "input" attribute.
Attempted "from" operation conflicts with file "input" attribute.
```

Name: display_pnotice

SYNTAX AS A COMMAND

```
display_pnotice name {control_arg}
```

FUNCTION

displays information on software protection notices contained in source programs.

ARGUMENTS

name
is the full or relative pathname of the source language module. The language suffix or the archive suffix must be included if an entire archive is to be processed. The archive pathname convention is supported, but the star convention is not.

CONTROL ARGUMENTS

-brief, -bf
specifies that the primary name of notices, without the "pnotice" suffix, be printed instead of the text of notices found. (Default)

-long, -lg
displays the full text of notices found.

NOTES

By default, the primary names of protection notices are printed instead of the entire notice text. If path includes the full archive name, then archives of source code programs can be audited for protection notices. If a source module does not contain any notices, or contains conflicting notices (copyright and trade secret), an error message is displayed. A warning message is also displayed if there is an embedded notice found in a source program (protection notices should be the first comment encountered).

EXAMPLES

```
! display_pnotice add_pnotice.pl1
  add_pnotice.pl1: HIS.1982

! display_pnotice add_pnotice.pl1 -lg
  Notices in add_pnotice.pl1:

Copyright, (C) Honeywell Information Systems Inc., 1982

! display_pnotice farf.pl1
  Warning: farf.pl1 has no protection notice.
```

Name: display_subsystem_usage

SYNTAX AS A COMMAND

```
display_subsystem_usage subsystem_name {-control_args}
```

FUNCTION

displays usage information recorded by a subsystem.

ARGUMENTS

subsystem_name
is normally the name of the subsystem whose usage information is to be displayed (see "Notes on Subsystem Usage Segments" below).

CONTROL ARGUMENTS

-first N
prints only the first N entries; when combined with -reverse, prints only the last N entries. The entries are sorted, if requested, before application of -first. It is incompatible with -totals.

-header, -he
prints a header defining each column of output. (Default)

-no_header, -nhe
suppresses printing of the header.

-no_reverse, -nrv
prints the entries in the selected order. (Default)

-reverse, -rv
prints the entries in reverse order from that selected.

-sort TYPE
sorts the individual entries before displaying them. It is incompatible with -totals. (See "List of Sorting Types" below.)

-totals, -tt
prints only the total number of invocations of the subsystem without listing any individual entries. It is incompatible with -first and -sort.

-user NAME
prints only those entries for users whose Person_id matches NAME. The star convention is allowed.

-version VERSION
prints only those entries for users who last used the version of the subsystem named by VERSION.

LIST OF SORTING TYPES

The TYPE given to -sort must be one of the following:

count
sorts by the total number of invocations of the subsystem by the user.

date_time_used, dtu
sorts by the date-time of the last invocation of the subsystem by the user.

name
sorts by the Person_id of the user whose usage is recorded in this entry.

version
sorts by the version number of the subsystem's last version used by the user.

NOTES

The information displayed by this command for a user of the subsystem includes

1. the user's Person_id.
2. the total number of times the user has used this subsystem.
3. the version number of the last version of the subsystem used by the user.
4. the number of times the user has used this version of the subsystem.
5. the date-time the user last used the subsystem.

NOTES ON SUBSYSTEM USAGE SEGMENTS

Subsystem usage information is recorded in the segment named "subsystem_name.ssusage" (e.g., read_mail.ssusage), and you can locate that segment by using the linker search rules. If you can't locate it, you can give the pathname of the subsystem usage segment, with or without the ssusage suffix, as an argument to this command in addition to the subsystem name.

Name: display_time_info, dsti

SYNTAX AS A COMMAND

```
dsti -control_args
```

FUNCTION

displays information selected from time_info_.

CONTROL ARGUMENTS

-all, -a
specifies all data are to be printed.

-day
asks for a list of all the day names.

-format, -fmt
asks for a list of all keywords that can be given in a time_format control string. This list does not include "date", "date_time", and "time" as they are not contained in time_info_. Use print_time_defaults to see them.

-language, -lang
asks for a list of all the time languages available, showing the name of each language in each language. You would usually use this form alone to enable you to see what languages you can refer to.

-language STR, -lang STR
asks for the output to be given in language STR. (Default: to show requested data in the process default language)

-map
asks for a time zone map of the world, with all the defined time zones and their offsets. Each zone is at its proper place on this map. The map is horizontally broken according to the line length currently in effect.

-month
asks for a list of all the month names.

-offset
asks for all the offset words to be printed.

-table STR, -tb STR
STR specifies the pathname of the table to be displayed. (Default: the reference name "time_info_")

-token {STR}
displays the structure used for binary, searching the tokens declared in the table. The display shows all words, with their meanings, in all languages, grouped by token. A token is a word converted to lowercase. If you give STR, only the data for that token is shown. Since STR represents a token and not a word, enter it in lowercase.

-word
asks for all the miscellaneous words to be printed.

-zone
asks for a list of all the zones available.

Name: display_ttt

SYNTAX AS A COMMAND

```
display_ttt {-control_args}
```

FUNCTION

prints all or part of a terminal type table (TTT) on your terminal, or outputs it to a file. The output's format is such that you can use it as a terminal type file (TTF).

CONTROL ARGUMENTS

-header, -he
prints a header (see "Notes").

-no_header, -nhe
suppresses printing of the header.

-output_file PATH, -of PATH
directs output to the file whose pathname is PATH. If you omit the ttf suffix from PATH, it is added. If you omit PATH, output is directed to your terminal.

-pathname PATH, -pn PATH
displays the TTT whose pathname is PATH. If you omit the ttf suffix from PATH, it is assumed. If you omit PATH, the process's current TTT is displayed.

-table NAME, -tb NAME
displays only the conversion, translation, function keys, or special table named NAME (see "Notes").

-terminal_type NAME, -ttp NAME
displays only the terminal type entry for the terminal type named NAME (see "Notes").

NOTES

If you give neither -ttp nor -tb, the entire contents of the TTT are displayed. If you give either -ttp or -tp, only the specified terminal type entry or table is displayed, without the introductory comment, unless you also give -he. If you give no -nhe, an introductory comment is printed, giving the pathname of the TTT, the date, and the User_id of the author of the original TTT.

Name: divide

SYNTAX AS A COMMAND

```
divide numA numB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[divide numA numB]
```

FUNCTION

returns the integer part of the decimal quotient of numA divided by numB (see the mod and quotient commands).

EXAMPLES

```
string [divide 5 4]
1
```

Name: dm_display_version

SYNTAX AS A COMMAND

```
dm_display_version
```

FUNCTION

displays the version of the data management (DM) system software currently in use by the executing process.

Name: dm_user_shutdown

SYNTAX AS A COMMAND

```
dm_user_shutdown
```

FUNCTION

removes the process invoking it from the current invocation of the data management system (DMS).

NOTES

All user process references to per-process and per-system data are invalidated to permit subsequent reentry to DMS. If a transaction is in progress in the process when you issue the command, the DM Daemon (Data_Management.Daemon) rolls it back automatically.

Normally all processes using data management are shut down as part of a data management system shutdown, with no explicit user intervention.

This command is part of the command level interface to Multics data management. Use it in a test environment or in debugging.

Name: do

SYNTAX AS A COMMAND

```
do {-control_args} {control_string {args}}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[do control_string {args}]
```

FUNCTION

substitutes arguments into a control string. The expanded control string is then passed to the command processor or the subsystem request processor for execution. As an active function or active request, returns the expanded control string without executing it.

do

ARGUMENTS

control_string
is a character string that can contain substitution constructs (see "List of Substitutions" below).

args
are zero or more character string arguments. Any argument supplied but not referenced by an argument substitution designator is ignored.

CONTROL ARGUMENTS

If you give control arguments with no control string, subsequent do invocations in the process are affected; with a control string and its arguments, subsequent do invocations are not affected. Give the control arguments first. (See "Notes on modes" below.)

-abort_line, -abl
aborts the line containing the do request if the request line is aborted during execution. Applies only to subsystem request invocations of do. (Default)

-brief, -bf
does not print the expanded control string. (Default)

-control_string, -cs
permits a control string to look like a control argument.

-go
passes the expanded control string to the command processor or subsystem request processor. (Default)

-inhibit_error, -ihe, -absentee
establishes a handler for the any_other condition during the execution of the expanded command control string.

-long, -lg
prints the expanded control string on error_output before executing or returning it.

-no_abort_line, -nabl
continues execution with the next request following the do request on the invoking line if the request line do invoked is aborted during execution. Applies only to subsystem request invocations of do.

-no_inhibit_error, -nihe, -interactive
does not catch any signals. (Default)

-nogo
does not pass the expanded control string to the request processor.

do

LIST OF SUBSTITUTIONS

The following expansion designators appearing in the control string are replaced by their expansion value, as described below. Any other use of the ampersand (&) produces an error.

&0, &1,...&9
expands to the zeroth through ninth arguments. &0 is the control string, &1 is the first argument following the control string, and so on. If the corresponding argument is missing, the designator expands to a null string.

&(0), &(1),...
expands to any argument, including arguments after the ninth. Use parenthesis when the argument number is two or more digits. If the corresponding argument is missing, the designator expands to a null string.

&q0,...&q9, &q(0), &q(1),...
expands to the corresponding argument following the control string. Quotes within the argument are doubled, according to the quote depth of the surrounding context within the control string (see "Notes on Quote Doubling" below).

&r0,...&r9, &r(0), &r(1),...
expands to the corresponding argument following the control string, enclosed in an added layer of quotes with internal quotes with the argument doubled accordingly (see "Notes on Requoting" below). This designator keeps the argument as a single unit after one layer of quote stripping by the command processor.

&f1,...&f9, &f(1),...
expands to the Nth through last arguments following the control string, with arguments separated by one space. If N is greater than &n, expands to a null string.

&qf1,...&qf9, &qf(1),...
expands to the Nth through last arguments following the control string, with quotes doubled within arguments, and arguments separated by one space. If N is greater than &n, expands to a null string.

&rf1,...&rf9, &rf(1),...
expands to the Nth through last arguments following the control string, with each argument individually requoted, and arguments separated by one space. If N is greater than &n, expands to a null string.

&n
expands to the number of arguments you give following the control string.

&f&n, &qf&n, &rf&n
expands to the last argument following the control string, with quotes doubled (&qf&n) or with requoting (&rf&n).

do

&control_string
expands to the control string (without expansions), with quotes doubled. It is equivalent to &q0.

&!
expands to a unique name. Each use of &! is replaced by a 15-character identifier. Every use within a single invocation is replaced by the same string, but the string is different for every invocation of do.

&&
expands to a single ampersand, to allow ampersands to be literally inserted into the expanded control string.

NOTES

When the control string is executed, abbreviations are expanded if the abbrev processor is enabled. Since the control string is usually enclosed in quotes, abbreviations in the control string are not expanded until control string expansion. (See the abbrev command.)

NOTES ON MODES

This command has four modes: the long/brief mode, the nogo/go mode, the abort-line mode, and the inhibit-error mode. These modes are kept in internal static storage and are thus remembered from one invocation of do to the next in a single process. Set the modes for the life of the process by invoking do with control arguments and no control string; set the modes for a single invocation by giving control arguments, a control string, and its arguments.

The abort-line mode applies only to subsystem request invocations of do. You can set the mode at command level, but cannot set it for a single command invocation of do.

Use the inhibit-error mode mainly in an absentee environment, in which any condition that normally enters a new command level terminates the process. In this mode, any signal caught by do terminates execution of the command line, not the process. The following conditions are not handled by do, however, but are passed on to the command processor: command_error, command_query_error, command_question, program_interrupt, quit, and record_quota_overflow (see the Programmer's Reference Manual).

The abort-line, inhibit-error, and go/nogo modes have no effect on active function and active request invocations of do.

NOTES ON QUOTE DOUBLING

Each parameter designator to be expanded is found nested a certain level deep in quotes. If it is found to be outside quotes, its quote level is zero; if found between a single pair of quotes, its quote level is one; and so on. If an "&q" construct is found nested to quote-level L, then, as the argument is substituted into the expanded control string, each quote character found in the argument 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 control string. If the "&q" construct is not between quotes, or if the corresponding argument contains no quotes, quote doubling has no effect.

NOTES ON REQUOTING

If an "&r" construct is found, the substituted argument is placed between an additional level of quotes before having its quotes doubled. For example, if &r1 is found nested to quote level L, 2**L quotes are inserted into the expanded control string; then, the first argument is substituted, with each of its quotes replaced by 2**(L+1) quotes; and, finally, 2**L more quotes are placed following it. If you give no argument, nothing is placed in the expanded control string; so, you can distinguish between arguments that are not supplied and arguments that are supplied but are null. If you give an argument, the expansion of an "&r" construct is identical to the expansion of an "&q" construct surrounded by an extra level of quotes.

EXAMPLES

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 "pll &1 -list &2 &3"
P2       do "pll &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.

do

The command line

```
CREATE games
```

expands to

```
cd games;sis games re *.Demo rew Jay.*
```

This shows an easy method of automatically setting initial access on the segments that are 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
```

which is expanded by abbrev to

```
fo Jay.list;do ".l";ws >udd>Demo>Jay "la;ls -dtem -a";ro;
   dp -dl Jay.list
```

See how you can use do at several levels and how it allows abbreviations within abbreviations.

The command line

```
P alpha
```

generates the expansion

```
pll alpha -list
```

whereas the command line

```
P alpha -table
```

expands to

```
pll alpha -list -table
```

Note how references to arguments not supplied 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: do_subtree

SYNTAX AS A COMMAND

Master process or single-process invocations:

```
do_subtree path -control_args
```

Slave process invocations:

```
do_subtree -slave
```

FUNCTION

operates on a given directory, called the starting node, and all directories inferior to it by executing one or two given command lines after substituting the pathname of that directory in the command line. The do command performs the substitution, the directory pathname being taken as the first executed at each node before inferior nodes are operated on (the top-down command line) and after inferior nodes are operated on (the bottom-up command line).

This command enables you to execute the argument command lines in several processes. The walking of the hierarchy can be substantially speeded up by use of this facility. The process in which the initial command lines in starting node is given is named the master process; the other cooperating processes are called the slave processes. The cooperating processes communicate via a segment called dos_mp_seg, which is found (or created if not found) in the working directory when do_subtree is issued. The master process must be logged in and begin executing first when multiple processes are used.

ARGUMENTS

path
is the starting node; -working_dir (-wd) specifies the working directory of the master process if multiple processes are being used.

CONTROL ARGUMENTS

-bottom_up STR, -bu STR
specifies the bottom-up command line. If STR contains blanks, it must be enclosed in quotes. The name of the directory of execution is the first argument to the do command. Access this value with the string "&r1" rather than "&1" in case any directory names contain special characters. Give one of -bottom_up or -top_down, but you can use both.

-first N, -ft N
makes N the first level of the directory hierarchy at which the command lines are executed. By definition, the starting node is at level 1. The default is -first 1.

-last, -lt N
makes N the last level in the storage system hierarchy at which the command lines are executed. The default is 99999, i.e., all levels.

-long, -lg
prints the names of the directories at which the command lines are executed. Unlike walk_subtree, this printing is off by default. In multiprocess executions with a bottom-up command line, an asterisk precedes all directory names for which the process executing the bottom-up command line is not the process that entered the directory first.

-multiprocess, -mp
makes the invoking process the master process of a multiprocess execution. The dos_mp_seg segment is created in the current working directory and execution begins. As slave processes are started, work is distributed by the master process among the slave processes. Execution ends in all processes simultaneously. The top-down/bottom-up order of execution is guaranteed by all processes: no command line is executed at a given directory until either the top-down command line is executed in all superior directories or the bottom-up command line is executed in all inferior directories.

-no_msf
does not treat multisegment files as directories. Unlike walk_subtree, multisegment files are treated as directories by default. Avoid -no_msf for most storage system maintenance operations.

-slave
executes the command line in another process, which must be in a working directory where an active master process has begun executing a multiprocess invocation of do_subtree. The master process uses all control arguments and command lines of the slave process. Execution in all processes finishes at the same time. Don't use more than 35 slave processes.

-top_down STR, -td STR
specifies the top-down command line. If STR contains blanks, it must be enclosed in quotes. The name of the directory of execution is the first argument to the do command. Access this value with the string "&r1" rather than "&1" in case any directory names contain special characters. Give one of -bottom_up or -top_down, but you can use both.

Entry: do_subtree$recover

This entry point is used to pick up the work load of a process that has died in a multiprocess execution. The process picking up the work load of the dead process must have as its working directory the directory in which the dos_mp_seg segment for the current multiprocess execution exists.

SYNTAX AS A COMMAND

```
do_subtree$recover processnumber
```

ARGUMENTS

processnumber
is the process number of the dead process. The process number of a do_subtree process in a multiprocess execution is typed out as it joins the execution.

Entry: do_subtree$abort

This entry point halts a multiprocess execution of do_subtree. All processes return to command level at once. The process executing this command must have as its working directory the directory in which the dos_mp_seg segment of the current multiprocess execution exists.

SYNTAX AS A COMMAND

```
do_subtree$abort
```

Entry: do_subtree$status

This entry point prints out much debugging and status information about all processes involved in a multiprocess execution of do_subtree, including the process identifiers and command lines. The process executing this command must have as its working directory the directory in which the dos_mp_seg of the current multiprocess execution exists.

Name: dprint, dp

SYNTAX AS A COMMAND

```
dp {-control_args} {paths}
```

FUNCTION

queues specified segments and/or multisegment files for printing on one of the Multics line printers. The output is by default identified by your Person_id. This command does not accept standard object segments.

Use enter_output_request; it has functionally replaced dprint.

ARGUMENTS

paths
: are pathnames of segments and/or multisegment files. The star convention is not allowed.

CONTROL ARGUMENTS

-access_label, -albl
: uses the access class of each pathi specified as a label at the top and bottom of every page (see "Notes" below).

-brief, -bf
: suppresses the message "j requests signalled, k already queued. (request_type queue)." This control argument cannot be overruled later in the command line. (See -request_type and -queue below.)

-bottom_label STR, -blbl STR
uses the specified string as a label at the bottom of every page (see "Notes" below).

-copy N, -cp N
prints N copies (N <= 4) of specified paths. It can be overruled by a subsequent -copy. If pathi is to be deleted after printing, all N copies are printed first. If this control argument is not specified, one copy is made.

-defer_until_process_termination, -dupt
does not process the request until the requesting process terminates. Process termination is caused by the logout command, new_proc, or a fatal process error.

-delete, -dl
deletes (after printing) specified paths.

-destination STR, -ds STR
labels subsequent output with the string STR, which is used to determine where to deliver the output. STR is limited to 24 characters and must be quoted if it contains spaces. If -destination is not specified, the default is your Project_id. This control argument can be overruled by a subsequent -destination.

-forms STR
indicates the type of forms to be used when processing the print file. Standard I/O daemon drivers ignore the forms specification when processing print requests.

-header STR, -he STR
identifies subsequent output by the string STR. STR is limited to 64 characters and must be quoted if it contains spaces. If -header is not selected, the default is your Person_id. This control argument can be overruled by a subsequent -header.

-indent N, -in N
prints specified paths so that the left margin is indented N columns. If not given, no indentation occurs.

-label STR, -lbl STR
uses the supplied string as a label at the top and bottom of every page (see "Notes" below).

-line_length N, -ll N
prints specified paths so that lines longer than N characters are continued on the following line; i.e., no line of output extends past column N. If not chosen, a line length of 136 characters is used.

-no_endpage, -nep
prints indicated paths so that the printer skips to the top of a page only when a form-feed character is encountered in the input path. This control argument ignores -page_length (if present).

-no_label, -nlbl
    does not place any labels on the printed output.

-non_edited, -ned
    prints nonprintable control characters as octal escapes rather than suppressing their printing.

-notify, -nt
    sends a confirming message when the requested output is done, showing the pathname and charge.

-page_length N, -pl N
    prints no more than N lines per page, where N is the logical page length (i.e., the number of lines of user data to appear). (Default: varies depending upon the request type)

-queue N, -q N
    prints supplied paths in priority queue N. This control argument can be overruled by a subsequent -queue; if not specified, the default queue for the request type is assumed. (See "Notes" below.)

-request_type STR, -rqt STR
    places specified paths in the queue for requests of the type identified by STR (see "Notes" below). If not specified, the default is "printer."

-single, -sg
    prints specified paths so that any formfeed or vertical-tab character in any of the paths is printed as a single newline character.

-top_label STR, -tlbl STR
    uses the specified string as a label at the top of every page (see "Notes" below).

-truncate, -tc
    prints specified paths so that any line exceeding the line length is truncated rather than "folded" onto subsequent lines.

ACCESS REQUIRED

You requires r access to the segment or multisegment file.

The process that performs the printing (as obtained by print_request_types) must have at least r access to the file and at least s access to the containing directory to verify that you also have at least r access to the file.

If -delete is specified, the I/O coordinator (normally IO.SysDaemon.z) must have at least m access to the containing directory and at least s access to the parent directory of the containing directory to verify that you also have at least m access to the containing directory.

NOTES

If you invoke dprint without any arguments, the system prints a message giving the status of the default printer queue.

If control arguments are present, they affect only paths specified after their appearance in the command line. If control arguments are specified without a following pathi argument, they are ignored for this invocation of the command and a warning message is printed.

The -queue 1 control argument places requests in the top priority queue, -queue 2 in the second priority queue, and -queue 3 in the third. There can be 1 to 4 priority queues for a specified request type as determined by the site. Higher priority queues usually have a higher cost associated with them.

The -brief, -delete, -single, -truncate, and -no_endpage control arguments cannot be reset in a specified invocation of the command; e.g., once -delete appears in a line, all subsequently specified paths are deleted after printing.

The -request_type control argument is used to ensure that a request is performed by a member of a particular group of printers, e.g., to distinguish between onsite printers and remote printers at various locations, or between printers being charged to different projects. Only request types of generic type "printer" can be specified; use the print_request_types command to list them. It is a system restriction that you cannot dprint from a private logical volume.

If a requested output operation cannot be done, the daemon process sends you a message of the form:

Request path reason.

The -label, -top_label, -bottom_label, and -access_label control arguments allow you to place labels on each page of printed output. The default labels are access labels; i.e., -access_label is assumed. These control arguments are read, in sequence, from left to right by dprint; for example, if -access_label is specified, it is printed at the top and bottom of the page. If you next give -top_label STR, then the top access label becomes STR, but the bottom label remains the same. Each label control argument can override the preceding one. The label lines are printed on the second and the next-to-last lines of the page. If the access class of pathi is system_low and the access class name defined for system_low is null, then the default access label is blank. The default access label can be overridden by -no_label, if labels are not wanted, or by any other label control argument.

The top and bottom labels are treated independently; thus, -top_label leaves an access label as the default bottom label. A page label that exceeds 136 characters is truncated to that length. Only the first line of a page label is printed, i.e., a new line terminates the page label. Formfeeds and vertical tabs are not permitted. The label control arguments are incompatible with -no_endpage, and they are ignored regardless of the -no_endpage's position in the command line.

Segments and multisegment files cannot be printed unless appropriate system processes have sufficient access. The process (normally IO.SysDaemon) that runs devices of the specified class must have r access to all paths to be printed and s permission on the containing directory. A file cannot be deleted after printing unless its safety switch is off and the system process has at least sm access on the containing directory. A file is not deleted if it has a date-time-contents-modified value later than the date-time-contents-modified value at the time of the dprint request.

This command does not accept the star convention: if it finds a name containing asterisks, it prints a warning message and continues processing the other arguments.

If pathi specifies a standard Multics storage system object segment, dprint prints a warning message and continues processing its other arguments.

EXAMPLES

The command line

```
! dp -he Jones -ds BIN-5 -cp 2 -dl test1 test7 -he Doe text.runout
```

dprints and then deletes two copies of the segments named test1 and test7 in the current working directory (with the header "Jones" and the destination "BIN-5") and two copies of the segment named text.runout in the current working directory with the header "Doe" and destination "BIN-5".

Name: dpunch, dpn

SYNTAX AS A COMMAND

```
dpn {-control_args} {paths}
```

FUNCTION

queues specified segments and/or multisegment files for punching by the Multics card punch. It is similar to dprint.

Use enter_output_request; it has functionally replaced dpunch.

ARGUMENTS

paths
: are pathnames of segments and/or multisegment files. The star convention is not allowed.

CONTROL ARGUMENTS

-7punch, -7p
punches the specified paths using 7-punch conversion. It can be overruled by either -mcc or -raw.

-brief, -bf
suppresses the message "j requests signalled, k already queued. (request_type queue)." This control argument cannot be overruled later in the command line. (See -request_type and -queue below.)

-copy N, -cp N
punches N copies (N <= 4) of specified paths. It can be overruled by a subsequent -copy. If pathi is to be deleted after punching, all N copies are punched first. If this control argument is not specified, one copy is made.

-defer_until_process_termination, -dupt
does not process the request until the requesting process terminates. Process termination is caused by the logout command, new_proc, or a fatal process error.

-delete, -dl
deletes (after punching) all specified paths.

-destination STR, -ds STR
uses the string STR to determine where to deliver the deck. If not specified, the default is your Project_id. This control argument can be overruled by a subsequent -destination.

-header STR, -he STR
identifies subsequent output by the string STR. If not specified, the default is your Person_id. This control argument can be overruled by a subsequent -header.

-mcc
punches the specified paths using character conversion. It can be overruled by either -raw or -7punch. (Default)

-notify, -nt
sends a confirming message when the requested output is done, showing the pathname and charge.

-queue N, -q N
punches specified paths in priority queue N (N <= 4). It can be overruled by a subsequent -queue. If not specified, the default queue for the request type is assumed. (See "Notes" below.)

-raw
punches the specified paths using no conversion. It can be overruled by either -mcc or -7punch.

-request_type STR, -rqt STR
places specified paths in the queue for requests of the type identified by the string STR (see "Notes" below). If not specified, the default request type is "punch."

ACCESS REQUIRED

You require r access to the segment or multisegment file.

The process that performs the punching (as obtained by print_request_types) must have at least r access to the file and at least s access to the containing directory to verify that you also have at least r access to the file.

If -delete is specified, the I/O coordinator (normally IO.SysDaemon.z) must have at least m access to the containing directory and at least s access to the parent directory of the containing directory to verify that you also have at least m access to the containing directory.

NOTES

If you invoke dpunch without any arguments, the system prints a message giving the status of the default punch queue.

If control arguments are present, they affect only paths specified after their appearance on the command line. If control arguments are specified without a following pathi argument, they are ignored for this invocation of the command and a warning message is printed.

The -delete control argument is the only one affecting segments that cannot be reset in a specified invocation of the command. Once -delete appears in a line, all subsequent segments are deleted after punching.

The -request_type control argument is used to ensure that a request is performed by a member of a particular group of punches; for example, to distinguish between onsite punches and remote punches at various locations, or between punches being charged to different projects. Specify only request types of generic type "punch." List request types by using print_request_types. If a requested output operation cannot be done, the daemon process sends you a message of the form:

"Unable to punch" path reason.

A segment or multisegment file cannot be punched unless appropriate system processes have sufficient access. The process (normally IO.SysDaemon) that runs devices of the specified class must have r access to all files to be punched and s permission on the containing directory. A file cannot be deleted after punching unless its safety switch is off and the system process has at least sm permission on the containing directory. A file is not deleted if it has a date-time-contents-modified value later than the date-time-contents-modified value at the time the request is completed.

This command does not accept the star convention: if it finds a name containing asterisks, it prints a warning message and continues processing the other arguments.

Before deleting a file that has been punched, read the deck back in and compare it (using the compare command) with the original, to ensure the absence of errors.

EXAMPLES

The command line

```
! dpunch a b -7punch -he RBurns c.pl1 -dl -he "FSchubert" alpha
```

punches segments a and b in the current working directory using mcc conversion, punches segment c.pl1 using 7-punch conversion with "for RBurns" added to the heading, and punches and deletes segment alpha using 7-punch conversion with "FSchubert" added to the heading.

Name: dump_segment, ds

SYNTAX AS A COMMAND

```
ds path {offset} {length} {-control_args}
ds segno {offset} {length} {-control_args}
ds virtual_pointer {offset} {length} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ds path {offset} {length} {-control_args}]
[ds segno {offset} {length} {-control_args}]
[ds virtual_pointer {offset} {length} {-control_args}]
```

FUNCTION

prints, in octal or hexadecimal format, selected portions of a segment. It prints out either four or eight words per line and optionally prints out an edited version of the ASCII, BCD, EBCDIC (in 8 or 9 bits), or 4-bit byte representation.

ARGUMENTS

path
: is the pathname or (octal) segment number of the segment to be dumped. If path is a pathname but looks like a number, make the preceding argument be -name. You can use the star convention for the command only.

offset
is the (octal) offset of the first word to be dumped. If you omit both offset and length, the entire segment is dumped. If you specify offset, add it to the offset given in the virtual pointer (i.e., 234|40 10 5 implies seg 234|50 for five words). If you need no offset but need length, supply offset as zero.

length
is the (octal) number of words to be dumped. If you supply offset and omit length, one word is dumped.

segno
is the octal segment number of the segment to be dumped. It cannot be a hardcore segment number.

virtual_pointer
is an ASCII representation of a pointer (see Section 1).

CONTROL ARGUMENTS

-4bit
prints out, or returns, a translation of the octal or hexadecimal dump based on the Multics unstructured 4-bit byte. The translation ignores the first bit of each 9-bit byte and uses each of the two groups of four bits remaining to generate a digit or a sign.

-address, -addr
prints the address (relative to the base of the segment) with the data. (Default)

-bcd
prints the BCD representation of the words in addition to the octal or hexadecimal dump. There are no nonprintable BCD characters, so periods can be taken literally. It causes the active function to return BCD.

-block N, -bk N
dumps words in blocks of N words separated by a blank line. The offset, if being printed, is reset to the initial value at the beginning of each block.

-character, -ch, -ascii
prints the ASCII representation of the words in addition to the octal or hexadecimal dump. Characters that cannot be printed are represented by periods. It causes the active function to return ASCII. (Default)

-ebcdic8
prints the EBCDIC representation of each eight bits in addition to the octal or hexadecimal dump. It causes the active function to return 8-bit EBCDIC. Characters that cannot be printed are represented by periods. If an odd number of words is requested to dump, the last four bits of the last word do not appear in the translation.

-ebcdic9
prints the EBCDIC representation of each 9-bit byte in addition to the octal or hexadecimal dump. Characters that cannot be printed are represented by periods. It causes the active function to return 9-bit EBCDIC.

This page intentionally left blank.

-entry_point NAME, -ep NAME
specifies that the offset of the first word to be dumped is relative to the location defined by the externally available symbol NAME. Use -entry_point only for object segments (created by a compiler or by the create_data_segment command).

-header, -he
prints a header line containing the pathname (or segment number) of the segment being dumped as well as the date-time printed. (Default: to print a header only if the entire segment is being dumped, i.e., if you give neither the offset nor the length argument)

-hex8
prints the dumped words in hexadecimal with nine hexadecimal digits per word rather than octal with 12 octal digits per word.

-hex9
prints the dumped words in hexadecimal with eight hexadecimal digits per word rather than 12 octal digits per word. Each pair of hexadecimal digits corresponds to the low-order eight bits of each 9-bit byte.

-interpreted, -it
prints the data decoded into the indicated format.

-long, -lg
prints eight words on a line. You can't use -long with -4bit, -bcd, -character, -ebcdic8, -ebcdic9, or -short. (Default: four)

-name PATH, -nm PATH
indicates that PATH is a pathname even though it may look like an octal segment number.

-no_address, -naddr
does not print the address.

-no_header, -nhe
does not print the header line even though the entire segment is being dumped.

-no_interpret, -nit
suppresses printing of the decoded data. (Defaul)

-no_raw, -nraw
suppresses printing of the raw data.

-no_offset, -nofs
does not print the offset. (Default)

-no_suppress_duplicates, -nsd
indicates that sequential lines are to be printed even if they would be identical to previous lines.

-octal, -oc
display the raw data in octal format, with 12 octal digits per word. (Default, for raw data)

-offset {N}, -ofs {N}
prints the offset (relative to N words before the start of the data block being dumped) along with the data. If you supply no N, 0 is assumed.

-raw
indicates that the raw data is to be printed. (Default)

-short, -sh
compacts lines to fit on a terminal with a short line length. Single spaces are placed between fields, and only the two low-order digits of the address are printed except when the high-order digits change. This shortens output lines to less than 80 characters.

-suppress_duplicates, -sd
indicates that if lines to be printed are identical to the previous line with a single block, they are to be replaced by a short line of equal signs. (Default)

NOTES

The defaults for use as a command are -address, -no_interpret, -no_offset, -raw, and -supress_duplicates with -header if the entire segment is printed, and -no_header if only parts of the segment are to be printed. The defaults for use as an active function are -no_address, -no_header, -no_interpret, -no_offset, -no_suppress_duplicates, and -raw.

Supply only one of -4bit, -bcd, -character, -ebcdic8, or -ebcdic9.

If you invoke -4bit, -bcd, -character, -ebcdic8, -ebcdic9, -hex8, or -hex9, the information is returned in the specified format only. All other arguments are ignored in active function invocation.

In the active function the following control arguments are invalid: -address, -block, -header, -offset, and -suppress_duplicates.

When you give conflicting control arguments, the last one on the command line is used.

The active function returns either raw data in octal or hexadecimal representation or the interpreted data representation.

Name: edm

SYNTAX AS A COMMAND

```
edm {path}
```

FUNCTION

invokes a simple Multics context editor. It is used for creating and editing ASCII segments. You can't called it recursively.

ARGUMENTS

path
: specifies the pathname of a segment to be created or edited. If you give no path, edm begins in input mode (see "Notes" below), ready to accept whatever is subsequently typed as input. If you supply path but the segment does not yet exist, edm also begins in input mode; however if the segment already exists, edm begins in edit mode.

LIST OF EDITOR REQUESTS

```
-         backup
=         print current line number
,         comment mode
.         mode change
b         bottom
c         change
d         delete
E         execute
f         find
i         insert
k         kill
l         locate
merge     insert segment
move      move lines within segment
n         next
p         print
q         quit
qf        quitforce
r         retype
s         substitute
t         top
updelete  delete to pointer
upwrite   write to pointer (upper portion of segment)
v         verbose
w         write
```

NOTES

This command operates in response to requests from you. To issue a request, make edm be in edit mode. This mode is entered in two ways: if the segment already exists, it is entered automatically when you edm invoke; if dealing with a new segment (and edm has been in input mode), you must issue the mode change character. This character is the period, issued as the only character on a line. The command announces its mode by typing "Edit." or "Input." when the mode is entered. From edit mode, input mode is also entered via the mode change character.

The edm requests are predicated on the assumption that the segment consists of a series of lines to which there is a conceptual pointer that indicates the current line. (The "top" and "bottom" lines of the segment are also meaningful.) Various requests explicitly or implicitly move the pointer; others manipulate the line currently pointed to. Most requests are indicated by a single character, generally the first letter of the name of the request. For these requests only the single character (and not the full request name) is accepted by the command. Certain requests have been considered sufficiently dangerous, or likely to confuse you, that you must specify their names in full. If you issue a quit signal while in edit mode and then invoke the program_interrupt command, the effect of the last request executed on the edited copy is nullified; in addition, any requests not yet executed are lost. If you type program_interrupt after a quit in comment or input modes, all input since last leaving edit mode is lost; if you wish to keep the input, invoke the start command following the quit.

See the *FAST Subsystem User's Guide* (AU25) for an introduction to the use of edm.

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.

ARGUMENTS

paths
: are pathnames 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 abbreviations. 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.

-force, -fc
permits the use of terminal type control arguments (-ttp, -query, -reset) when in the video system; however, the -ttp argument is restricted to video controllers.

-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 line speed to obtain proper padding (for ARPANet users), where N is the output line baud rate in bits/second. This control argument is ignored in the video system.

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

-no_force, -nfc
prevents the use of terminal type control arguments when in the video system. (Default)

-no_start_up, -no_startup, -ns
prevents use of your startup (start_up.emacs).

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

-query
queries you for a terminal type without checking the Multics terminal type first. The query response can be any recognized editor terminal type. (See "Notes.")

-reset
specifies that Emacs disregard the terminal type set by -terminal_type and set it in accord with the Multics terminal type instead (see "Notes").

-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. If STR is not a recognized type, Emacs queries you after entry, providing a list of recognized types. (See "Notes.")

NOTES

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

The control arguments -query, -reset, and -terminal_type are incompatible. You can't use them in the video system unless you provide -force.

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 Text Editor* (CP31). You can find a tutorial introduction to the Emacs Text Editor, fully describing the editing requests available and containing instructions for using special features of emacs, in the *Emacs Text Editor User's Guide* (CH27). A guide for programmers writing extensions and terminal control modules (CTLs) in Lisp is provided in the *Emacs Extension Writer's Guide* (CJ52).

You can get a complete list of available requests in emacs via the make-wall-chart request while in emacs. Type the following:

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

where ESC is the escape key on the terminal.

In addition, emacs provides its own online, interactive tutorial, which you can invoke by typing the following:

```
! emacs
! ^_?
```

where "^" stands for the CONTROL key, which you must hold down while pressing the underscore character.

See also the list_emacs_ctls command.

Name: encode

SYNTAX AS A COMMAND

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

FUNCTION

enciphers a segment's contents according to a key that is not stored in the system. The enciphered segment has the same length as the original segment. The encode command provides additional security for data stored in a Multics segment.

ARGUMENTS

path1
: is the pathname of a segment to be enciphered. The star convention is allowed.

path2
: is the pathname of an enciphered segment to be produced. If the last path2 is not specified, it is assumed to be the same as path1. The equal convention is allowed. This command always appends the code suffix to path2 to produce the name of the enciphered segment.

CONTROL ARGUMENTS

-key STR
: specifies the encipherment key STR on the command line and does not query for one. This control argument is useful in exec_com's for multiple invocations of the command with the same key.

NOTES

The encode command requests an encipherment key (1-11 characters not including space, semicolon, or tab) from the terminal. Printing on the terminal is suppressed (the printer is turned off) while the key is typed. The command then requests that the key be typed again, to guard against the possibility of mistyping the key. If the two keys do not match, the key is requested twice again.

All segments specified in an invocation of encode are enciphered with the same key.

To reconstruct the original segment from the enciphered segment, see the decode command.

Name: encode_access_class, eac

SYNTAX AS A COMMAND

```
eac {STR} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[eac {STR} {-control_arg}]
```

FUNCTION

prints/returns an encoded version (a short character string suitable for inclusion in entrynames) of an access class; this is an interface to the subroutine convert_access_class_$encode.

ARGUMENTS

STR
: is the access class to be encoded. If you use STR, don't use the control argument. If you give no access class, the process's current authorization is used.

CONTROL ARGUMENTS

-access_class ACC, -acc ACC
: is an alternate way to specify the access class, where ACC is the access class to be encoded.

Name: enter_abs_request, ear

SYNTAX AS A COMMAND

```
ear path {-control_args}
```

FUNCTION

allows you to request the creation of an absentee process, which you can delay until a specified time. An absentee process executes commands from a segment and places the output in another segment.

ARGUMENTS

path
: specifies the pathname of the absentee control segment associated with this request. The absin suffix is assumed.

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 are taken as arguments to the absentee control segment; therefore put -ag last in the command line.

-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, enclose it in quotes. The comment is printed whenever you or the operator lists the request. It indicates 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.

-extend
appends absentee process output to the absout file. It overrides -tc. (Default)

-foreground, -fg
places the request in the foreground queue, rather than in one of the numbered background queues. For load control and charging, jobs in the foreground queue are treated as interactive logins; that is, a foreground job is logged in as if you would have logged in interactively, and, while logged in, it occupies a primary slot in your load control group. (See -secondary.)

-limit N, -li N
places a limit on the CPU time used by the absentee process. N must be a positive decimal integer specifying the limit in seconds. Your site defines the default limit for each queue and the upper limit 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 you (by an interactive message sent to your mailbox) when the job is logged in, when it is logged out, or when it is deferred for any reason other than your request. The latter may 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 is logged in to run the job. The system administrator controls the use of -proxy by an access control segment.

-queue N, -q N
submits the request to queue N, where N is an integer specifying the number of the absentee process queue. Your site administrator designates the default queue. There are four background queues, with queue 1 having the highest priority. Your site determines the highest numbered queue processed on each shift. For convenience in writing exec_coms and abbreviations, the word "foreground" (fg) following -q is equivalent to -fg.

-resource STR, -rsc STR
specifies resources given in STR (e.g., one or more tape drives); don't start them until they are available. These resources are also reserved for the absentee job before it is logged in. You need not do resource reservation (using the reserve_resource command) in the absin segment. Enclose the resource description in quotes if it contains blanks or other command language characters.

-restart, -rt
starts over the computation of this request from the beginning if interrupted (for example, by a system crash). (Default: not to restart the computation)

-secondary
logs in a foreground job as a secondary user (subject to preemption) if there are no primary slots available in your load control group. By default a foreground job is only logged in if a primary process can be created for you.

-sender STR
enters requests only from sender STR. In most cases, the sender is an RJE station identifier.

-time DT, -tm DT
delays the creation of the absentee process until a specified date-time, where DT must be a character string acceptable to convert_date_to_binary_ (see the Subroutines manual). If DT contains blanks, enclose it in quotes.

-truncate, -tc
truncates the absout file, so that the absentee process begins writing at the beginning of the absout file.

NOTES

The main 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 you don't supply the pathname of the output segment, the output of the absentee process is directed to a segment whose pathname is the same as the absentee control segment, having the absout suffix instead of absin. If you omit the absout suffix 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 w 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.

Specifying -tm is as if you issued ear at the deferred time. 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 -tm. If an absentee job cannot be run or if it terminates abnormally, the system sends an interactive user message to your mailbox, whether or not you give -nt.

All input and output that occurs in the absentee job is written to the segment STR.absout in the same directory as the absentee segment STR.absin. This absout segment has its safety switch turned on temporarily while the job is running, since deleting the absout segment crashes the absentee job.

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 or certain types of process damage occur, the messages may not appear in the absout segment. If you are diverting output to another file (by file_output, for example) when such an error occurs, you may need to issue adjust_bit_count for that file; this is because revert_output was never executed and the bit count of the file being written was not updated.

To make sure that the absout is printed after absentee logout, even if it does not reach completion, put the following command line near the beginning of the absin file:

```
eor -dupt [user absout]
```

where "-dupt" is short for -defer_until_process_termination.

To delete the absout when done, make the following the last line in the absin file:

```
dl [user absout] -force; logout -brief
```

The logout command prevents an abnormal termination trying to write another line to the deleted absout file.

An alternative to deleting the absout is to rename it so as to keep only the latest copy:

```
answer yes -brief rename [user absout] ===.old
```

This command line, which can appear anywhere in the absin file, forces deletion of any previous old copy and saves the current absout with suffix old for later examination.

To delete the absin when completed, make the following the last line in the absin file:

```
dl [user absin]; logout
```

The logout command prevents an abnormal termination trying to read another line from the deleted absin file.

The -tc control argument truncates the absout file at the time the absentee job is starting to run, but if the job is being restarted after a system crash, the truncation is not performed.

See also list_abs_requests and cancel_abs_request.

EXAMPLES

Suppose you want 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
eor -delete x.list
logout
```

The command line

```
! ear absentee_pl1
```

creates an absentee process that does the following:

1. Sets the working directory to the directory named current_dir, which is inferior to your normal home directory.

enter_abs_request

2. Compiles a PL/I program named x.pl1 with two control arguments.

3. Prints 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.

This page intentionally left blank.

Suppose that an absentee control segment, trans.absin, contains the following:

```
change_wdir &1
&2 &3 -map &4
eor -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 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.

-non_edited, -ned
prints nonprintable characters as octal escape sequences (e.g.,\000 or \577).

A formfeed character in the input file normally begins printing the subsequent output at the top of the next page. Similarly, a vertical-tab character normally begins printing the subsequent output at the next vertical-tab stop. Vertical-tab stops are set at lines 1, 11, 21, 31, 41, and 51.

-vertical_space, -vertsp
skips to the top of the page when a formfeed character is found in the file being processed. It skips to the next vertical-tab stop when a vertical-tab character is encountered. (Default)

-no_vertical_space, -nvertsp
treats formfeed and vertical-tab characters as newline characters during printing.

Print Page Labels

The following control arguments govern the printing of labels at the top and bottom margins of each page of printed output.

-label {-control_args} STR, -lbl {-control_args} STR
puts STR as a label at the top and bottom of each printed page.

The label can be constructed from STR using the active string evaluation or the equal convention, as described below.

-active_string, -astr
makes STR an active string, which eor evaluates for each request that is submitted. For example,

```
-label -astr date
```

uses today's date as the label.

The pathname of the file being processed can be used in the active string because STR is evaluated as:

```
[do "[STR]" pathname]
```

For example,

```
-label -astr "date;string -;spe &1"
```

produces the label "12/23/84-test" when submitting a request to print test.pl1 on December 23, 1984.

Name: enter_output_request, eor

SYNTAX AS A COMMAND

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

FUNCTION

submits requests to printer, punch, or plotter queues. All control arguments are nonpositional. Paired control arguments override one another if both are used in a single command. You 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 CONTROL ARGUMENTS

-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 (see "Queuing Controls" below).

-queue N, -q N
submits requests to queue N of the request type (see "Queuing Controls" below).

-header {-control_args} STR, -he {-control_args} STR
identifies output with a heading of STR (see "Processing Controls" below).

-destination {-control_args} STR, -ds {-control_args} STR
labels output with STR, which is used to determine where to deliver the output (see "Processing Controls" below).

-copies N, -cp N
produces N copies of the printed, punched, or plotted output (see "Processing Controls" below).

-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)

BASIC OPERATION: When this 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 you give no control arguments, 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.

This command provides standard settings for control arguments not given. However, it also allows you to change these default values for the various request types. By using user-settable defaults, you can tailor eor to the type of print, punch, or plot requests most frequently queued. (See "Setting Defaults" below.)

The eor command offers precise control over how print, punch, or plot requests are queued; how the requests are processed (e.g., formatting of printed pages or conversion of punched files); and what actions are taken after processing (e.g., delete the file, notify the submitter). However, many users find that the following subset of control arguments provide adequate control for printing, punching, and plotting.

CONTROL ARGUMENTS

The control arguments accepted by eor are described below, organized by function.

QUEUING CONTROLS
-print, -pr
-punch, -pch
-plot
-request_type, -rqt
-queue, -q
-name, -nm
-brief, -bf
-long, -lg
-force, -fc
-no_force, -nfc

enter_output_request

PROCESSING CONTROLS
-header, -he
-destination, -ds
-copies, -cp
-forms

POSTPROCESSING CONTROLS
-delete, -dl
-no_delete, -ndl
-notify, -nt
-no_notify, -nnt

PREPROCESSING CONTROLS
-defer_until_process_termination, -dupt
-no_defer_until_process_termination, -ndupt

PRINT FORMAT
-page_length, -pl
-line_length, -ll
-indent, -ind
-truncate, -tc
-fold
-no_end_page, -nep
-end_page, -ep

PRINT OUTPUT CONVERSION
-non_edited, -ned
-edited, -ed
-no_vertical_space, -nvertsp
-vertical_space, -vertsp

PRINT PAGE LABELS
-label, -lbl
-top_label, -tlbl
-bottom_label, -blbl
-access_label, -albl
-no_label, -nlbl

PUNCH OUTPUT CONVERSION
-mcc_punch, -mcc
-raw_punch, -raw
-7punch, -7p

SETTING DEFAULTS
-list_defaults, -ldft
-print_defaults, -pdft
-all, -a
-replace_defaults, -rdft
-set_defaults, -sdft
-delete_defaults, -ddft
-default_name, -dnm
-set_default_request_type, -sdrqt

Queuing Controls

The following control arguments govern how print, punch, and plot requests are submitted.

A request is submitted to a particular request type to control where and how it is printed, punched, or plotted. For example, a file to be printed on a remote printer must be submitted to the request type associated with that printer. Similarly, a file to be printed on a special print form must be submitted to a special request type associated with that form. Use the print_request_types (prt) command to list available request types.

Some request types are associated with printing, some are for card punching, and others are for plotting of output on installation-defined plotting devices. The request type can be chosen in several ways.

-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 print_request_types. (Default: printer when printing, punch when punching, plotter when plotting)

-punch, -pch
submits requests to the default punch request type.

-plot
submits requests to the default plotter request type.

-print, -pr
submits requests to the default print request type. (Default)

To get the default print, punch, and plotter request types, use "eor -ldft" or print_request_types.

You can submit requests using priority queue 1, 2, 3, or 4, queue 1 having the highest priority. Requests are processed by the IO Driver process by priority: all requests from queue 1 are processed before any ones from queue 2, and so on; within a given queue, requests are processed in the order in which they were submitted. The higher the queue number, the quicker the request is processed and the higher the billing rate. Your site associates billing rates with each priority queue.

-queue N, -q N
submits the request to queue N of the request type. If N is "default" (dft) or "-default" (-dft), the default priority queue is used. Some request types have fewer than four queues, so the default priority queue varies, depending upon the request type.

You can submit a request to process a file whose name looks like a control argument or starname.

-name path, -nm path
submits a request for the single file identified by the pathname.

To print, punch, or plot a file the IO Driver process must have at least r access to the file and s access to the directory that contains the file and the file must have a nonzero bit count. The print_request_types command lists the access name associated with the IO Driver for each request type.

-force, -fc
forces sufficient access to a file to allow printing, punching, or plotting and adjusts the bit count of segments having a zero bit count.

-no_force, -nfc
prints an error message for files having a zero bit count and for files to which the IO Driver has insufficient access. (Default)

The eor command reports how many files are submitted during an invocation with a message of the form:

J requests submitted; K already in REQUEST_TYPE queue N.

-long, -lg
prints the above message. (Default)

-brief, -bf
suppresses the above message.

Processing Controls

The following control arguments govern how the IO Driver processes the output request.

The IO Driver places leading and trailing "banner pages" around a print file to identify it. Similarly the driver for central-site card punch devices places specially punched "flip cards" around a punch file. For remote card punches, special ID cards are punched at the start of each output file. For plot requests the IO driver creates a plotted banner page to identify each plot request. The banner pages, flip cards, and ID cards contain a destination, which tells the operator how to route the output back to the submitter, and a heading, which further identifies the file. For printed files the first 13 characters of the destination and header are printed in large, block letters to help identify the file.

-header {-control_args} STR, -he {-control_args} STR
identifies output with a heading of STR. STR is limited to 59 characters; "quote" it if it contains spaces. If "-default" (-dft) is given, the heading string is reset to its default value. (Default: submitter's Person_id)

You can construct the heading string from STR using the active string evaluation or the equal convention, as described below.

-active_string, -astr
makes STR an active string, which eor evaluates for each request that is submitted; for example,

```
-he -astr date
```

uses today's date as the heading.

The pathname of the file being processed can be used in the active string because STR is evaluated as:

```
[do "[STR]" pathname]
```

For example,

```
-he -astr "date;string -;spe &1"
```

produces the heading "12/23/84-test" when submitting a request to print test.pl1 on December 23, 1984.

-equal_name, -enm
the heading is constructed by applying the equal convention to STR and the entryname of the file being processed. If the STR contains any equal signs (=) or percent characters (%), then -equal_name is assumed by default unless you supply -string. You can give this operand with -active_string to apply the equal convention to the evaluated active string.

-string, -str
treats STR as an ordinary heading, even though it may contain equal signs or begin with a hyphen.

-destination {-control_args} STR, -ds {-control_args} STR
labels output with STR, which is used to determine where to deliver the output. STR is limited to 24 characters; quote it if it contains spaces. If "-default" (-dft)is used, the destination string is reset to its default value.
(Default: submitter's Project_id)

You can construct the heading string from STR using the active string evaluation or the equal convention, as described by the control arguments listed with -header above. *.cbn

-copies N, -cp N
produces N copies of the printed, punched, or plotted output. N can be any number from 1 to 30, or it can be "default" (dft) or "-default" (-dft) to obtain the default number of copies. (Default: 1)

-no_separator, -nsep
specifies that when multiple copies of a request are processed the inner head and tail sheets should not be included.

-separator, -sep
specifies that when multiple copies of a request are processed the inner head and tail sheets should be included. (Default)

You can specify the type of software-generated forms to be used when printing a file (e.g., when printing on microfiche equipment); such forms differ from printer paper stock, which you should select by giving a -request_type identifying a set of queues associated with the desired paper stock.

-forms STR
specifies the type of forms to be used when printing a file. Currently standard I/O daemon drivers ignore -forms when processing a print request.

Preprocessing Controls

The following control arguments govern when a request is to be processed.

-defer_until_process_termination, -dupt
does not process the request until the requesting process terminates. Process termination is caused by the logout command, new_proc, or a fatal process error.

-no_defer_until_process_termination, -ndupt
processes the request normally. (Default)

Postprocessing Controls

The following control arguments govern what the IO Coordinator does with a file after it is printed or punched.

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

-no_delete, -ndl
does not delete files. (Default)

If -delete is specified, the IO Coordinator (usually IO.SysDaemon.z) must have at least m access to the directory containing the entry. A file is not deleted if it is modified after the request to output it is submitted.

When the file is processed, the IO Driver can send a confirmation message of the form:

```
printed PATHNAME $COST queue N DEVICE_NAME PROCESSING_ID
```

where PATHNAME is the pathname of the file that was output, COST is the dollar charge for processing the file, DEVICE_NAME is the name of the physical device on which the file was printed (e.g., prta or MDC_Office.prt), and PROCESSING_ID is a number by which the file was identified to the operator.

-no_notify, -nnt
does not notify the submitter. (Default)

-notify, -nt
notifies the submitted after the entry is printed, punched, or plotted.

Print Format

The following control arguments govern the format of printed pages.

You can control the length of lines in the printed output, the page length, the indentation of the left margin, and the method used in processing lines longer than the line length or pages longer than the page length. For example, when printing on special forms, it is often necessary to specify an indentation for the left margin to position the output properly on the form.

Most printers can print at least 132 columns per line, some as high as 136, and up to 66 lines per page. However the maximum line and page length allowed for a particular request type depends upon both the width and the length of print forms mounted on the printer and the line and page length constraints of the printer device.

-line_length L, -ll L
prints L columns per line only. L can be any number from 1 to 250, or it can be "default" (dft) to obtain the default line length. (Default: varies depending upon the request type)

-indent I, -ind I
indents the left margin by I columns. I can be any number from 0 to L (the line length) or it can be "default" (dft) to obtain the default indentation. (Default: 0)

-truncate, -tc
truncates lines longer than L-I columns.

-fold
continues lines longer than L-I columns on subsequent print lines. (Default)

-page_length P, -pl P
prints no more than P lines per page, where P is the logical page length (i.e., the number of lines of user data to appear). P can be "default" (dft) to obtain the default page length. (Default: varies depending upon the request type)

-end_page, -ep
skips to the top of the next page after P lines are printed on a page. (Default)

-no_end_page, -nep
skips to the top of the page only when a formfeed (newpage) character is encountered in the input. Use of -no_end_page disables -page_length.

Print Output Conversion

The following control arguments govern how nonprinting ASCII (or non-ASCII) characters are printed in the listing.

-edited, -ed
suppresses printing of nonprintable characters. (Default)

-equal_name, -enm
the label is constructed by applying the equal convention to STR and the entryname of the file being processed. If the STR contains any equal signs (=) or percent characters (%), then -equal_name is assumed by default (unless -string is given). This operand can be given with -active_string to apply the equal convention to the evaluated active string.

-string, -str
treats STR as an ordinary label, even though it may contain equal signs or begin with a hyphen.

-center
centers the label on the printer page, based upon the line length given by -line_length or the default line length associated with the request type. (Default: label is left justified)

-top_label {-control_args} STR, -tlbl {-control_args} STR
puts STR as a label at the top of each printed page.

The top label can be constructed from STR using the active string evaluation or the equal convention, as described by the control arguments listed with -label above.

-bottom_label {-control_args} STR, -blbl {-control_args} STR
puts STR as a label at the bottom of each printed page.

The bottom label can be constructed as the top label.

-access_label, -albl
puts the access class of the entry being printed at the top and bottom of every page; for entries at system_low access class, this is equivalent to -no_label. (Default)

-no_label, -nlbl
does not place any labels in the printed output.

Access labels are centered in the top and bottom margins on each printed page. All other labels are aligned with the left margin.

For a file at system_low access class, the access label is a null string. Thus, -access_label for such files is equivalent to -no_label.

The top and bottom labels on a page are treated independently. Giving -top_label alone leaves an access label at the bottom of the page; giving -bottom_label alone leaves an access label at the top of each page.

When -no_end_page is specified, the top and bottom margins of each page are eliminated; therefore, -no_end_page is incompatible with the label control arguments.

Punch Output Conversion

The following control arguments govern the type of output conversion performed when punching a file. Three conversion modes are available: character conversion using Multics card codes, binary conversion using Multics 7-punch card codes, or raw (no) conversion. (See the Programmer's Reference Manual for information on the input/output system and conversion codes.)

-mcc_punch, -mcc
punches files using character conversion. (Default)

-raw_punch, -raw
punches files using no conversion.

-7punch, -7p
punches files using 7-punch conversion.

When any of the above control arguments are used, -punch is assumed.

Setting Defaults

The following control arguments allow you to print or change the default control argument values used by eor (see "Examples.")

A different group of control argument values can be defined for each print, punch, or plot request type. This allows different defaults for local and remote printers and for specialized print forms.

Groups of control argument settings are stored for each user in the default value segment (usually a home directory segment called Person_id.value). If it does not already exist, this segment is created automatically the first time you invoke eor. The following control arguments list the values for one or more of the defined groups.

-list_defaults, -ldft
lists the names of all groups of control argument values that have been defined. Printing, punching, and plotting groups are listed separately. The list also identifies the default group used for printing, punching, or plotting (the group used when -request_type is not given).

-print_defaults, -pdft
prints the control argument values associated with the group identified by -request_type. If -request_type is omitted, prints values for the default punch group (if punch-oriented control arguments are given), default plot group (if -plot is given), or the default print group.

-all, -a
prints the control argument values associated with all defined groups (-print_defaults is assumed).

A new group of control argument settings can be defined by referencing the group name in -request_type and using either -replace_defaults or -set_defaults. Similarly, an existing group can be modified or deleted.

When defining a group of control argument settings, the group can be initialized to standard settings, and then any control arguments supplied in the command line are used to modify these settings. Alternately, an existing group can be modified by applying the given control arguments to the group without resetting the group to standard values.

-replace_defaults, -rdft
resets control argument settings in the group to their standard values and then applies the specified control arguments to modify the group.

-set_defaults, -sdft
adds control arguments given in the command line to the existing default values.

-delete_defaults, -ddft
deletes the definition of the named group of control argument settings.

When eor is invoked without -request_type, a default request type is used. The standard default request type for print requests is "printer", which usually designates the site's local printer; for punch requests is "punch", which usually designates the site's local punch; and for plot requests is "plotter". You can change these default request types.

-set_default_request_type STR, -sdrqt STR
sets the default request type for printing, punching, or plotting to STR. The print default request type is set by default). STR must be a request type or the name of a group of control argument settings.

It is sometimes desirable to define several different groups of control argument settings associated with the same request type. For example, you might want to indent only segments containing Multics mail. Several groups can be associated with the same request type by using the following control argument along with -request_type.

-default_name STR, -dnm STR
uses STR as the name for a new group of control argument settings being defined. Use -default_name when the name of the new group differs from the request type defined by the group. Subsequent references to the group are made by using -request_type (i.e., the new group name becomes a user-defined request type).

ACCESS REQUIRED

The IO Daemon process that performs the printing or punching must have at least r access to the entry and at least s access to the directory that contains the entry. Use the print_request_types command to print the access name of IO Daemon processes.

If -delete is given, the IO coordinator (normally IO.SysDaemon.z) must have at least m access to the directory that contains the entry.

NOTES

If eor is invoked without arguments, it gives the status of the default printer request type.

EXAMPLES

The following examples illustrate how groups of control argument settings can be defined, modified, printed, or deleted.

```
! eor -rdft -rqt cisl_prt -nt -q 2 -ind 10
```

resets the cisl_prt group to the standard control argument values and then applies the -nt, -q, and -ind settings to change the defaults. You can print the new defaults by

```
! eor -rqt cisl_prt -pdft

   cisl_prt:
     -rqt cisl_prt  -print
     -he "LvBeethoven"
     -ds "SysLib"
     -nt  -q 2  -ind 10  -albl
```

You can make the cisl_prt group the default request type for printing by

```
! eor -sdrqt cisl_prt
```

When you invoke eor without control arguments, the cisl_prt control argument settings are used. You can now modify and print these defaults by

```
! eor -ind 0 -sdft -pdft

   cisl_prt: (default for printing)
     -rqt cisl_prt  -print
     -he "JSBach"
     -ds "SysLib"
     -nt  -q 2  -albl
```

You don't have to give the cisl_prt request type in the command line because it is now the default request type for printing. Several groups of defaults can apply to the same request type by using -default_name. For example, to define a request type to print mail, indented by 20, use

```
   eor -rdft -dnm mail -rqt printer -ind 20 -pdft -q dft
```

```
mail:
  -rqt printer  -print
  -he "LdVinci"
  -ds "SysLib"
  -q 3 (default)  -ind 20  -albl
```

Segments with a mail suffix can be printed with the mail defaults by

```
eor -rqt mail **.mail
```

The mail defaults can be modified by

```
eor -rqt mail -sdft -nt -dl -pdft

mail:
  -rqt printer  -print
  -he "LdVinci"
  -ds "SysLib"
  -q 3 (default)  -dl  -nt  -ind 20  -albl
```

The known groups of defaults (and their associated request types) can be listed by

```
eor -ldft

Defaults for printing:
  printer
  cisl_prt  (default)
  mail  -rqt printer
Defaults for punching:
  punch  (default)
Defaults for plotting:
  plotter (default)
```

All the groups of control argument settings can be printed by

```
eor -pdft -a

cisl_prt:  (default for printing)
  -rqt cisl_prt  -print
  -he "LdVinci"
  -ds "SysLib"
  -nt  -q 2  -albl

mail:
  -rqt printer  -print
  -he "LdVinci"
  -ds "SysLib"
  -dl  -nt  -q 3 (default)  -ind 20  -albl
```

```
printer:
  -rqt printer  -print
  -he "LdVinci"
  -ds "SysLib"
  -q 3 (default)  -albl

punch: (default for punching)
  -rqt punch  -punch
  -he "LdVinci"
  -ds "SysLib"
  -q 1 (default)  -mcc

plotter: (default for plotting)
  -rqt plotter  -plot
  -he "LdVinci"
  -ds "SysLib"
  -q 1 (default)
```

Name: enter_retrieval_request, err

SYNTAX AS A COMMAND

```
err path {-control_args}
```

FUNCTION

queues volume retrieval requests for specific segments, directories, multisegment files (MSFs), and subtrees.

ARGUMENTS

path
is the pathname of a segment, directory, or node of a subtree. The star convention is not allowed.

CONTROL ARGUMENTS

-brief, -bf
supresses printing of the ID and number of requests in queue.

-from DT, -fm DT
specifies that the search for path and all inferior branches, if supplied, stops at time DT; thus, objects dumped before time DT are not recovered. (See Section 1 for a description of valid DT values.) If you give no -from, all valid dump volumes are searched.

-long_id
prints the long ID of the request. (Default: to print the short ID)

-multisegment_file, -msf
specifies that the object named in path is an MSF and that all its components are to be recovered.

-new_path newpath
specifies that if you have the correct access to retrieve the segment specified in path and to create a segment with the pathname newpath, then the object identified by path is retrieved into newpath. You can't cross-retrieve directories, MSFs, or subtrees.

-notify, -nt
notifies you by online mail of the success or failure of the request. (Default: not to notify you)

-previous, -prev
retrieves the object dumped prior to the object presently online. With -prev you can retrieve successively earlier copies of an object. (Default: to retrieve the most recent copy)

-queue N, -q N
queues requests in priority queue N. (Default: 3)

-subtree, -subt
retrieves the subtree inferior to the directory given in path as well as the directory. If a subtree is found intact after a directory is recovered, no further action is taken unless you have provided a time interval (see "Notes"). (Default: not to retrieve subtrees)

-to DT
searches for path and all inferior branches from time DT backwards; thus, objects dumped later than time DT are not recovered. (See Section 1 for a description of valid DT values.) If you don't select -to, time DT is assumed to be the start of the retrieval operation.

ACCESS REQUIRED

To retrieve a segment, you need w access to the segment or m access to the containing directory; to retrieve a directory, you need m access to the directory or m access to the containing directory.

NOTES

In certain cases where a directory is damaged the inferior subtree may be unavailable until the directory is recovered. When a directory is recovered and you use -subt, a check is made to see if the subtree is available, and, if so, retrieval is assumed complete.

Retrieval requests of objects for which the online copy is more recent or the same as the dump copy are refused unless you use -fm, -prev, or -to.

You need not supply as a set of primary names the pathnames of the segments and directories to be retrieved. Any set of valid entrynames is acceptable.

You have to log in to ring 1 to submit retrieval requests for mailboxes and other ring 1 objects.

Name: entries

SYNTAX AS A COMMAND

```
entries star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[entries star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of segments, directories, multisegment files (MSFs), links, data management (DM) files, and extended entries that match one or more star names.

ARGUMENTS

star_names
: are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
: returns absolute pathnames rather than entrynames.

-chase
: processes the targets of links when you specify a starname.

-inhibit_error, -ihe
: returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
: does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

-select_entry_type type_name, -slet type_name
returns entrynames of entries of the specified type. You need not give the suffix in the starname. Use the list_entry_types command to obtain a list of valid entry type values.

NOTES

Only one name per entry is returned; i.e., if an entry has more than one name that matches a starname, only the first match found is returned.

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

See the directories, directory, and entry commands.

This page intentionally left blank.

Name: entry

SYNTAX AS A COMMAND

```
entry path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[entry path]
```

FUNCTION

returns the entryname portion of path, after it has been expanded into an absolute pathname.

ARGUMENTS

path
is the pathname whose entryname portion is to be returned.

NOTES

Since the pathname is returned in quotes, the command processor treats it as a single argument regardless of special characters in the name.

EXAMPLES

```
! entry >udd>Proj>Myname>start_up.ec
  start_up.ec

! entry >udd>Multics>Library>Source>bound_command_demos_.s::program.pl1
  bound_command_demos_.s.archive
```

Name: entry_path

SYNTAX AS A COMMAND

```
entry_path path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[entry_path path]
```

FUNCTION

returns the absolute pathname of the entry represented by the path argument. If the path is an archive component pathname, this returns the pathname of the archive segment; otherwise this command is equivalent to the path command.

ARGUMENTS

path
is the pathname whose directory and entryname portion is to be returned as a single absolute pathname.

NOTES

Since the pathname is returned in quotes, the command processor treats it as a single argument regardless of special characters in the name.

EXAMPLES

```
! entry_path [hd]>start_up.ec
  >udd>Proj>Myname>start_up.ec

! entry_path >udd>Multics>Library>Source>bound_command_demos_.s::program.pll
  >udd>Multics>Library>Source>bound_command_demos_.s.archive
```

Name: equal

SYNTAX AS A COMMAND

```
equal STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[equal STRA STRB]
```

FUNCTION

returns true if strA is equal to strB; otherwise it returns false.

ARGUMENTS

STRA, STRB
are character strings to be compared.

NOTES

The strings are compared character by character according to their ASCII code value (i.e., if the first character in each string has the same ASCII code value, compare the second character; if their values are identical, compare the third character; etc.). Strings of unequal length are compared by padding with blanks.

EXAMPLES

```
! string [equal Ab ab]
  false

! string [equal this this]
  true
```

Name: equal_name, enm

SYNTAX AS A COMMAND

```
enm path equal_path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[enm path equal_path]
```

FUNCTION

returns a pathname, constructed by applying the equal convention to the specified arguments.

ARGUMENTS

path
: is the source pathname to which the equal convention is applied. You can use the archive component pathname convention.

equal_path
: is a pathname whose entryname and component name portions are equal names.

NOTES

Since the pathname is returned in quotes, the command processor treats it as a single argument regardless of special characters in the name.

With the active function you can apply equal names within abbreviations and exec_com segments.

For a complete description of the equal convention see the Programmer's Reference Manual.

EXAMPLES

```
! enm apple.ec orange.==
  orange.ec

! enm fruits.archive::apple.ec orange.==
  orange.ec
```

Name: exec_com, ec (version 2)

SYNTAX AS A COMMAND

```
ec {-control_args} path {ec_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ec {-control_args} 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, so make the first line of the exec_com program be "&version 2". For a description of Version 1 syntax, see exec_com (version 1).

ARGUMENTS

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

ec_args
: are optional arguments to the exec_com program and are substituted for parameter references such as &1 (see "List of Parameters").

CONTROL ARGUMENTS

-no_trace KEYWORD_LIST
turns off tracing of specified types of exec_com lines, overriding any &trace statements in the exec_com for those types of lines. KEYWORD_LIST is composed of any of the keywords "all_types", "command", "comment", "control", and "input", separated by commas with no intervening space.

-trace KEYWORD_LIST
turns on tracing of specified types of exec_com lines, overriding any &trace statements in the exec_com for those types of lines. KEYWORD_LIST is composed of any of the following, separated by commas, with no intervening space:

all_types, command, comment, control, input
turns on tracing for the corresponding type of line(s).

unexpanded, expanded, all_expansions or all, both
affects how the expansion of lines is traced. These are equivalent to &unexpanded, &expanded, &all_expansions or &all, and &both in &trace statements inside the exec_com.

prefix=STR
specifies a prefix for traced lines, equivalent to &prefix in &trace statements.

osw=SWITCHNAME
specifies an I/O switch on which to write the trace, equivalent to &osw in &trace statements.

-trace_default
uses &trace statements in the exec_com and the default tracing modes to determine what and how to trace. (Default)

LIST OF PARAMETERS

&1 - &9
expands to the 1st through 9th ec_args or to defaults defined by a &default statement or to the null string if there is no corresponding ec_arg. The string &0 is invalid.

&(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.

exec_com (version 2)

&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 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 given to exec_com as is, with quotes doubled, or requoted.

&condition_info_ptr, &cond_info_ptr
inside an &on unit, expands to a virtual pointer (<segment_number>|<offset>) to the condition_info structure for the condition that is signalled (see "List of Condition-Handling Statements"). Programs can be written to interpret the structure for a particular condition.

&condition_name, &cond_name
inside an &on unit, expands to the name of the condition that caused the &on unit to be invoked (see "List of Condition-Handling Statements"). Outside an &on unit, expands to null string.

exec_com (version 2)

&ec_dir
expands to the pathname of the directory containing the exec_com currently running. All links in the pathname have been chased. You can use this construct to call other exec_com's in the same directory: "ec &ec_dir>foo".

&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.

&ec_path
expands to the expanded, suffixed pathname of the current exec_com. Unlike &ec_dir, links in the pathname have not been chased.

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

&handlers
expands to a list of condition names for which &on handlers are currently in effect (see "List of Condition-Handling Statements"). Condition names are individually requoted and separated by spaces. To test whether a handler is currently in effect for NAME, type: "&if &[or [equal NAME (&handlers)]] &then ..."

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 indicated, consequently, 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 and white space.

&(N)
expands to the value of the Nth (where N is a positive integer) 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.

exec_com (version 2)

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

&[ACTIVE STRING], &||[ACTIVE STRING]
expands to the return value of an active string by calling the command processor. This construct ends with the matching right bracket. The &||[...] construct is used in &set statements to treat the expansion as a single argument to &set. It is important to note that &[...] active strings are expanded by exec_com, whereas [...] strings are expanded at command line execution time. Therefore, ||[...], not &||[...], must be used in a command line to treat the expansion as a single command argument.

LIST OF LITERALS
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.

exec_com, ec (version 2)

&!
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. This construct expands to "true" if &(NAME) can be expanded, "false" if &(NAME) is an error.

&is_defined(N)
expands to "true" if an Nth (where N is a positive integer) ec_arg is given to exec_com or an Nth default is defined by the &default statement (see "List of Assignment Statements" below), "false" otherwise.

&is_absin
expands to "true" if the exec_com is being executed by the absentee facility, "false" if it is by the exec_com command or active function. In the case of an absentee executing the start_up.ec, this value is returned as "false" because it is being executed by the exec_com command. The command "ear program" causes the absentee listener to execute program.absin, and then &is_absin returns "true."

&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 by an &attach statement, "false" otherwise (see "Notes on Input Attachment" below). 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.

&was_attached
inside an &on unit, expands to "true" if the parent exec_com was attached by &attach at the time the condition occurred, "false" otherwise (see "List of Condition-Handling Statements" below). Outside an &on unit, always expands to "false."

exec_com, ec (version 2)

LIST OF CONTROL STATEMENTS

&attach {&trim on/off}
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" below). Specifying "&trim off" causes the input lines to be read intact, without stripping off the leading and trailing white space as is done with most exec_com lines. (Default: "&trim on")

&detach
causes any commands subsequently invoked in command lines to read their input from the terminal (see "Notes on Input Attachment" below). (Default)

&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" above). 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" below).

&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 "Examples of if Statements" below). The contents of an &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
designates 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 ("").

exec_com (version 2)

&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 quotes, for example:

```
&set answer "&[response Answer?]"
```

Alternatively, the &||[...] construct can be used, causing the entire return value to be taken as a single argument:

```
&set answer &||[response Answer?]
```

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 expands 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.

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 CONDITION-HANDLING STATEMENTS

&on CONDITION_LIST &begin LINES &end
establishes a condition handler (&on unit) to be invoked whenever any of the conditions named in CONDITION_LIST is signaled. Condition names are separated by white space. LINES is any sequence of exec_com lines, optionally including &goto statements to transfer to labels either inside the &on unit or outside (i.e., in the parent exec_com). When executed, LINES is treated as a separate exec_com in the sense that changes to its &attach, &ready_proc, and &trace modes (initially off) do not affect the parent exec_com. However, &on units share the parent ec's variables, and any changes to variables affect the parent exec_com. The &begin and &end keywords are required for delimiting LINES, even if it consists of a single line. No &quit statement is required.

&revert CONDITION_LIST
reverts any &on units for the conditions named in CONDITION_LIST. Condition names are separated by white space.

&signal CONDITION_NAME
signals the indicated condition.

The following statement is allowed only inside &on units:

&exit {&continue}
causes the &on unit to exit immediately. This statement is useful for conditionally exiting part-way through an &on unit. If &continue is supplied, the condition continues to be propagated to other handlers down the stack.

LIST OF TRACING STATEMENTS

&list_variables {match_names} {&control_args},
&lsv {match_names} {&control_args}
lists the values of all or selected exec_com variables, where match_names are starnames and/or qedx regular expressions surrounded by slashes (/). Control arguments are "&exclude match_name" ("&ex match_name") to prevent certain names from being listed, &variable (&var) to list just the variable names, and &value (&val) to list just the values.

&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). This statement is ignored in the absentee environment.

&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. This statement is ignored in the absentee environment.

&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
&all_types	specifies all of &command, &comment, &control, and &input.

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". * MCR 6691
&all_expansions	is a synonym for &all.
&both	prints each line as it appears in the exec_com, and again after all expansion. Implies "on".

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 designates 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.

exec_com (version 2)

NOTES ON ABSENTEE ENVIRONMENT

An exec_com/absin runs in the absentee environment only when it has been invoked directly by the absentee facility, i.e., 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 absin 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 (V2). In many ways similar to the old Version 1 (V1), 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. Second, V2 parses lines into control keywords and tokens (separated by white space) before expansion, so that expansion can only change the values of tokens but not the syntax of a line.

A V2 exec_com has "&version 2" as its first line. If this first line is not present, the exec_com is interpreted as V1. V1 exec_com's can optionally begin with "&version 1". At some future time, V2 will be the default and "&version 1" will be required.

A conversion command is available to translate V1 exec_com's to V2: convert_ec.

NOTES ON WHITE SPACE

White space (SPACE, HORIZONTAL TAB, VERTICAL TAB, and FORM-FEED) is ignored at the beginning and end of each line, with the exception of input lines specifically read with "&attach &trim off" in effect. As a result, exec_com lines can be indented as desired for readability. 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".

exec_com (version 2)

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, and can be specified by the literal escapes described in "List of Literals." Therefore, comments can be aligned at a particular column without affecting the executable text.

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 white space following the &+ is part of the executable line, and in the above example it 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 ("...").

exec_com (version 2)

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 "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 results 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 significant; 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 command 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 is 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
```

exec_com (version 2)

EXAMPLES OF IF STATEMENTS

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

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

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

More examples:

```
&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

exec_com (version 2)

&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
&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

exec_com (version 1)

Name: exec_com, ec (version 1)

SYNTAX AS A COMMAND

```
ec path {optional_args}
```

FUNCTION

executes a sequence of command lines contained in a segment. It allows you to construct command sequences that are invoked frequently without retyping the commands each time. In addition, you can use control strings to substitute argument values into the executed text, manage I/O switches, and execute portions of the text conditionally.

This section describes Version 1 exec_com, which has effectively been replaced by Version 2. The first line of a Version 1 exec_com can optionally be "&version 1"; 1 is currently the assumed version.

ARGUMENTS

path
: is the pathname of a segment containing commands to be executed and control statements to be interpreted. The entryname of the segment must have the ec suffix, although you can omit the suffix in the command invocation. If you supply an entryname only, i.e., one containing no < or > characters, the exec_com search list is used to locate the segment. (See "Notes on Search List" below.)

optional_args
: are character strings to be substituted for special strings in the exec_com segment (see "Notes on Argument Substitution.")

CONTROL ARGUMENTS

-no_trace KEYWORD_LIST
: turns off tracing of specified types of exec_com lines, overriding any &trace statements in the exec_com for those types of lines. KEYWORD_LIST is composed of any of the keywords "all_types", "command", "comment", "control", and "input", separated by commas with no intervening space.

-trace KEYWORD_LIST
: turns on tracing of specified types of exec_com lines, overriding any &trace statements in the exec_com for those types of lines. KEYWORD_LIST is composed of any of the following, separated by commas, with no intervening space:

exec_com (version 1)

all_types, command, comment, control, input
turns on tracing for the corresponding type of line(s).

unexpanded, expanded, all_expansions or all, both
affects how the expansion of lines is traced. These are equivalent to &unexpanded, &expanded, &all_expansions or &all, and &both in &trace statements inside the exec_com.

prefix=STR
specifies a prefix for traced lines, equivalent to &prefix in &trace statements.

osw=SWITCHNAME
specifies an I/O switch on which to write the trace, equivalent to &osw in &trace statements.

-trace_default
uses &command_line, &comment_line, &control_line, and &input_line statements in the exec_com and the default tracing modes to determine what and how to trace. (Default)

NOTES ON INPUT SEGMENT

The exec_com segment should contain only command lines, input lines, and control statements. Normally it is created using a text editor, such as qedx. You can use the exec_com command in conjunction with the abbrev command to form abbreviations for command sequences that are used frequently.

When the ampersand character (&) appears in the exec_com segment, it is interpreted as a special character: it denotes a string used for argument substitution and to signify the start of a control statement.

NOTES ON ARGUMENT SUBSTITUTION

Strings of the form &i in the exec_com segment are interpreted as dummy arguments and are replaced by the corresponding arguments to the exec_com command; for instance, optional_arg1 is substituted for the string &1 and optional_arg10 is substituted for &10. The strings &qi, &ri, &fi, &qfi, and &rfi also indicate argument substitution. The string &qi is replaced by the i'th argument to the exec_com command with quotes doubled. The string &ri is replaced by the i'th argument, requoted. Refer to do in this manual for a description of quote doubling and requoting and for examples of the use of &qi, &ri, &fi, &qfi, and &rfi. The string &fi is replaced by a string of the i'th through last arguments to exec_com, separated by blanks. Likewise, &qfi is replaced by a string of the i'th through last arguments with quotes doubled and &rfi is replaced by a string of the i'th through last arguments, requoted.

exec_com (version 1)

The string &n is replaced by the number of arguments to the exec_com command. The string &f&n, therefore, is replaced by the last argument to exec_com. The string &ec_name is replaced by the entryname portion of the exec_com pathname without the ec suffix. The string &ec_dir is replaced by the directory name portion of the exec_com pathname. The string &ec_switch expands to the name of the I/O switch through which the exec_com is being read.

Argument substitution can take place in command lines, input lines or in control statements, since the replacement of arguments is done before the check for a control statement.

LIST OF PREDICATES

The following predicates expand to true or false:

&is_active_function, &is_af
expands to "true" if exec_com was invoked as an active function.

&is_absin
expands to "true" if the current exec_com is running as an absentee.

&is_attached
expands to "true" if &attach is currently in effect.

&is_input_line
expands to "true" if some program is currently reading input lines under &attach, "false" if lines are interpreted as command lines.

LIST OF CONTROL STATEMENTS

Control statements permit more variety and control in the execution of the command sequences. Currently the control statements are: &label, &goto, &attach, &detach, &input_line, &command_line, &ready, &print, &quit, &if, &then, and &else.

Control statements generally must start at the beginning of a line with no leading blanks. Two exceptions to this rule are the &then statement, which can follow an &if clause, and the &else statement, which can follow a &then clause. Any control statement other than &label, &if, &then, and &else is allowed to follow the control words &then and &else.

&label and &goto

These statements permit the transfer of control within an exec_com segment.

&label LOCATION
identifies the place to which a goto control statement transfers control. The LOCATION is any string of 32 or fewer characters, unique within the exec_com segment.

&goto LOCATION
causes control to be transferred to the place in the exec_com segment specified by the label LOCATION. Execution then continues at the line immediately following the label.

&attach, &detach, and &input_line

&attach
causes the user_input I/O switch to be attached to the exec_com segment. This means that if this control statement is executed, all input read by subsequent commands is taken from the segment rather than from the previous source of data to which the user_input I/O switch was attached.

&detach
causes the user_input I/O switch to be reverted to its original value. By default, the user_input I/O switch is left attached to its original source.

&input_line on
causes input lines returned when using the attach feature to be written on the user_output I/O switch. This is the default.

&input_line off
causes input lines to not be written out.

Tracing, &ready, and &print

These statements allow the control of the user_output I/O switch. They are useful as tools in observing the progress of the exec_com execution and in printing messages.

&command_line on {osw SWITCHNAME}
&command_line off {osw SWITCHNAME}
causes subsequent command lines to be written on the user_output I/O switch or on another specified SWITCHNAME before they are executed. The "off" usage causes subsequent command lines to not be written out.

&comment_line on {osw SWITCHNAME}
&comment_line off {osw SWITCHNAME}
controls tracing of comment lines, lines beginning with "& ". The default if no SWITCHNAME is specified is user_output.

&control_line on {osw SWITCHNAME}
&control_line off {osw SWITCHNAME}
controls tracing of &if, &goto and all other exec_com control statements. The default if no SWITCHNAME is specified is user_output.

&input_line on {osw SWITCHNAME}
&input_line off {osw SWITCHNAME}
controls tracing of lines read as input lines under &attach. The default if no SWITCHNAME is specified is user_output.

exec_com (version 1)

&ready on
&ready off
causes your ready procedure to print a ready message whenever it is invoked after the execution of a command line. The "off" usage causes your ready procedure not to print ready messages.

&ready_proc on
&ready_proc off
controls whether your ready procedure is called after each command line. The default is "on". This mode is completely independent of "&ready".

&print char_string
causes the character string following &print to be written out on the user_output I/O switch. The character ^ is treated as a special character in a print statement. The following is a list of strings that can appear and the characters that replace them:

^/ or ^N/	newline character
^| or ^N|	form feed (new page)
^- or ^N-	horizontal tab
^^	^

where N expresses the number of special characters to be written out. No other characters should appear following the ^ character in the print statement.

&quit

This statement causes the current invocation of exec_com to return to its caller and not to execute subsequent command lines. If exec_com has been invoked as an active function, the return value is the null string.

&return rest of line
Equivalent to &quit but returns a value. If exec_com was invoked as an active function, the rest of the &return line is returned as the value. Otherwise, the rest of the line is printed before quitting.

&if, &then, and &else

These statements provide the ability to have command lines, input lines, and control statements interpreted conditionally.

The format of these control statements is

&if [ACTIVE_FUNCTION {arg1} ... {argn}]
&then THEN_CLAUSE
&else ELSE_CLAUSE

The active function reference in an &if control statement is evaluated. If the value of the active function is the string true, THEN_CLAUSE is executed. If the value is false, ELSE_CLAUSE is executed.

exec_com (version 1)

&if [ACTIVE_FUNCTION {arg1} ... {argn}]
The active function is any active function (user-provided or system-supplied) that returns as its value a string with the value true or false. The arguments to the active function can themselves be active functions. (Nesting of active functions is permitted.) The active function and its optional arguments, enclosed in brackets, must be on the same line as the &if string. An &if must begin a line or immediately follow &then or &else, as in the example:

```
&if [equal &1 tape]
  &then &if [equal &2 hdr] . . .
```

&then THEN_CLAUSE
This statement must immediately follow the &if statement; it can appear on the same line or on the following line. THEN_CLAUSE is an exec_com statement, and can include a command line, an input line, the null statement and most control statements. The &label, &then, and &else control statements are not allowed. THEN_CLAUSE must be on the same line as &then.

&else ELSE_CLAUSE
This statement is optional. When it appears, it must immediately follow the &then statement; it can appear on the same line or on the following line. ELSE_CLAUSE is an exec_com statement and can include a command line, an input line, the null statement and most control statements. The &label, &then and &else control statements are not allowed. ELSE_CLAUSE must be on the same line as &else.

NOTES ON SEARCH LIST

The exec_com command uses the exec_com search list that has the synonym ec. Type:

```
! psp ec
```

to see what the current exec_com search list is. The default exec_com search list is the working directory. For more information on the search facility, see the description of the add_search_paths command in this manual.

NOTES ON HANDLING CONDITIONS

The on command and active function can be used to handle conditions raised during the execution of an exec_com. To handle command_error when executing the copy command, for example, an exec_com can say:

```
&if [on command_error "" -bf copy PROJ_DIR>&1 MY_DIR>=]
  &then &goto copy_failed
an MY_DIR>&1 &1.[date]
...
&label copy_failed
&print PROJ_DIR>&1 not copied
...
```

exec_com (version 1)

The -bf control argument suppresses a message printed by on when the condition is raised.

The discard_output command can be used to suppress output from the command whose success is being tested, for example:

```
&if [on command_error "" -bf dco -osw error_output -osw
  user_output archive tb source &1.pll]
&then &goto no_component
&print &1.pll in source.archive
...
&label no_component
&print &1.pll not found in source.archive
...
```

The on command can be used to execute another exec_com, or a recursive entry point in the current one, with a handler in effect. For example:

```
on any_other "ec handler" ec test_ms
...
&quit
&label handler
tmr mail mbsa mailbox_
in >sss>mail
&quit
&label test_ms
tmr mail mbsa mailbox_
in MS>mailbox_
MS>mbsa test.mbx adros
MS>mail test
&quit
```

For more information, see the description of the on command.

NOTES ON HANDLING QUESTIONS

The answer command can be used to supply preset answers to questions asked by commands invoked in an exec_com. (It is not recommended that answers be supplied on successive lines of the exec_com with &attach on.) The following exec_com prints only the first three sections of an info segment by answering "yes" twice and then "no":

```
answer yes -times 2 -then no help &1
&quit
```

The following example prints the first three sections of an info segment, then prints the next three only if your answer yes:

exec_com (version 1)

```
answer yes -times 2 -then -query -then yes -times 2
   -then no help &1
&quit
```

For more information, see the description of the answer command in this manual.

NOTES

If a line begins with the & character but is not one of the current control statements, the entire line is ignored. This is one way of including comments in the exec_com segment. You are cautioned to leave a blank immediately following the & to ensure compatibility with control requests to be added to exec_com in the future.

The segment executed by exec_com can contain calls to exec_com. You must exercise caution when invoking this feature in conjunction with the &attach feature. When exec_com is called from an exec_com using this feature, the input read by commands in the second exec_com is read from the first exec_com segment. Generally, if the &attach feature is used, all calls to exec_com should be preceded by &detach control statements.

Several exec_coms can be combined into one segment, by using the dummy argument &ec_name together with the &label and &goto statements. If exec_coms are grouped together, the exec_com segment should have all the names (concatenated with an ec suffix) on its storage system entry that can replace &ec_name.

EXAMPLES

Assume that the segment a.ec in your working directory contains

```
pl1 &1 -table -list
dprint -delete &1.list
&quit
```

The command line

```
! exec_com a foo
```

causes the following commands to be executed:

```
pl1 foo -table -list
dprint -delete foo.list
```

exec_com (version 1)

Assume that the segment b.ec in your working directory has an additional name a.ec and contains

```
&goto &ec_name
&
&label b
print &1 1 99
&quit
&
&label a
pl1 &1 -table -list
dprint -delete &1.list
&quit
```

The command line

```
! exec_com b my_file
```

causes the following command to be executed:

```
print my_file 1 99
```

The command line

```
! exec_com a foo
```

causes the following commands to be executed:

```
pl1 foo -table -list
dprint -delete foo.list
```

Assume that the segment d.ec in your working directory contains the following:

```
&if [exists segment &1.pl1] &then
&else &goto not_found
pl1 &1 -table -list
dprint -delete &1.list
&quit
&label not_found
&print &1.pl1 not found
&quit
```

If the segment foo.pl1 exists, the command line

```
! exec_com d foo
```

causes the following commands to be executed:

```
pl1 foo -table -list
dprint -delete foo.list
```

exec_com, ec (version 1)

If the segment foo.pl1 does not exist, the command line

```
!  exec_com d foo
```

outputs the following:

```
foo.pl1 not found
```

Assume that the segment test.ec in your working directory contains

```
&print begin &ec_name exec_com
&command_line off
create &1.pl1
&attach
edm &1.pl1
i &1: proc;
&input_line off
i end &1;
w
q
&detach
&goto &2
&label compile
pl1 &1
&label nocompile
&print end &ec_name &1 &2 exec_com
&quit
```

The command line

```
!  exec_com test x compile
```

produces the following output:

```
begin test exec_com
Edit.
i x: proc;

PL/I
end test x compile exec_com
```

LIST OF CONSTRUCTS

This is an alphabetical list of version 1 exec_com constructs:

&attach
&command_line
&comment_line
&control_line
&detach
&else
&goto
&if
&input_line
&is_absin
&is_active_function, &is_af
&is_attached
&is_input_line
&label
&print
&quit
&ready
&ready_proc
&return
&then

Name: execute_string, exs

SYNTAX AS A COMMAND

```
exs {-control_args} {control_string {args}}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[exs {-control_args} control_string {args}]
```

FUNCTION

substitutes arguments into a control string. The expanded control string is then passed to the command processor or the subsystem request processor for execution. As an active function or active request, evaluates the expanded control string as an active function.

ARGUMENTS

control_string
is a character string that can contain substitution constructs (see "List of Substitutions" below).

args
are zero or more character string arguments. Any argument supplied but not referenced by an argument substitution designator is ignored.

CONTROL ARGUMENTS AS A COMMAND

If you give control arguments with no control string, subsequent exs invocations in the process are affected; with a control string and its arguments, subsequent exs invocations are not affected. Give the control arguments first. (See "Notes on modes" below.)

-abort_line, -abl
aborts the line containing the exs request if the request line is aborted during execution. Applies only to subsystem request invocations of exs. (Default)

-brief, -bf
does not print the expanded control string. (Default)

-control_string, -cs
permits a control string to look like a control argument.

-go
passes the expanded control string to the command processor or subsystem request processor. (Default)

-inhibit_error, -ihe, -absentee
establishes a handler for the any_other condition during the execution of the expanded command control string.

-long, -lg
prints the expanded control string on error_output before executing or returning it.

-no_abort_line, -nabl
continues execution with the next request following the exs request on the same line if the request line exs invoked is aborted during execution. Applies only to subsystem request invocations of exs.

-no_inhibit_error, -nihe, -interactive
does not catch any signals. (Default)

-nogo
does not pass the expanded control string to the request processor.

CONTROL ARGUMENTS AS AN ACTIVE FUNCTION

-brief, -bf
does not print the expanded active string. (Default)

-control_string, -cs
permits a control string to look like a control argument.

-error_value CONTROL_STRING, -erv CONTROL_STRING
evaluates and returns the expanded control string if an error occurs, where CONTROL_STRING is a character string that can contain substitution constructs. In a subsystem active request, an error is anything that aborts the line; in an active function, anything that raises the active_function_error condition; in inhibit-error mode, any condition that -inhibit_error would handle as a command. (See "Notes on modes" below.)

-inhibit_error, -ihe
establishes a handler for the any_other condition during the execution of the expanded control string. Valid only if you give -erv.

-long, -lg
prints the expanded control string on error_output before it is evaluated.

-no_inhibit_error, -nihe
does not catch any signals. (Default)

-no_rescan, -nrsc
does not permit the command processor to rescan the result of the active function for white space, semicolons, parentheses, or brackets. This is equivalent to the ||[...] evaluation, but the result is not protected from reevaluation after exs returns the result, unless ||[...] also encloses the exs invocation. (Default)

-rescan, -rsc
permits the command processor to rescan the result of the active function evaluation.

-rescan_tokens, -rsct
permits the command processor to rescan only the active function result for white space and quotes. This is similar to |[...] evaluation, in that it strips a level of quotes, but it concatenates the tokens back together without requoting, so that information may be lost. Use -no_rescan and place the |[...] around the execute_string invocation to retain this information.

LIST OF SUBSTITUTIONS

The following expansion designators appearing in the control string are replaced by their expansion value, as described below. Any other use of the ampersand (&) produces an error.

&0, &1,...&9
expands to the zeroth through ninth arguments. &0 is the control string, &1 is the first argument following the control string, and so on. If the corresponding argument is missing, the designator expands to a null string.

&(0), &(1),...
expands to any argument, including arguments after the ninth. Use parenthesis when the argument number is two or more digits. If the corresponding argument is missing, the designator expands to a null string.

&q0,...&q9, &q(0), &q(1),...
expands to the corresponding argument following the control string. Quotes within the argument are doubled, according to the quote depth of the surrounding context within the control string (see "Notes on Quote Doubling" below).

&r0,...&r9, &r(0), &r(1),...
expands to the corresponding argument following the control string, enclosed in an added layer of quotes with internal quotes with the argument doubled accordingly (see "Notes on Requoting" below). This designator keeps the argument as a single unit after one layer of quote stripping by the command processor.

&f1,...&f9, &f(1),...
expands to the Nth through last arguments following the control string, with arguments separated by one space. If N is greater than &n, expands to a null string.

&qf1,...&qf9, &qf(1),...
expands to the Nth through last arguments following the control string, with quotes doubled within arguments, and arguments separated by one space. If N is greater than &n, expands to a null string.

&rf1,...&rf9, &rf(1),...
expands to the Nth through last arguments following the control string, with each argument individually requoted, and arguments separated by one space. If N is greater than &n, expands to a null string.

&n
expands to the number of arguments you give following the control string.

&f&n, &qf&n, &rf&n
expands to the last argument following the control string, with quotes doubled (&qf&n) or with requoting (&rf&n).

&control_string
expands to the control string (without expansions), with quotes doubled. It is equivalent to &q0.

&!
expands to a unique name. Each use of &! is replaced by a 15-character identifier. Every use within a single invocation is replaced by the same string, but the string is different for every invocation of exs.

&&
expands to a single ampersand, to allow ampersands to be literally inserted into the expanded control string.

NOTES

This command is similar to the do command. The do command is an older interface that acts like exs as a command and like substitute_arguments as an active function.

When the control string is executed, abbreviations are expanded if the abbrev processor is enabled. Since the control string is usually enclosed in quotes, abbreviations in the control string are not expanded until control string expansion. (See the abbrev command.)

NOTES ON MODES

This command has four modes: the long/brief mode, the nogo/go mode, the abort-line mode, and the inhibit-error mode. These modes are kept in internal static storage and are thus remembered from one invocation of exs to the next in a single process. Set the modes for the life of the process by invoking exs with control arguments and no control string; set the modes for a single invocation by giving control arguments, a control string, and its arguments.

The abort-line mode applies only to subsystem request invocations of exs. You can set the mode at command level, but cannot set it for a single command invocation of exs.

Use the inhibit-error mode mainly in an absentee environment, in which any condition that normally enters a new command level terminates the process. In this mode, any signal caught by exs terminates execution of the command line, not the process. The following conditions are not handled by exs, however, but are passed on to the command processor: command_error, command_query_error, command_question, program_interrupt, quit, and record_quota_overflow (see the Programmer's Reference Manual).

The abort-line and go/nogo modes have no effect on active function and active request invocations of exs. The active function is always evaluated, and execution of the containing command line cannot continue if there is no active function result. The inhibit-error mode is ignored for active function evaluation if you give no -erv.

The modes of the exs command are separate from the modes of the do and substitute_arguments commands, although they provide similar functions.

NOTES ON QUOTE DOUBLING

Each parameter designator to be expanded is found nested a certain level deep in quotes. If it is found to be outside quotes, its quote level is zero; if found between a single pair of quotes, its quote level is one; and so on. If an "&q" construct is found nested to quote-level L, then, as the argument is substituted into the expanded control string, each quote character found in the argument 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 control string. If the "&q" construct is not between quotes, or if the corresponding argument contains no quotes, quote doubling has no effect.

NOTES ON REQUOTING

If an "&r" construct is found, the substituted argument is placed between an additional level of quotes before having its quotes doubled. For example, if &r1 is found nested to quote level L, 2**L quotes are inserted into the expanded control string; then, the first argument is substituted, with each of its quotes replaced by 2**(L+1) quotes; and, finally, 2**L more quotes are placed following it. If you give no argument, nothing is placed in the expanded control string; so, you can distinguish between arguments that are not supplied and arguments that are supplied but are null. If you give an argument, the expansion of an "&r" construct is identical to the expansion of an "&q" construct surrounded by an extra level of quotes.

Name: exists

SYNTAX AS A COMMAND

```
exists argument {str_args}

exists key star_name{s} {-control_arg{s}}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[exists argument {str_args}]

[exists key star_name{s} {-control_arg{s}}]
```

FUNCTION

checks for the existence of various types of items depending on the value of the first argument (key).

ARGUMENTS

argument
is the key "argument" described below in "List of Keys."

str_args
are character string arguments.

key
is any key as described below in "List of Keys."

star_name{s}
are star names to be matched. You can give up to 20 names.

CONTROL ARGUMENTS

-chase
specifies that any keyword that looks for branch entries chase links and look at the link targets. When used, the link names are used for starname matching and the targets for type matching.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
specifies that any keyword that looks for branch entries do not chase links. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

-select_entry_type STR, -slet STR
selects entries of the types specified by STR, which is a comma-delimited list of file system entry types; for example, exists entry ** -slet ms,mbx. This control argument is recognized when the key is "entry." Use the list_entry_types command to obtain a list of valid entry type values.

LIST OF KEYS

argument
true if you specify any str_args, false otherwise.

branch
true if any branches--segments, multisegment files (MSFs) , or directories--with a pathname matching star_name exist, false otherwise.

component
true if any archive components with a pathname matching star_name exist, false otherwise. Both the archive segment name and the component name can be a star_name.

directory, dir
true if any directories with a pathname matching star_name exist, false otherwise.

entry
true if any entries--segments, directories, MSFs, links, data management (DM) files, or extended entries--with a pathname matching star_name exist, false otherwise.

file
true if any segments or MSFs with a pathname matching star_name exist, false otherwise.

link
true if any links with a pathname matching star_name exist, false otherwise.

master_directory, mdir
true if any master directories with a pathname matching star_name exist, false otherwise.

msf
true if any MSFs with a pathname matching star_name exist, false otherwise.

non_null_link, nonnull_link, nnlink
true if any links with a pathname matching star_name exist and point to an existing segment, directory, or MSF, false otherwise.

nonbranch
true if any links with a pathname matching star_name exist, false otherwise.

nondir
true if any segments, MSFs, or links with a pathname matching star_name exist, false otherwise.

nonfile
true if any links or directories with a pathname matching star_name exist, false otherwise.

nonlink
true if any directories, segments, or MSFs with a pathname matching star_name exist, false otherwise.

nonmaster_directory, nmdir
true if any directories that are not master directories with a pathname matching star_name exist, false otherwise.

nonmsf
true if any directories, segments, or links with a pathname matching star_name exist, false otherwise.

nonobject_file, nobfile
true if nonobject files with a pathname matching starname exist, false otherwise. Segments or MSFs you do not have at least r access to are ignored.

nonobject_msf, nobmsf
true if nonobject MSFs with a pathname matching starname exist, false otherwise. MSFs you do not have at least r access to are ignored.

nonobject_segment, nobseg
true if nonobject segments with a pathname matching star_name exist, false otherwise. Segments you do not have at least r access to are ignored.

exists

nonobject_segment, noseg
true if no executable object segments with a pathname matching start_name exist, false otherwise. Segments you do not have at least r access to are ignored.

nonsegment, nonseg
true if any links, directories, or MSFs with a pathname matching star_name exist, false otherwise.

nonzero_file, nzfile
true if any nonzero-length segments or MSFs with a pathname matching star_name exist, false otherwise.

nonzero_msf, nzmsf
true if any nonzero-length MSFs with a pathname matching star_name exist, false otherwise.

nonzero_segment, nzseg
true if any nonzero-length segments with pathname matching star_name exist, false otherwise.

null_link
true if any links with a pathname matching star_name exist and point to nonexistent entries, false otherwise.

object_file, obfile
true if object files with a pathname matching starname exist, false otherwise. Segments or MSFs you do not have at least r access to are ignored.

object_msf, obmsf
true if object MSFs with a pathname matching starname exist, false otherwise. MSFs you do not have at least r access to are ignored.

object_segment, obseg
true if object segments with a pathname matching star_name exist, false otherwise. Segments you do not have at least r access to are ignored.

segment, seg
true if any segments with a pathname matching star_name exist, false otherwise.

zero_segment, zseg
true if any zero-length segments with a pathname matching star_name exist, false otherwise.

Name: expand_cobol_source, ecs

SYNTAX AS A COMMAND

```
ecs oldpath {newpath} {-control_args}
```

FUNCTION

applies a transformation to a COBOL source program. The nature of the source transformation is defined by control arguments. If you give no control argument, a segment containing text of a standard format COBOL source program that possibly contains COPY and REPLACE statements is translated into an equivalent source program not containing these statements.

ARGUMENTS

oldpath
: is the pathname of the input segment. If it does not have a suffix of .cobol, one is assumed. The suffix .cobol, however, must be the last component of the name of the source segment.

newpath
: is the pathname of the output segment. If it does not have a suffix of .cobol, one is assumed. If you omit it, the translated segment is in the form of the first component with the suffix .ex.cobol.

CONTROL ARGUMENTS

-card
: deletes meaningless trailing blanks from a standard fixed-format COBOL source program in card-image format and ignores characters in the identification field (columns 73-80).

-expand, -exp
: translates a standard fixed-format COBOL source program that possibly contains COPY and REPLACE statements into an equivalent source program not containing these statements. (Default)

-format, -fmt
: translates a pseudofree-form COBOL source program into a standard fixed-format COBOL source program. All characters in the source program are left exactly as typed.

-lower_case -lc
: translates a pseudofree-form COBOL source program into a standard fixed-format COBOL source program. All characters except for those in alphanumeric literals are converted to lowercase.

expand_cobol_source

-no_expand, -no_exp
    does not translate COPY and REPLACE statements in a standard fixed-format COBOL source program.

-upper_case, -uc
    translates a pseudofree-form COBOL source program into a standard fixed-format COBOL source program. All characters except for those in alphanumeric literals are converted to uppercase.

NOTES

You can use -fmt, -lc, and -uc with -exp, but not with -card. Don't use them if the source program is already in standard fixed format.

The control argument -card causes a standard fixed-format COBOL source program in card image format to be translated into an equivalent standard fixed-format program. If a line is 80 characters long, the identification field is deleted before removing meaningless trailing blanks. You can use -card with -exp.

If the -fmt, -lc, or -uc control arguments are used, the expand_cobol_source command assumes that the input file is in free form (as would be typically typed in from a terminal) and attempts to reformat each line into the standard COBOL reference format described in the *Multics COBOL User's Guide* (AS43). Statements in a COBOL source program generally begin in area B (column 12 and beyond). However, certain entries must begin in area A (column 8 through 11). These are COBOL-defined division names, section names, paragraph names, level indicators, and certain level numbers, as well as user-defined section names and paragraph names. Additionally, certain characters have special meaning when appearing in the indicator area (column 7), such as the asterisk, slash, hyphen, and letter "d".

The expand_cobol_source command recognizes all COBOL-defined names that are required to appear in area A and reformats lines containing them to guarantee that they do so. User-defined section names are recognized by the appearance of the word "section" on the line while words beginning the line and followed immediately by a period are assumed to be user-defined paragraph names. Source lines containing either of these are reformatted similarly to lines containing COBOL-defined sections and paragraphs. Lines beginning with level numbers 01, 66, 77, 88 are reformatted to begin in area A (at column 8) as required in standard American National Standard (ANS) COBOL. Lines beginning with level numbers 02 through 49 are indented a number of spaces identical to the numeric value of the level number plus seven (e.g., 02 begins at column 9, 05 at column 12).

Certain characters force special interpretation when they begin a free form source line. The slash (/) and asterisk (*) when used in this way denote a comment line with or without page eject, respectively; the hyphen (-) denotes a continuation line. Such lines are reformatted so that these special characters appear in the indicator area followed by the rest of the line. Additionally, for continuation lines, the remainder of the line following the hyphen is shifted to begin in area B as COBOL prohibits use of area A in this case.

Debugging lines are probably of little interest for Multics COBOL users due to the powerful symbolic debugging facilities available on an interactive basis, but they can be specified in free form source by beginning the line with "d*". In rare instances, in which a user-defined section or paragraph name is specified in a way not contextually recognizable by the expand_cobol_source command, you can force reformatting beginning in area A by beginning the line with "a*" (or "da*" in the case of debugging lines).

All other source lines (i.e., those not beginning with special character(s) and not containing entries required to begin in area A) are reformatted by insertion of eleven blanks forcing commencement in area B. Any indentation already existing in the free form file is thereby maintained relative to column 12.

The expand_cobol_source command also converts all horizontal tab characters (ASCII code 011) not contained in nonnumeric literals to spaces. The number of spaces is determined by subtracting the position of the tab character on the source line modulo 10 from 10. In this way, you can input the source program using the tab character as a formatting tool, yet avoid the fact that this is not part of the standard COBOL character set.

The COBOL source program output is acceptable to any ANS compiler with regard to reference format. (Actually, Multics COBOL relaxes many of these format requirements. However, it is usually desirable to eliminate the warnings and observations issued when such ANS rules are violated.) For transportability purposes, the output file can be created entirely in uppercase or lowercase (with the contents of nonnumeric literals left as is) by use of the -upper_case and -lower_case control arguments. If neither is specified, the case of all words remains the same as in the input file. Notice, all COBOL-defined names and characters with special meaning are recognized regardless of case, i.e., they can be all in uppercase, all in lowercase, or in mixed case.

For those users wishing to keep source files in free form, identical function described above is available on a per use basis via the -format control argument of the cobol command. Refer to the description of the cobol command for further information.

Name: explain_doc, edoc

SYNTAX AS A COMMAND

```
edoc manual_name {-control_args}
```

FUNCTION

returns information about a specified Multics manual(s).

ARGUMENTS

manual_name
is the manual's name, a short name for the manual, or the manual's order number. The name or the short name can contain blank spaces; it need not be enclosed in quotation marks. Capitalizing letters is not necessary. Use iteration to get more than one manual (see "Examples" below).

CONTROL ARGUMENTS

-all, -a
prints all the sections of manual information.

-audience, -aud
describes the audience for which the manual is intended.

-database_pathname PATH, -dbpn PATH
specifies the pathname of the data base you want instead of the default one. Once you supply -database_pathname, the specified data base is used for all subsequent invocations of explain_doc during your process until you select another data base.

-description, -desc
returns a brief description of the manual's contents. (Default)

-new_features, -nf
lists all new features that have been added to the manual with the last update (revision or addendum).

-no_audience, -no_aud
does not describe the manual's intended audience. (Default)

-no_description, -no_desc
suppresses printing of the brief description of the manual's contents.

-no_new_features, -no_nf
does not list new features. (Default)

-no_request_loop, -nrql
does not enter the request loop.

-no_table_of_contents, -no_toc
does not print the manual's table of contents. (Default)

-output_file PATH, -of PATH
directs the output to a file instead of to your terminal.

-request_loop, -rql
enters a request loop after the sections specified by control arguments have been printed. (Default)

-table_of_contents, -toc
prints the manual's table of contents.

NOTES

When explain_doc cannot find a data base entry that matches the manual name supplied, it may, in some cases, find a partial match that enables it to identify that name as belonging to a particular group such as the FORTRAN manuals or the Administrator's manuals. In that case, the relevant set of manual names is listed, and you can then choose to see the information on one of those manuals or return to command level.

NOTES ON REQUESTS

When you have invoked explain_doc and the section has been displayed, you are prompted

More information?

You can respond with one of the following requests:

yes, y
? (lists available responses)
description, desc
audience, aud
table_of_contents, toc
new_features, nf
all, a
no, n, quit, q (quits the request loop and returns you to command level).

EXAMPLES

```
!  edoc ag92 -audience

Title:  Multics Commands and Active Functions
Order No.:  AG92-05A
Release Supported:  MR10.2

Audience:
   Programmers and nonprogrammers who use Multics commands and
   active functions.

More information?  ! no

r 13:51 6.782 183
```

When using iteration, enclose names containing blank spaces within quotes; for instance,

```
! edoc MAM ("registration and accounting" system project)
```

or

```
! edoc ("multics commands" subroutines)
```

Name: exponent_control

SYNTAX AS A COMMAND

```
exponent_control -control_args
```

FUNCTION

controls the behavior of the system in the event of a computational overflow or underflow.

CONTROL ARGUMENTS

-restart STRs, -rt STRs
: specifies that either overflow or underflow or both are to be automatically restarted with defined results. STRs can be either or both of the strings "overflow" or "underflow."

-fault STRs, -flt STRs
: specifies that either overflow or underflow or both are to cause the normal fault conditions.

-overflow_value STR, -ovfv STR
: specifies the value to be returned for an overflowing computation. If no value is given the largest possible floating point value is used.

-print, -pr
: prints the current behavior with respect to exponent errors and the current overflow value.

NOTES

By default Multics signals fault conditions on computational overflows and underflows. See the Programmer's Reference Manual for more information on faults and other unusual conditions.

This command only affects the system's handling of exponent overflow and underflow when the overflow condition or the underflow condition is raised. In certain cases, the error condition is raised instead. This command does not affect the system's handling of the cases in which the error condition is raised.

EXAMPLES

To restart on underflows:

```
! exponent_control -restart underflow
```

To signal a fault on overflows:

```
! exponent_control -fault overflow
```

To restart on both underflows and overflows:

```
! exponent_control -restart underflow overflow
```

Name: fast

SYNTAX AS A COMMAND

```
fast
```

FUNCTION

puts you into the FAST subsystem, a time-sharing facility designed primarily for creating and running BASIC and FORTRAN programs.

NOTES

For a description of the commands available under FAST, see the *Multics FAST Subsystem User's Guide* (AU25).

To exit the subsystem and return to Multics system command level, type quit (q).

Name: file_output, fo

SYNTAX AS A COMMAND

```
fo {path} {-control_args}
```

FUNCTION

directs I/O output switches to a specified file. The effects of this command can be stacked.

ARGUMENTS

path
is the pathname of a segment. If the segment does not exist, it is created. If you give no path, the segment output_file in your working directory is assumed.

CONTROL ARGUMENTS

-extend
extends the output file. (Default)

-source_switch STR, -ssw STR
specifies the name of an I/O switch to be redirected. (Default: user_output)

-truncate, -tc
truncates an existing output file for file_output. (Default: to extend the output file)

NOTES

Each command invocation of file_output stacks up another attachment for each of the specified switches.

See the revert_output, syn_output, and terminal_output commands.

EXAMPLES

The command line

```
! fo text.cpa;cpa text.old text.new;ro;dp text.cpa
```

makes a comparison of two text segments named 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.

This 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: files

SYNTAX AS A COMMAND

```
files star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[files star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of segments and multisegment files (MSFs) that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
    does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
    signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: floor

SYNTAX AS A COMMAND

```
floor num
```

SYNTAX AS AN ACTIVE FUNCTION

```
[floor num]
```

FUNCTION

returns the largest decimal integer less than, or equal to, its argument.

EXAMPLES

```
string [floor 4.7]
4.0    string [floor -4.7]
-5.0
```

Name: format_document, fdoc

SYNTAX AS A COMMAND

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

FUNCTION

formats text segments.

ARGUMENTS

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

CONTROL ARGUMENTS

-hyphenate {N}, -hph {N}
changes the default hyphenation mode from off to on. N is the length of the smallest separated word part; its default value is 2. (Default: off)

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

-no_hyphenate, -nhph
does not hyphenate words. (Default)

-output_file {PATH}, -of {PATH}
directs the output to a file instead of to your terminal. If you provide no PATH, 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". (Default: off)

-page_numbers, -pgno
ends each page with two blank lines and a centered page number. (Default: off)

LIST OF CONTROL LINES

The following is a summary of the control lines recognized by the command:

.alb
(align both) inserts 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 insert extra spaces into the lines. The left margin is even, the right ragged. This control line is effective only if fill (.fin) is also in effect.

.brf
(break format) finishes the current output line by formatting any pending texts as a short line.

.brf (break page) finishes the current page, formatting any pending texts as a short line.

.fif
(fill off) retains lines in the output file as they are in the input file no matter the length.

.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 .alb and .all above.) (Default)

.hy
hyphenates according to the default.

.hyf
sets the hyphenation to off. (Default)

.hyn {N}
sets hyphenation to on. (Default: 2)

.in {N}, .inl {N}
(indent, indent left margin) sets the indention level. If you give N a + or −, then N is added to or subtracted from the current indention level; without a sign, N becomes the indention level. An error message is displayed if an indention level is less than zero (default) or greater than the line length.

.pdl {N}
(page length) sets the page length. If you give N a + or − sign, then N is added to or subtracted from the current page length; without a sign, the page length is changed to N. The 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); if less than 13 or 14, an error message is displayed. (Default: 66)

.spf {N}
(space format) finishes the current output line and then adds N blank lines. (Default: 1)

.pdw {N}
(page width) sets the page width (line length). If you give N a + or − sign, then N is added to or subtracted from the current line length; without a sign, the line length is changed to N. An error message is displayed if the set line length does not accommodate the input file. (Default: 65)

`.un {N}, .unl {N}`
(undent, undent left margin) sets the indention level for the output of the next line only. If you give N a + sign or no sign, then indent N characters less than the current indention level; if you give N a −, then indent N characters more than the current indention level. An error message is displayed if the indention that is caused by undenting is less than zero or more than the line length.

NOTES

This command takes an input file that 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, place special control lines in the input file. All control lines begin with a period and must be on a line by themselves. This command makes two assumptions about how the document is to be formatted: it assumes that the output is to be on standard-sized paper with 66 lines per page and lines 65 characters wide (these values represent an 8 1/2 by 11 inch page with one-inch margins all around) and that both the left and right margins are justified.

Output lines are built by the embedded control lines within the input file being formatted; these input control lines do not appear in the output.

EXAMPLES

The following page shows an example of a business letter created using format_document. Suppose the letter is to be printed on a standard 8-1/2 by 11 inch piece of paper with lines 60 characters long. The input file is first created with a text editor. In this example the input file is labeled letter.fdocin. Line numbers are shown on the example to reference comments below.

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

Line 1 sets the line length (page width) to 60 characters.

Line 2 turns fill mode off so that the lines will appear just the way they are input. 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. The same thing is done at line 40 just prior to the closing.

Line 3 sets the indention to character position 35 so that the return address be on the right-hand side of the letter starting at character position 35.

Lines 4-6 is the return address.

Lines 7-9 are three blank lines inserted by pressing the newline (NL) or carriage return (CR) key three times.

Line 10 resets the indention level to 0 (the absence of a number after the control gives the default).

Lines 11-17 are the address of the recipient, two blank lines, the salutation, and another blank line.

Line 18 turns fill mode on.

Line 19: the indention level is set to 0 by the control in line 10, but you want to indent the next line by only five 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.

Lines 20-37 is the body of the letter. No attempt to control the entered line lengths has been made (free form). The fdoc command formats all the data for you, so long as fill mode is on. Lines can be as short or as long (wrap-around) as you wish.

Line 40 turns fill mode off.

Line 41 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 42-47).

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 very grateful.

                                        Sincerely yours,



                                        Michael P. Marley

     r 1154 0.149 25
```

To make a final copy of the letter, you must first center it on the paper. Most terminals and printers print a standard 8-1/2 by 11 inch piece of paper with 85 characters on a line, which means that the lines are 25 characters shorter than the width of the paper, so if each line begins at character position 12 (roughly half of 25) the letter will be centered on the page. The command line

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

accomplishes this.

If the letter is to be saved in a file so that it can be printed on another terminal or on a high-speed printer, use -output_file:

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

In this example, the file is named letter.fdocout.

If 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 (if the letter is more than one page). When the letter has been printed you can enter another carriage return and you are returned to Multics command level.

Name: format_line, fl

SYNTAX AS A COMMAND

```
fl control_string {args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[fl control_string {args}]
```

FUNCTION

returns a single, quoted character string that is formatted from an ioa_ control string and other optional arguments.

ARGUMENTS

control_string
is an ioa_ control string used to format the return value of the active function (see "Notes" below).

args
are character strings substituted in the formatted return value according to the control string.

NOTES

The following ioa_ control codes are allowed (see the ioa_ subroutine). The control string is output exactly as is except that certain constructs beginning with a caret (^) are expanded, which may involve argument substitution or interpretation. Unimplemented constructs are output as is, but avoid them to allow for future extensions.

CONTROL	ACCEPTABLE ARGUMENTS
^a ^Na	any character string
^d ^Nd	a character representation of a number, including optional exponent (315.44 or .2789e+2 or 1101b)
^i ^Ni	same as ^d
^f ^Nf	same as ^d
^.Df ^N.Df	same as ^d
^e ^Ne	same as ^d
^o ^No	same as ^d
^[...^	"true", "false", or an integer character string
...^]	any number of any character string
^(...^) or ^N(...^)	an integer character string.
^s ^Ns	

In addition, you can use any of the following carriage movement controls:

^N/	^N|	^N-
^Nx	^N^	^R
^B	^t	^Nt
^N.Dt	^/	^|
^_	^x	^^

where N is an integer count or a "v". When you give "v", an integer character string from the args is used for count.

If you don't give optional arguments, the value returned depends on the ioa_ control string that you specified.

EXAMPLES

In an exec_com segment, the lines

```
&if [query [fl "^a copies already exist.^/Build another?"] &2]
&then ec build_new_data [plus 1 &2]
```

might be expanded when &2 is 3 to

```
3 copies already exist.
Build another?
```

The lines

```
! string [fl "Insurance option:  ^[no fault^;regular^]"
    [query "No Fault?"]]
```

print the following if you answer "no" to the query

```
Insurance option:  regular
```

Name: format_line_nnl, flnnl

SYNTAX AS A COMMAND

```
flnnl control_string {args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[flnnl control_string {args}]
```

FUNCTION

is the command/active function interface to the ioa_$nnl subroutine. When used as a command, the formatted string is printed without a trailing newline.

ARGUMENTS

control_string
: is an ioa_ control string used to format the return value of the active function.

args
: are character strings substituted in the formatted return value according to the control string.

NOTES

The following ioa_ control codes are allowed (see the ioa_ subroutine). The control string is output exactly as is except that certain constructs beginning with a caret (^) are expanded, which may involve argument substitution or interpretation. Unimplemented constructs are output as is, but avoid them to allow for future extensions.

CONTROL	ACCEPTABLE ARGUMENTS
^a ^Na	any character string
^d ^Nd	a character representation of a number, including optional exponent (315.44 or .2789e+2 or 1101b)
^i ^Ni	same as ^d
^f ^Nf	same as ^d
^.Df ^N.Df	same as ^d
^e ^Ne	same as ^d
^o ^No	same as ^d
^[...^	"true", "false", or an integer character string
...^]	any number of any character string
^(...^) or ^N(...^)	an integer character string.
^s ^Ns	

In addition, you can use any of the following carriage movement controls:

^N/	^N|	^N-
^Nx	^N^	^R
^B	^t	^Nt
^N.Dt	^/	^|
^-	^x	^^

where N is an integer count or a "v". When you give "v", an integer character string from the args is used for count.

If you don't give optional arguments, the value returned depends on the ioa_ control string that you specified.

Name: format_pl1, fp

SYNTAX AS A COMMAND

```
fp in_path {-control_args}
```

FUNCTION

formats a PL/I, create_data_segment, reductions, or pl1_macro source segment to make it more readable.

ARGUMENTS

in_path
is the pathname of the source segment. Suffixes for PL/I, create_data_segment, reductions, and pl1_macro are recognized. If in_path does not have a recognized suffix, format_pl1 attempts to use in_path.pl1 or in_path.cds, in that order.

CONTROL ARGUMENTS

-brief, -bf
does not print a warning if the prevailing style differs from the command line modes.

-check_comments, -ckcom
prints a warning if a comment contains "/*". It is useful if you have omitted a "*/" from a comment.

-check_strings, -ckstr
prints a warning if a character string constant contains "/*", "*/", or vertical white space. It is useful after receiving an error message indicating that you have omitted a quote from a character string constant.

-force, -fc
forms the prevailing style by concatenating the default style, the prevailing style control comment modes, and the command line modes. If the program already has a prevailing style control comment and the prevailing style differs from the prevailing style control comment modes, the prevailing style control comment is deleted. If you provide -record_style, a new prevailing style control comment is inserted.

-long, -lg
prints a warning indicating the modes that differ if the prevailing style differs from the command line modes. (Default)

-modes STR, -mode STR, -md STR
specifies a modes string STR used in forming the prevailing style (see "Notes on Prevailing Style" below).

-no_check_comments, -nckcom
does not print a warning if a comment contains "/*". (Default)

-no_check_strings, -nckstr
does not print a warning if a character string constant contains "/*", "*/", or vertical white space. (Default)

-no_force, -nfc
forms the prevailing style by concatenating the default style, the command line modes, and the prevailing style control comment modes. (Default)

-no_record_style, -nrcst
does not put a control comment in the source specifying the prevailing style. (Default)

-no_require_style_comment, -nrqst
formats the source even if it does not already contain a prevailing style control comment. (Default)

-no_version, -nver
does not print the version of format_pl1. (Default)

-output_file path, -of path
specifies the pathname of the formatted source segment. The equal convention is allowed. The suffix of in_path is assumed if not specified. If omitted and there were no errors, in_path is overwritten; if omitted and there were errors, a formatted copy is left in the process directory.

-record_style, -rcst
puts a control comment in the source specifying the prevailing style if the source does not already have a prevailing style control comment. The comment is placed immediately before the first token of the program so it does not interfere with copyright notices.

-require_style_comment, -rqst
prints an error message if the source does not already contain a prevailing style control comment. This is useful if you are concerned with accidentally destroying the style of a hand-formatted program.

-version, -ver
prints the version of format_pl1.

NOTES

Alternate methods of formatting particular language constructs are selected by means of modes. Several popular styles, consisting of groups of modes, are defined. Modes and styles are specified on the command line and in comments in the source segment.

An exec_com tool called compare_pl1 compares PL/I source segments of dissimilar formats via format_pl1.

If you give two opposing control arguments, the rightmost one is chosen. The term "token" excludes comments. See the *Multics PL/I Language Specification* (AG94) for definitions of words describing syntactic constructs in a PL/I program, e.g., independent statement, declaration list, etc. Condition and label prefixes are not considered part of a statement.

NOTES ON MODES STRING

A modes string changes the style format_pl1 uses to format a program. It consists of mode names separated by commas. Many modes can be preceded by ^ to turn the specified mode off. 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. Modes not specified by the modes string are left unchanged.

NOTES ON CONTROL COMMENTS

A control comment has the form "/* format: STR */", where STR is a modes string. The control comment can start with one of the other comment indicators listed in "List of Comment Indicators" below. Control comments can occur only before the first token of the program, between a semicolon and the first token of the next statement, or after the last semicolon in the program. Control comments cannot occur in the middle of a statement. Optional horizontal white space can precede "format" or surround STR. Some modes changed by a control comment may not take effect immediately; for example, end statements are formatted according to the modes in effect when the matching do, begin, or procedure statement was formatted.

There are two special control comments that are used in if statements. If a comment containing "/* case */" or "/* tree */" immediately follows the word "if" in an if statement, then the current style is changed for the duration of that if statement; exactly one space must precede and follow "case" and "tree". (See case and tree modes in the "List of if Statement Modes" below.)

NOTES ON PREVAILING STYLE

The style in which format_pl1 formats a PL/I program is formed from a combination of three sources: format_pl1's default style, modes specified on the command line, and control comments in the program. The first control comment of the program preceding the first token of the program is called the prevailing style control comment. A program might not have a prevailing style control comment. The style specified by the concatenation of the default style, the command line modes, and the prevailing style control comment modes is called the prevailing style. This is the style in which most of the program is formatted.

Since a styleN mode specifies the setting of every mode, if the prevailing style control comment contains a styleN mode, the default format_pl1 style and the command line modes are ignored. If the program doesn't already have a prevailing style control comment, the command line

```
! format_pl1 in_path -modes MY_STYLE -record_style
```

formats a program in MY_STYLE and records the style in a prevailing style control comment. If the program has a prevailing style control comment, the program is formatted in the style specified by its prevailing style control comment and -record_style has no effect. The prevailing style control comment created as a result of -record_style always begins with a styleN mode.

NOTES ON EXAMPLES

The examples show how various program fragments are formatted. If you give no control comment, then style1 (default), is assumed; if you give a control comment, the default is used for all unspecified modes. Unless you use delnl,insnl mode, each line of the input source segment contains the same tokens as the corresponding line of the example. If you use delnl,insnl mode, then newline characters are inserted and deleted as required by the style. (See "Notes on Styles" below.)

LIST OF MODES

Modes for various language constructs are listed separately.

`styleN`
specifies formatting style N.

`revert`
changes the formatting style to the prevailing style. You can't supply this mode in -modes or in the prevailing style control comment. Note that the on mode is changed to the phase specified in the prevailing style.

`off, ^on`
leaves the source exactly as it is until a control comment changes the style to on. In this mode, block and group entries and exits are noticed so the program following the on mode control comment is formatted correctly.

`on, ^off`
starts formatting the source again. (Default)

`indN`
N is the number of columns to indent for each block or group indentation level. An independent statement in a then or else clause that does not have a condition or label prefix is indented a minimum of five columns when not in ifthen mode even if indN is less than five. This avoids placing the then or else clause on the line after the "then" or "else". The five columns are measured from the column the "else" would start in if the else clause was an independent statement. (Default: 5)

```
Example:  /* format: ind3 */
          if v = 2
          then
             do;
                x = 12;
                y = 128;
             end;
          else z = 12;
```

`lIN`
N is the output line length. (Default: 122)

`lineconindN`
N is the number of columns to indent the continuation of lines that exceed the output line length. (Default: 5)

`equalindN`
if N is greater than zero, places the equal sign of assignment statements in a column indented N columns from the left margin. If in insnl mode, a new line is inserted if necessary to insure that the equal sign is indented the specified column positions. If N is zero, then assignment statements are formatted just as ordinary statements. (Default: 0)

```
Example:  /* format: style2,equalind8 */
          a          = 12;
          index      = 1;
          second_index
                     = 43;
```

`initcolN`
N is the initial column that statements occurring before the first procedure statement should start at. This is useful for include files. (Default: 6)

NOTES ON DECLARE STATEMENTS

Depending upon delnl and insnl modes, each level one identifier that is declared is placed on a line by itself. In indattr mode, all attributes are indented to the same column. If a declaration list (parenthesized list) has all the attributes factored (i.e., each declaration component in the declaration list consists only of an identifier) and doesn't contain any comments and none of the identifiers contain a $, then the declaration list is placed on as few lines as possible instead of placing each identifier on a separate line.

```
Example:  dcl (hbound, index, null) builtin;
```

LIST OF DECLARE STATEMENT MODES

`indattr`
always indents the attributes so they start in the same column. (Default)

^indattr
doesn't indent the attributes from the identifier being declared.

inddcls
indents declare statements so they start in the same column any other statement would start in. (Default)

^inddcls
always starts declare statements in column 1.

declareindN
indents N columns after the start of dcl. (Default: 8)

dclindN
indents N columns after the start of dcl. (Default: 8)

idindN
indents N columns after the start of an identifier before starting the attributes. Ignored if in ^indattr mode. (Default: 23)

struclvlindN
indents N columns for each level in a structure. (Default: 2)

LIST OF IF STATEMENT MODES

ifthenstmt
puts the then clause on the same line as the "if", if it fits. These criteria must be met: (1) the then clause must be an independent statement and cannot be another if statement; (2) the then clause must not have a condition or label prefix; (3) if in tree mode, the if statement must not have an else clause; (4) if in case mode, the if statement must fall into one of the following categories: (a) there is no else clause, (b) the else clause consists of an if statement, or (c) the if statement under consideration is an else clause of another if statement.

```
Example:  /* format: ifthenstmt */
          if x > 3 then return;
```

^ifthenstmt
doesn't put the then clause on the same line as the "if". (Default)

```
Example:  if x > 3
          then return;
```

ifthendo
puts the "then do" on the same line as the "if", if it fits even if indN is less than five, if the then clause of an if statement is a noniterative do group without a condition or label prefix; puts the "then do" on the same line if it fits and in ^thendo mode; lines the "then" up with the "if" if in thendo mode; puts the "else do" on the same line, if it fits even if indN is less than five, if the else clause of an if statement is a noniterative do group without a condition or label

prefix. In ^delnl mode, the "then" or the "else" must already be on the same line as the "do".

Example:
```
/* format: ifthendo,^indnoniterdo */
if v = 2 then do;
     x = 8;
     y = 9;
end;
else do;
     x = 9;
     y = 92;
end;

/* format: ind3,ifthendo,^indnoniterdo */
if v = 2 then do;
   x = 8;
   y = 9;
end;
else do;
   x = 9;
   y = 92;
end;
```

^ifthendo
doesn't put the "then do" on the same line as the "if". (Default)

Example:
```
/* format: ^indnoniterdo */
if v = 2
then do;
     x = 8;
     y = 9;
end;
else do;
     x = 9;
     y = 92;
end;
```

thendo
if in ifthendo mode, lines the "then" up with the "if".

Example:
```
/* format: ind3,ifthendo,thendo,^indnoniterdo */
if v = 2
then do;
   x = 8;
   y = 9;
end;
else do;
   x = 9;
   y = 92;
end;
```

^thendo
if in ifthendo mode, puts the "then do" on the same line as the "if" if it fits. (Default)

```
Example: /* format: ind3,ifthendo,^indnoniterdo */
         if v = 2 then do;
            x = 8;
            y = 9;
         end;
         else do;
            x = 9;
            y = 92;
         end;
```

ifthen
puts the "then" on the same line as the "if".

```
Example:  /* format: ind3,ifthen */
          if v = 2 then
             do;
                x = 12;
                y = 128;
             end;
          else
             do;
                x = 128;
                y = 12;
             end;
```

^ifthen
lines the "then" up with the "if". (Default)

```
Example:  if v = 2
          then x = 8;
          else x = 9;
```

elsestmt
places independent statements on the same line as the else clause; places nonindependent statements on the same line as the else clause if indN is set to five or more. (Default)

^elsestmt
does not place any statements on the same line the else clause appears on unless the else clause is part of an if statement in "case" mode and the statement following the else clause is another if statement.

```
Example:  /* format: style2,ifthen,^elsestmt */
          if a = b then
               x = y;
          else
               x = z;
```

indnoniterdo

indents the statements of the do group two indentation levels from the column in which the "if" starts if a then or else clause contains a noniterative do group without a condition or label prefix; indents three levels if in indthenelse mode. (Default)

```
Example:  if v = 2
          then do;
                    x = 3;
                    y = 4;
               end;
          else do;
                    x = 35;
                    y = 27;
               end;
```

^indnoniterdo

indents the statements of the do group one indentation level from the column in which the "if" starts if a then or else clause contains a noniterative do group without a condition or label prefix: indents two levels if in indthenelse mode.

```
Example:  /* format: ^indnoniterdo */
          if v = 2
          then do;
               x = 3;
               y = 4;
          end;
          else do;
               x = 35;
               y = 27;
          end;
```

inditerdo

indents the statements of the do group two indentation levels from the column in which the "if" starts if a then or else clause contains an iterative do group and it does not have a condition or label prefix; indents three levels if in indthenelse mode. (Default)

```
Example:  if v = 2
          then do i = 1 to 4;
                    a (i) = i;
               end;
```

^inditerdo
indents the statements of the do group one indentation level from the column in which the "if" starts if a then or else clause contains an iterative do group and it does not have a condition or label prefix; indents two levels if in indthenelse mode.

```
Example:  /* format: ^insnl,^inditerdo */
          if v = 2 then do i = 1 to 4;
               a (i) = i;
          end;
```

indnoniterend
starts the end statement of the noniterative do group in the same column as the statements of the noniterative do group if a then or else clause contains a noniterative do group without a condition or label prefix.

```
Example:  /* format: ^indnoniterdo,indnoniterend */
          if v = 2
          then do;
               x = 8;
               y = 9;
               end;
          else do;
               x = 9;
               y = 92;
               end;
```

^indnoniterend
starts the end statement of the noniterative do group in the column that is one indentation level before the column the statements of the noniterative do group start in if a then or else clause contains a noniterative do group without a condition or label prefix. (Default)

```
Example:  /* format: ^indnoniterdo */
          if v = 2
          then do;
               x = 8;
               y = 9;
          end;
          else do;
               x = 9;
               y = 92;
          end;
```

indthenelse
indents the then and else clauses two indentation levels from the column in which the "if" is placed. Places the "else" one indentation level from the column in which the "if" is started. If in ^ifthen mode and the ifthenstmt and ifthendo modes do not apply to the if statement, places the "then" in the same column as the "else". If in case mode and the if statement under consideration is the else

clause of another if statement, then indents from the column in which the preceding "else" is placed instead of the column in which the "if" is placed. In case mode this mode is ignored for the else clause if the else clause consists of an if statement or the if statement under consideration is an else clause of another if statement.

Example:
```
/* format: indthenelse */
if v = 2
     then x = 8;
     else do;
               x = 9;
               call default;
          end;

/* format: indthenelse */
if v = 2
     then x = 8;
else if v = 3
     then x = 25;
else call error;
```

^indthenelse

indents the then and else clauses one indentation level from the column in which the "if" is placed. Places the "else" in the same column as the "if" is placed. If in ^ifthen mode and the ifthenstmt and ifthendo modes do not apply to the if statement. places the "then" in the same column as the "else". If in case mode and the if statement under consideration is the else clause of another if statement, then indent from the column in which the preceding "else" is placed instead of the column in which the "if" is placed. (Default)

Example:
```
if v = 2
then x = 8;
else do;
          x = 9;
          call default;
     end;

if v = 2
then x = 8;
else if v = 3
then x = 25;
else call error;
```

case, ^tree

indents "else if" clauses like a case statement. (Default)

```
Example:  if char = "a"
          then call char_a;
          else if char = "b"
          then call char_b;
          else if char = "c"
          then call char_c;
          else call error;

          /* format: ifthenstmt */
          if char = "a" then call char_a;
          else if char = "b" then call char_b;
          else if char = "c" then call char_c;
          else call error;
          /* Decision tree formatted like a case statement. */
          if condition_1
          then if condition_2
               then call condition (0);
               else call condition (1);
          else if condition_2
          then call condition (2);
          else call condition (3);
```

tree, ^case
indents "else if" clauses like a decision tree.

```
Example:  if /* tree */ condition_1
          then if condition_2
               then call condition (0);
               else call condition (1);
          else if condition_2
               then call condition (2);
               else call condition (3);

          /* Case statement formatted like a decision tree. */
          /* format: tree */
          if char = "a"
          then call char_a;
          else if char = "b"
               then call char_b;
               else if char = "c"
                    then call char_c;
                    else call error;
```

indbegin
indents the body of those begin blocks that do not follow then clauses, else clauses, or on statements. (Default)

```
Example:  a = 15;
          begin;
               x = 21;
          end;
          y = 2;
```

^indbegin
does not indent the body of those begin blocks that do not follow then clauses, else clauses, or on statements.

```
Example:  /* format: ^indbegin */
          a = 15;
          begin;
          x = 21;
          end;
          y = 2;
```

indbeginend
indents the end statement of those begin blocks that do not follow then clauses, else clauses, or on statements to the level of the begin block body. If in ^indbegin mode, this option does not effect the placement of the end statement. The end statement always lines up under the begin block body and in the same column as the begin statement since they are the same column.

```
Example:  /* format: indbeginend */
          a = 15;
          begin;
               x = 21;
               end;
          y = 2;

Example:  /* format: ^indbegin,indbeginend */
          a = 15;
          begin;
          x = 21;
          end;
          y = 2;
```

^indbeginend
places the end statement of those begin blocks that do not follow then clauses, else clauses, or on statements in the same column as the begin statement. (Default)

indthenbegin
indents the body of begin blocks that follow then clauses, else clauses, or on statements. (Default)

```
Example:  if a = 2
          then begin;
                    x = 12;
               end;
          y = 15;
```

^indthenbegin
does not indent the body of begin blocks that follow then clauses, else clauses, or on statements. By default the corresponding end statement is indented one less level than the begin block body (see indthenbeginend).

```
Example:  /* format: ^indthenbegin */
          if a = 2
          then begin;
               x = 12;
          end;
          y = 15;
```

indthenbeginend
indents the end statement of those begin blocks that follow then clauses, else clauses, or on statements to the same level as the begin block body.

```
Example:  /* format: indthenbeginend */
          if a = 2
          then begin;
                    x = 47;
                    end;
          y = 100;
```

```
Example:  /* format: ^indthenbegin,indthenbeginend */
          if a = 89
          then begin;
               x = y;
               end;
          y = 100;
```

^indthenbeginend
indents the end statement of those begin blocks that follow then clauses, else clauses, or on statements one less level than the indentation of the begin block body. (Default)

```
Example:  if a = b
          then begin;
                    x = 15;
               end;
          y = 9;
```

```
Example:  /* format: ^indthenbegin */
          if a = b
          then begin;
               x = 15;
          end;
          y = 9;
```

indproc
starts the procedure statement in the same column as other statements of the containing procedure if a procedure is contained within at least two other procedures, i.e., the procedure is an internal procedure of an internal procedure; starts the procedure statement in column indN+1, otherwise.

```
Example:  /* format: indproc */
          extp:
               procedure;
                    a = 1;
          intp1:
               procedure;
                    b = 2;
          intp2:
                    procedure;
                         c = 3;
                    end intp2;
               end intp1;
               end extp;
```

^indproc
start procedure statements in column indN+1. (Default)

```
Example:  extp:
               procedure;
                    a = 1;
          intp1:
               procedure;
                    b = 2;
          intp2:
               procedure;
                    c = 3;
               end intp2;
               end intp1;
               end extp;
```

indprocbody
indents the body of the procedure, the statements following the procedure statement, one level farther than the indentation of the procedure statement. (Default)

^indprocbody
begins the statements following a procedure statement in the same column as the procedure statement.

```
Example:  /* format: ^indprocbody */
          test:
               procedure;
               a = 12;
               end test;
```

indend
starts the end statement of a do group, except those affected by indnoniterend mode, in the same column as the statements of the group.

```
Example:  /* format: indend */
          do i = 1 to 5;
               a (i) = i;
               end;
```

^indend
starts the end statement of a do group, except those affected by indnoniterend mode, in the column that is one indentation level before the column the statements of the group start in. (Default)

```
Example:  do i = 1 to 5;
               a (i) = i;
          end;
```

NOTES ON HORIZONTAL WHITE SPACE

All horizontal white space, except within character string constants and comments, is removed from the program. Spaces are inserted before left parentheses, after commas, around operators, and in other places to improve readability. Where possible, horizontal tabs are used to conserve space in the output segment.

Statements continued onto another line are indented lineconindN from the current left margin. The left margin at which a statement is indented is increased by indN for every nested begin block, group, and then or else clause except as required by the indnoniterdo, initerdo, indthenelse, case, indproc, and indprocbody modes. Entry statements are placed in the same column as the corresponding procedure statement. The left margin before a procedure statement is saved; it is restored after the procedure's end statement. After a procedure statement the left margin is increased by indN if the indprocbody option is set. End statements are started in the same column as the statement that began the block or group except as required by the indnoniterend, indend, indbeginend, and indthenbeginend modes. Condition and label prefixes are placed on lines by themselves except possibly in ^insnl mode.

NOTES ON VERTICAL WHITE SPACE

Vertical white space within character string constants and comments is never changed. Other vertical white space can be intrastatement and interstatement. In on mode, vertical tabs and newlines before newpages are removed, newlines before vertical tabs are removed, and multiple newpages are reduced to one. Then a newline is inserted before and after a sequence of vertical tabs and newpages if there is not already one there. Interstatement vertical white space is never changed except for the above canonicalizations; intrastatement vertical white space is also canonicalized as above and processed depending upon delnl and insnl modes.

LIST OF VERTICAL WHITE SPACE MODES

`delnl`
deletes all existing intrastatement vertical white space.

`^delnl`
leaves existing intrastatement vertical white space in the program. (Default)

`insnl`
inserts newlines in the program if necessary. Newlines are inserted when statements are too long to fit on a line. To determine where newlines are inserted, various heuristics exist that use the statement type and the precedence of the tokens in the statement to determine where to insert newlines. The driving force of format_pl1 is what column statements or other language constructs should start in. Newlines are inserted to start a statement, subset of a statement, or a comment in a particular column.

`^insnl`
doesn't insert newlines into the program. (Default)

NOTES ON COMMENTS

Comments are classified by where they occur within a PL/I program and where they are placed in the output segment. They are divided into three categories: intrastatement comments, indented comments, and block comments. Intrastatement comments occur between the first token of a statement and the semicolon ending the statement. They are normally separated from surrounding tokens by a space except as required for linecom mode. Comments that follow a semicolon and are separated by at most one newline character are considered indented comments. They are placed in column comcolN. All other comments are block comments. Block comments are placed in column one or indented to the current left margin according to the indcom and indblkcom modes. All comments following a blank line, following a block comment, and before the first token of the program are block comments. Placing a comment in column N means that the "/*" starts in column N.

In these special cases intrastatement comments are treated as indented comments and placed in column comcolN: comments following a comma; preceding the right parenthesis of a declaration component; following the colon in a condition or label prefix; and, in if statements that the ifthenstmt or ifthendo modes do not apply to, following the "then" in ifthen mode and preceding the "then" in ^ifthen mode.

A comment indicator is placed at the beginning of a comment to specify where the comment is placed or to inhibit indcomtxt mode on the comment. If necessary, even in ^insnl mode, a newline is inserted to place the comment in the specified column. Comment indicators consist of the "/*", which starts the comment followed by other characters as listed below. A comment indicator does not affect the classification of succeeding comments.

LIST OF COMMENT INDICATORS

/*
places the comment according to its default classification.

/**
places the comment in column comcolN.

/***
indents the comment relative to the current left margin according to indblkcom mode.

/****
places the comment in column one.

```
Example:  /* format: ind3,comcol21 */
          /**** column one comment */
                /*** comment indented to left margin */
                              /** indented comment */
                a = 3;        /* indented comment */

          /* column one comment by default */
                b = 4;
```

/*^
places the comment according to its default classification; formats the comment according to ^indcomtxt mode.

/**^
places the comment in column comcolN; formats the comment according to ^indcomtxt mode.

/***^
indents the comment relative to the current left margin according to indblkcom mode; formats the comment according to ^indcomtxt mode.

`/****^`
places the comment in column one; formats the comment according to ^indcomtxt mode.

LIST OF COMMENT MODES

`comcolN`
N is the column indented comments are placed at. (Default: 61)

`indcom`
indents block comments relative to the current left margin according to indblkcom mode. This mode doesn't apply before the first token of the program so it doesn't interfere with copyright notices.

```
Example:  /**** format: ind3 */
                /* comment indented to left margin */
                a = 3;
```

`^indcom`
places block comment in column one. (Default)

```
Example:  /* format: ind3 */
          /* column one comment */
                a = 3;
```

`indblkcom`
starts comments that begin with the "/***" or "/***^" comment indicators and in indcom mode, block comments, at the current left margin. (Default)

```
Example:  /* format: ind3,indcom */
          /**** column one comment */
                /*** block comment */
                a = 5;
```

`^indblkcom`
starts comments that begin with the "/***" or "/***^" comment indicators and in indcom mode, block comments, one indentation level before the current left margin.

```
Example:  /* format: ind3,indcom,^indblkcom */
          /**** column one comment */
             /*** block comment */
                a = 5;
```

`linecom`
intrastatement comments at the end of a line in the original source segment apply to an entire line. These comments are treated as indented comments and are placed in column comcolN.

```
Example:  /* format: linecom */
          if line_status < 3            /* Is line active? */
               | char_count > 0
          then return;
```

^linecom
intrastatement comments apply to the preceding token. (Default)

```
Example:  /* format: ^delnl */
          if char = "040"b3 /* space */
               | char_count > 0
          then return;
```

indcomtxt
inserts a space if there is no horizontal or vertical white space between the comment indicator and the comment text or between the end of the comment text and the "*/" or the reverse of any comment indicator. Indents the text of continuation lines of a multiline comment so they line up; indenting the text of continuation lines does not apply to intrastatement comments. The horizontal white space between the comment indicator and the comment text on the first line of a comment is not reduced; however, leading horizontal white space on subsequent lines is replaced by sufficient horizontal white space to indent the line. If the comment is placed in column N, the length of the comment indicator is L, then the text of each line of the comment begins in column N+L+1. This mode does not apply to comments whose comment indicator ends with "^". Example:

```
Input:    /* format: indcomtxt */
          a = 3; /* Here we have a very
complicated assignment
statement. */

Output:   /* format: indcomtxt */
          a = 3;              /* Here we have a very
                                 complicated assignment
                                 statement. */
```

^indcomtxt
leaves the white space at the beginning of each line of a comment alone. The character string between the "/*" and the "*/" of a comment is never changed in this mode. (Default)

NOTES ON IRREVERSIBLE CHANGES

Several modes can cause irreversible changes to be made to the source program. Suppose that program p.pl1 was formatted with style S and does not contain a prevailing style control comment, then style T causes an irreversible change if the following command lines produce a program q.pl1 that differs substantially from p.pl1:

```
!  format_pl1 p -modes T -output_file q.pl1
!  format_pl1 q -modes S
```

The following modes can cause irreversible changes: delnl, insnl, ^linecom, and indcomtxt. If a program is not formatted with format_pl1, the on mode may also cause irreversible changes.

NOTES ON STYLES

```
style1:   on,ind5,ll122,initcol6,indattr,inddcls,declareind8,
          dclind8,idind23,struclvlind2,^ifthenstmt,^ifthendo,
          ^thendo,^ifthen,indnoniterdo,inditerdo,
          ^indnoniterend,^indthenelse,case,^indproc,^indend,
          ^delnl,^insnl,comcol61,^indcom,indblkcom,^linecom,
          ^indcomtxt  (Default)

style2:   style1,delnl,insnl

style3:   style2,^inddcls,declareindl0,dclindl0,idind20

style4:   style1,^indattr,^inddcls,declareind9,dclind5,
          ifthendo,^indnoniterdo,^inditerdo,indproc,linecom,
          indcomtxt

style5:   style2,linecom,ifthen,^indnoniterdo,indnoniterend,
          indcomtxt,^indthenbegin,indthenbeginend,^indprocbody,
          ^elsestmt,ind8,ll80,initcol0,idind24,comcol57,lineconind4,
```

Style1 indents declare statements, the attributes of declare statements, and the statements of the do group and lines up the end statement of a noniterative do group in a then or else clause under the "do". No irreversible changes, such as with the delnl, insnl, or indcomtxt modes, are made. Example:

```
/* format: style1 */
          declare entryname                char (32);
          if x = 2
          then do;
                    a = 43;
                    b = 21;
               end;
```

Style2 is the same as style1 except it uses the delnl,insnl modes. It may cause irreversible changes.

Style3 is the same as style2 except that declare statements start in column one and the identifiers and attributes start in columns aligned on tab stops. It may can cause irreversible changes. Example:

```
/* format: style3 */
declare   entryname           char (32);
          if x = 2
          then do;
                    a = 43;
                    b = 21;
               end;
```

Style4 starts declare statements in column one, doesn't indent the attributes in declare statements, formats noniterative do groups in then or else clauses by indenting the statements of the noniterative do group one indentation level from the "if" or the "else", and starts the end statement in the same column as the "if" or the "else". This style uses the ^delnl,^insnl modes, but still may cause irreversible changes from indcomtxt mode; it resembles that of the indent command. Example:

```
/* format: style4 */
declare  entryname char (32);
          if x = 2 then do;
               a = 43;
               b = 21;
          end;
```

Style5 starts declare statements in column one, sets the line length to 80 columns, sets the indentation amount to eight columns, sets the line continuation indentation to four columns, places the then clause of the "if/then" statement on the same line with the if, doesn't place any statement on a line with an if statement or else clause except for placing an if statement on the line after an else when in case mode, doesn't indent the procedure body, and doesn't indent the body of "begin/end" blocks or noniterative do groups under if statements.

Example:

```
/* format: style5 */
test:
        procedure;
        a = 12;
        if a = b then
                a = 15;
        else
                a = -15;
        b = 78;
        if b = c then
                do;
                x = 1;
                y = 99;
                end;
        a = 0;
        end test;
```

NOTES ON ERROR CHECKING

Parenthesis balance checking is done for statements that are not partially contained in include files. A warning is printed if an end statement with a closure label terminates more than one block or group. If you provide no -output_file and there were errors, the source segment is not overwritten and a formatted copy is left in the process directory. An error message is printed if a control comment is incorrect.

NOTES ON ERROR SEVERITIES

The following severity values are returned by the severity active function when the "format_pl1" keyword is used:

Value	*Meaning*
0	No error
1	Warning
2	Correctable error
3	Fatal error
4	Unrecoverable error
5	Bad control arguments, could not find source, or other severe errors.

NOTES ON MACROS

This command makes certain assumptions about macros. Include files must contain complete statements and balanced blocks or groups. Macro constructs can only occur between statements; they must not occur within a statement. All macro constructs are placed in column one. All %then and %else clauses of a %if macro must have the same effect on the current left margin and on block and group nesting; in addition, the effect on block and group nesting can only be to leave it unchanged or to start new blocks or groups. Blocks or groups that were started before the %if macro cannot be closed in the %then or %else clauses. A %then or %else clause cannot start with an else clause.

Name: **format_string, fstr**

SYNTAX AS A COMMAND

```
fstr {-control_args} text
```

SYNTAX AS AN ACTIVE FUNCTION

```
[fstr {-control_args} text]
```

FUNCTION

uses format_document_$string to fill and optionally adjust a text string. As a command, it prints the filled and adjusted text; as an active function, it requotes the return value.

ARGUMENTS

text
is the text strings to be filled. If you give more than one, they are concatenated, separated by a space. All arguments that do not begin with a minus and are not an operand of a preceding control argument are treated as text strings, and so are all arguments following the first text string encountered, whether or not they begin with a minus.

CONTROL ARGUMENTS

-adjust, -adj
left- and right-justifies text.

-break_word
to prevent line length from being exceeded, arbitrarily breaks into two or more parts words that are longer than the current line length and that cannot be hyphenated (when you use -hyphenate). (Default)

-column N, -col N
formats text as if N-1 characters already appeared on the first line. The output begins at column N. (Default: 1)

-hyphenate {N}, -hph {N}
changes the default hyphenation mode from OFF to ON. The optional parameter N is the length of the smallest separated word part; its default value is 2.

-indent N, -ind N
indents output N spaces from the left margin. (Default: 0)

-line_length N, -ll N
sets the line length to N characters. (Default: 65)

-no_adjust, -nadj
turns off justification. (Default)

-no_break_word
does not brake words longer than the line length; instead, it returns a line longer than the given line length.

-no_hyphenate, -nhph
turns off hyphenation. (Default)

-string text, -str text
    treats all remaining arguments as part of the text to be formatted.

-undent N, -und N
    undents the first line by N characters. If N is negative, the first line is indented (with respect to the -indent value) by N characters.

EXAMPLES

```
ioa_ "Subject:^-^a" [fstr -in 10 -col 11 -ll 20 lengthy subject to
   be filled]

Subject:  lengthy
          subject to
          be filled
```

fills the subject to lines 20 characters long, indented by 10 characters. The first output line begins in column 11, and so is not indented.

The command line

```
fstr -ll 23 -in 10 -un -3 -adj Now is the time for all good people to
   support us Multicians.
```

produces

```
          Now is the
       time  for all
       good   people
       to support us
       Multicians.
```

Name: fortran, ft

SYNTAX AS A COMMAND

```
ft path {-control_args}
```

FUNCTION

invokes the FORTRAN compiler.

ARGUMENTS

path
is the pathname of a FORTRAN source segment; you need not give the fortran suffix.

CONTROL ARGUMENTS

-ansi66
interprets the program according to the 1966 standard for FORTRAN, with Multics FORTRAN extensions. (Default)

-ansi77
interprets the program according to the 1977 standard for FORTRAN, with Multics FORTRAN extensions.

-auto_zero
initializes to zero, when allocated, automatic storage for local variables. (Default) * MCR 7069

-binary_floating_point, -bfp
makes the internal representation of floating point numbers be 2 ** exponent * mantissa. (Default)

-brief, -bf
writes error messages in short form.

-brief_table, -bftb
generates partial symbol table giving correspondence between source line numbers and object locations.

-card
specifies that the source segment is in card-image format.

-check, -ck
checks the source segment for syntactic and semantic errors. Code generation is skipped; no object segment is produced.

-check_multiply, -ckmpy
produces code that checks for integer multiplication overflow. (Default, if you supply -ansi77 but not -optimize)

-debug_io, -dbio
establishes a new command level after a run-time I/O error.

-default_full, -dff
sets the default optimizer to "full_optimize" (see -optimize). (Default)

-default_safe, -dfs
sets the default optimizer to "safe_optimize" (see -optimize).

-fold
maps uppercase letters to lowercase form.

-full_optimize, -full_ot
invokes the full optimizer to speed up program execution and reduce its size.

-free
specifies that the source segment is in free-form format. (Default)

-hexadecimal_floating_point, -hfp
allows the use of HFP numbers. Real, complex, and double-precision numbers can have four times more magnitude in HFP mode at the expense of some precision. Compilations using -hfp and execution of any such programs require rw access to >sc1>admin_acs>Fortran_hfp.acs. This feature is only supported on the DPS8 hardware. Your site may disallow the use of HFP code by not creating the above access control segment or by denying access to it.

-large_array, -la
is used when more automatic or static storage is required by a program than would fit in stackframes/linkage sections. Individual arrays and common blocks must not exceed 255K words. (See -very_large_array.)

-line_numbers, -ln
gives a source segment with line numbers.

-list, -ls
produces a complete source program listing plus an assembly-like listing.

-long, -lg
writes error messages in long form. (Default)

-long_profile, -lpf
generates extra code to meter execution of individual statements using virtual CPU time and page fault measurements.

-map
produces complete source program listing.

-no_auto_zero
does not initialize to zero, when allocated, automatic storage for local variables; that is, initial values of variables are undefined. It is faster than -auto_zero.

-no_check_multiply, -nckmpy
does not produce code that checks for integer multiplication overflow. (Default, if you specify -ansi66 or -optimize)

-no_debug_io, -ndbio
does not establish a new command level after a run-time I/O error. (Default)

-no_fold
does not map uppercase letters into lowercase form. (Default)

-no_large_array, -nla
suppresses large_array.

-no_line_numbers, -nln
    does not give a source segment with line numbers.

-no_map
    produces no program listing. (Default)

-no_optimize, -not
    performs no optimizations. (Default)

-no_stringrange, -nstrg
    does not produce range-checking code for all substring references. (Default, if you use -ansi66 or -optimize)

-no_subscriptrange, -nsubrg
    does not produce range-checking code for all subscripted array references. (Default, if you supply -ansi66 or -optimize)

-no_table, -ntb
    does not generate a runtime symbol table. (Default, if you give -optimize)

-no_version
    does not print out the version of the current compiler.

-no_very_large_array, -nvla
    suppresses -very_large_array. (Default)

-no_vla_parm
    inhibits -vla_parm.

-non_relocatable, -nrlc
    inhibits generation of relocation information by the compiler. The resulting object segment cannot be bound.

-optimize, -ot
    invokes an extra compiler phase just before code generation to perform certain optimizations.

-profile, -pf
    generates extra code to meter execution of individual statements.

-relocatable, -rlc
    generates relocation information. (Default)

-round
    rounds intermediate results of real and double-precision calculations before storing. (Default)

-safe_optimize, -safe_ot
    like -optimize, but inhibits some code movement.

-severity N, -sv N
prints only error messages whose severity is N or greater (where N is 1, 2, 3, or 4). (Default: 1)

-stringrange, -strg
produces range-checking code for all substring references. (Default, if you give -ansi77 but not -optimize)

-subscriptrange, -subrg
produces range-checking code for all subscripted array references. (Default, if you specify -ansi77 but not -optimize)

-table, -tb
generates full symbol table. For use with the probe and debug commands. (Default, unless you use -optimize)

-time, -tm
prints table giving time (in seconds), number of page faults, and size of temporary area for each phase of the compiler.

-time_ot
prints out timing information on the subphases of the optimizer.

-top_down
optimizes loops using a top-down approach.

-truncate, -tc
truncates intermediate results of real and double-precision computations before storing.

-version
prints out the version of the FORTRAN compiler before compiling.

-very_large_array, -vla
is used when arrays or common blocks have sizes exceeding segment size (255K < array size or common block length < 2**20 [16M] words). It implies -large_array.

-vla_parm
is used in subroutines when parameters that are passed to them could come from very-large-array procedures. It allows the use of very-large arrays but not of large arrays within the subroutine.

NOTES ON SEVERITY VALUES

This command associates the following severity values to be used by the severity active function:

Value	*Meaning*
0	No compilation yet or no error
1	Warning
2	Correctable error
3	Fatal error
4	Unrecoverable error
5	Could not find source.

NOTES

Mutually exclusive control arguments are:

-ansi66 and -ansi77
-long_profile and -profile
-optimize, -safe_optimize and -stringrange, -subscriptrange
-round and -truncate

For more information on the FORTRAN compiler and on using FORTRAN on Multics, see the *Multics FORTRAN Reference Manual* (AT58) and the *Multics FORTRAN User's Guide* (CC70).

Name: fortran_abs, fa

SYNTAX AS A COMMAND

```
fa paths {-ft_args} {-dp_args} {-control_args}
```

FUNCTION

submits an absentee request to perform FORTRAN compilations.

ARGUMENTS

paths
are pathnames of segments to be compiled.

ft_args
are one or more control arguments accepted by the fortran command.

dp_args
are one or more control arguments (except -delete) accepted by the dprint command.

CONTROL ARGUMENTS

-hold
specifies that fortran_abs should not dprint or delete the listing segment.

-limit N, -li N
places a limit on the CPU time used by the absentee process. N must be a positive decimal integer specifying the limit in seconds. An upper limit is defined by your 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. (Default: defined by your site for each queue)

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

-queue N, -q N
is the priority queue of the request (see "Notes" for a description of the interaction with the dprinting of output files). (Default: defined by your site)

NOTES

The absentee process for which fortran_abs submits a request compiles the segments named and dprints and deletes the listing segment. If you specify no -of, the output segment path.absout is created in your working directory (if more than one path is specified, only the first is used). If none of the segments to be compiled can be found, no absentee request is submitted.

You can freely mix control arguments and can put segment pathnames anywhere on the command line after fortran_abs. All control arguments apply to all segment pathnames. If you give an unrecognizable control argument, the absentee request is not submitted.

Unpredictable results may occur if you submit two absentee requests that could simultaneously attempt to compile the same segment or write into the same absout segment.

When doing several compilations, give several pathnames in one command invocation. With one command, only one process is set up; thus the dynamic intersegment links that need to be snapped when setting up a process and when invoking the compiler need be snapped only once.

If you give no -q, the request is submitted into the default absentee priority queue defined by your site and, if requested, the output files is dprinted in the default queue of the request type specified on the command line. (If you give no request type, "printer" is used.)

If you give -q, the output files is dprinted in the same number queue as the absentee request. If the request type specified for dprinting does not have that queue, the highest numbered queue available for the request type is used and a warning is issued.

This page intentionally left blank.

Name: gcos, gc

SYNTAX AS A COMMAND

```
gc job_deck_path {-control_args}
```

FUNCTION

invokes the GCOS environment simulator to run a single GCOS job, immediately, in your process.

ARGUMENTS

job_deck_path
is the pathname of the file containing a GCOS job deck. The file can contain ASCII lines or can be in GCOS standard system format.

CONTROL ARGUMENTS

-ascii, -aci
input file contains ASCII lines.

-block N, -bk N
specifies the block size to be used for gcos tape buffer (in words). For buffers larger than 2800 words, the use must have rw access to >sc1>rcp>workspace.acs. Use of the ",block=nnnn" control on the volume_id for nstd_ tape attachments overrides this buffer size setting. If multiple buffer sizes are supplied on a command line, the rightmost is used. (Default: 2800 words; max 4096)

-brief, -bf
suppresss printing of all terminal output except fatal error messages.

-continue, -ctu
continues processing the job when a nonfatal error occurs.

-debug, -db
causes the simulator to call debug command under certain error conditions.

-dprint, -dp
queues the converted print files for printing followed by deletion (-list is implied).

-dprint_options options, -dpo options
queues the converted print files for printing, but use -dprint supplied in the options string (-list and -dprint are implied). The options argument must be enclosed in quotes.

-dpunch, -dpn
queues converted punch files for punching in raw mode, followed by deletion (-raw is implied).

-dpunch_options options, -dpno options
queues converted punch files for punching, but use -dpunch supplied in the options string (-raw and -dpunch are implied). The options argument must be enclosed in quotes.

-gcos, -gc
input file in GCOS standard system format.

-hold, -hd
does not perform the default conversion and daemon output of print and punch files.

-job_id id, -jd id
uses job identification specified by id in the names of files created by the simulator for this job. The id can be any character string of up to 18 characters, or -unique to indicate Multics unique name, or -jd_seg (-jd) to indicate the entryname of job deck segment.

-list, -ls
converts print files (both sysout and simulated printer) from BCD (or binary) to ASCII and delete the BCD copy.

-long, -lg
prints certain lines from the execution report in addition to normal terminal output.

-lower_case, -lc
translates alphabetic BCD characters in print files to lowercase ASCII. (Default: uppercase)

-no_bar, -nobar, -nb
request the simulator run the slave programs in Multics mode instead of BAR mode.

-no_canonicalize, -nocan, -no
ASCII input file contains no tab characters, and the fields on all the card images are aligned in the columns required by GCOS.

-parameter parameters, -pm parameters
specifies that the remaining arguments in the command line are to be used to override the $PARAM card values.

-raw
converts punch files (both sysout and simulated card punch) from BCD to format suitable for punching in raw mode and delete the BCD copy.

-smc path
specifies a directory to be used as the system master catalog. When -smc is given, the first field in catalog/filename descriptions is retained.

-syot_dir path, -sd path
puts GCOS format copies of print, punch, and sysout files in directory named "path" rather than working directory. (Default)

-temp_dir path, -td path
puts temporary GCOS files in directory named "path" rather than process directory. (Default)

-truncate, -tc
discards ASCII input file lines longer than 80 characters (after canonicalization). If not given, the first line longer than 80 characters causes the job to be rejected.

-userlib
enables the use of GCOS slave software libraries supplied by you.

NOTES

Related facilities include the GCOS daemon, which provides batch processing for GCOS jobs under Multics, and appropriate commands, used to manipulate GCOS format files that reside in the Multics storage system. These commands, and the gcos command, are fully described in the *Multics GCOS Environment Simulator Manual* (AN05).

EXAMPLES

If no control arguments are given, the command

```
! gcos path
```

is equivalent to the command

```
! gcos path -aci -dpo -dl -dpno "-dl -raw" -id -jd -bk 2800
```

Name: general_ready, gr

SYNTAX AS A COMMAND

```
gr {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[gr {-control_args}]
```

FUNCTION

prints a ready message containing specified values in a specified format.

LIST OF PREFIX CONTROL ARGUMENTS

You must use these control arguments prior to the format ones. They allow you to override the default formats for the contents of the ready message.

-string
allows you to specify the character string at the beginning of the ready message. The argument following -string is used instead of "r" at the beginning of the ready message. Since it is put into the ioa_ control string, you can use "^/", "^R", and "^B" to cause new lines, red ribbon shifts, and black ribbon shifts, respectively.

-control
allows you to specify the entire ioa_ control string used to format the ready message. The string is passed to ioa_$nnl without change so it must contain specifications for each of the various values to be included in the ready message. The ioa_ control string formats for the various values that you can insert into the ready message are given below for each type of value (see "List of Format Control Arguments"). This control argument overrides any format arguments that would normally affect the format of the ready message; however, you must still give format keywords to indicate which values are to be output and the order in which these values correspond to the ioa_ control characters in the control string.

LIST OF FORMAT CONTROL ARGUMENTS

The format and content of the ready message are controlled by format control arguments. They include control arguments that identify values to be included in the ready message, optional precision numbers following some of these control arguments that control the number of digits after the decimal point in numeric values, and literal character strings that are inserted in the ready message. The format control arguments are combined in the order of their appearance in general_ready to form an ioa_ control string that controls the format of the ready message. You can use seven types of values in a ready message:

1. content values
2. processor usage values (virtual CPU seconds)
3. memory usage values (memory units)
4. usage cost values (dollar charges)
5. paging operations (and demand page faults)
6. command processor (level numbers)
7. date/time values (date, time of day, day of the week, etc.).

Both total usage values (total usage accrued during this process) and incremental usage values (usage accrued since the last ready message printed by general_ready) can be output in the same ready message. The values are selected for use in the ready message by format control arguments to general_ready. The format control arguments are listed below by type.

Content values:

-active_string STR, -astr STR
expands the active string STR each time the ready message is printed.

Processor usage values:

-inc_rcpu {N}
incremental real CPU value.

-inc_vcpu {N}
incremental virtual CPU value.

-total_rcpu {N}
total real CPU value.

-total_vcpu {N}
total virtual CPU value.

where N can be a single numeric digit from 1 to 9, indicating the number of digits that should appear to the right of the decimal point in the number that is output. The default is three digits. The output format of the value can be described by the ioa_ control string "^.Nf", where N is 3 by default.

Memory usage values:

-inc_mem_units {N}
incremental units.

-total_mem_units {N}
total memory units.

These control arguments are used in the same manner as the ones for processor usage values.

Usage cost values:

-inc_cost {N}
incremental cost charges.

-total_cost {N}
is the cost charges.

These control arguments are used in the same manner as the ones for processor usage values except that the default number of digits following the decimal point is two. The output format of the value can be described by the ioa_ control string " $^.Nf" where N is 2 by default.

Charges being accrued for devices (terminal, tape mounts, etc.) are not updated in the PIT where the user can see them. Therefore, general_ready estimates the dollar figure based on connect/cpu time.

Paging operations values:

-inc_bf is the incremental bounds faults.

-inc_pft
is the incremental page faults.

-inc_sf
is the incremental segment faults.

-inc_vr
is the incremental VTOC reads.

-inc_vw
is the incremental VTOC writes.

-total_bf
is the bounds faults.

-total_pft
is the page faults.

-total_sf
is the segment faults.

-total_vr
is the VTOC reads.

-total_vw
is the VTOC writes.

These control arguments are output by the ioa_ control string "^d", where ^d is the number of demand page faults.

Command processor values:

-level, -lev
specifies the number of command processor invocations to be included in the ready message. The level numbers are output by the ioa_ control string "^a", but the printed format can be described by "level ^d". If the command processor level is 1, the printed format is the null string. The number of digits is not settable.

-frame, -fr
specifies the number of stack frame level numbers to be included in the ready message. The level numbers are output by the ioa_ control string "^a", but the printed format can be described by "frame ^d". If the command processor level is 1, the printed format is the null string.

If you give both control arguments, the printed format can be described by "level ^d,^d".

Date/time values:

-date
is your default date format. Type "print_time_defaults date" to display the format and "gr -date" to display a sample date value.

-date_time
is your default date/time format. Type "print_time_defaults date_time" to display the format and "gr -date_time" to display a sample date/time value.

-day
is a two-digit day (dd).

-day_name
is a three-character day of the week (www).

-hour
is a two-digit hour (hh).

-minute
is a two-digit minute (mm)

-month
is a two-digit month (mm).

-time, -tm
is your default time format. Type "print_time_defaults time" to display the format and "gr -time" to display a sample value.

-time_format STR, -tfmt STR
where STR is a time format control string defining a user-specified format for any of the various date values (see "Time Format" in Section 1).

-year
is a two-digit year (yy)

-zone
is a three-character time zone (zzz).

These values can be described by the ioa_ control string "^a" except for the -day, -minute, and -year control arguments, which use the ioa_ control string "^a" (without a leading space). The number of digits is not settable.

LIST OF OPERATION CONTROL ARGUMENTS

The following control arguments affect the operation of gr, but do not change the format of ready messages.

-call cmdline
when used with -set, calls the command processor to execute cmdline after the completion of every command line. The argument cmdline is a single argument to gr; you must therefore enclose it in quotes if it contains any blanks. A frequent use of -call is "-call print_messages -brief"; cmdline is executed even if the printing of ready messages has been inhibited by executing ready_off.

-reset
resets incremental usage values to zero without printing a ready message.

-revert
makes the system ready procedure the current ready message procedure and resets any timer alarms to catch shift changes.

-set
establishes gr as the current ready message procedure and sets an alarm timer to catch shift changes. The command processor then calls gr to print a ready message after each command line is complete. In addition, the ready, ready_on, and ready_off system commands determine the operation of gr.

NOTES ON OPERATION CONTROL ARGUMENTS

The -revert and -set control arguments are mutually exclusive. A gr command that includes -set does not print a ready message; instead it saves the ioa_ control string built from the format and prefix control arguments in the command and uses this ioa_ string to control the format of ready messages printed when command lines complete execution or when a ready command is issued. A gr command that includes -revert prints a single ready message only if format or prefix control arguments appear in the command with -revert; otherwise no ready message is printed.

If you give neither -revert nor -set, gr prints one ready message according to the format and prefix arguments given in the command.

This command is designed to allow an almost arbitrary format at no additional cost (relative to the system's ready procedure) other than the one associated with gr, which sets up the ready message. Once a ready message is specified, the ready, ready_on, and ready_off commands control the printing of the ready message in the normal manner.

This page intentionally left blank.

The command builds up an ioa_ control string (see the the Subroutines manual) from the order of the keywords passed to it. The keywords specify which values to output in the ready message. Virtual CPU usage and cost can be printed. Both incremental usage (usage accrued since the last ready message produced by general_ready) and total usage (usage accrued during this process) can be in the same ready message with the precision of the output (the number of decimal places to the right of the decimal point) you specified. As a command, you can use general_ready to either print a single ready message or define the contents of the ready message printed by the ready command (and after every command line if you execute ready_on); as an active function, the return value is the ready message.

The values for total virtual CPU time and total memory units is valid across new processes. The value for cost is valid unless a shift change occurred during a previous process. When you invoke general_ready for the first time in a process, the cost of all usage in that process up to that time is computed at the rates then in effect.

Due to the manner in which ready message procedures and procedures that set up alarm timers are invoked, such procedures should not be terminated (by the terminate and terminate_refname commands). If you want to terminate general_ready, invoke it with -revert before it is terminated.

EXAMPLES

The following examples illustrate some of the facilities of general_ready:

```
gr -string READY -date ^xTIME -time ^xVCPU -inc_vcpu
   -total_vcpu -set
```

establishes general_ready as the current ready procedure since the -set keyword appeared. Each ready message has the format

```
READY 01/15/83 TIME 1234.3 VCPU 3.456 23.349
```

If the -set keyword had not appeared, a single ready message having the above format is printed. The ioa_ control string that general_ready uses to generate the above ready message is:

```
"READY ^a^xTIME ^a^xVCPU ^.3f ^.3f^2/"
```

The command line

```
gr -string READY -date -hour : -minute ^xVCPUI -inc_vcpu ^xVCPUT
   -total_vcpu 2
```

results in a single ready message of the form

```
READY 01/15/83 09:46 VCPUI 2.345 VCPUT 34.21
```

using the ioa_ control string

```
"READY ^a ^a:^a^xVCPUI ^.3f^xVCPUT ^.2f^2/"
```

You can specify the above ready message by the command line:

```
gr -control "READY ^a ^a:^a VCPUI ^.3f VCPUT ^.2f^2/" -date -hour
   -minute -inc_vcpu -total_vcpu
```

Name: generate_pnotice

SYNTAX AS A COMMAND

```
generate_pnotice {-control_args}
```

FUNCTION

allows Multics source and object archives and executable software to be legally protected via copyright or trade secret notices and provides software identification via Software Technical Identifiers (STIs).

CONTROL ARGUMENTS

-id STR
: specifies the Marketing Identifier (MI) of the product as derived from psp_info_. This control argument and -name are mutually exclusive.

-name STR, -nm STR
: specifies the product's generic name found in psp_info_.

-special
: used in cases where there may be no entry in psp_info_ for the software being protected. This likely occurs when you are protecting software in an experimental or development library. You are prompted for the information to be put into the PNOTICE segments. (See "Notes.")

-sti STR
: specifies a valid 12-character STI. You can use it to override the STI found in psp_info_ when you give -name or -id.

NOTES

This command allows protection of software residing in a library other than the one specified in psp_info_ or of software not specified in psp_info_, via -special.

The command generates ALM source and object segments with the names of "PNOTICE.<generic name>.alm" and "PNOTICE.<generic name>", where <generic name> comes from the psp_info_ data base or from -special.

These segments contain the text of one or more software protection notices and three 12-character STIs. The segments are appended to a product's primary source and object archives, as defined in the psp_info_ data base. If you select -special, you must provide these archive names. If PNOTICE segments with the same name exist in the archives, they are replaced. Order the archives such that these segments are the first components. The binding of the object archive places the protection notices and STIs into the bound segment as well. Make the bindfile "Order" statement indicate that the PNOTICE component be first. Don't retain the PNOTICE segment name in the bound segment.

To find PNOTICE segments' information for installed products, issue the display_psp command. Unless you use -special, the source and object archives must be in your working directory, in which case you must have sma access to the directory as well as rw access to the archives; then you can specify archive pathnames to generate_pnotice. If you supply -special, access is checked, and if it is not sufficient it is forced; otherwise, access is not forced.

The command asks you the following set of questions when you select -special. Have the requested information ready.

Generic name?
: You supply a short (<= 20 characters) name that is descriptive of the module(s) being protected. The name can be the same one contained in psp_info_ if the module is a newer version; otherwise, you can create the name.

STI?
: This is the Software Technical Identifier, a 12-character identifier used by Honeywell to provide information on released software products. It can be blank for nonproducts.

Include the notices from psp_info?
: The module(s) being protected have an entry in psp_info_. You are asked whether the notices there are to be included.

Source pnotice name?
: You are asked to provide primary names of notices, without the .pnotice suffix, for protection of source. When done, type "q". Use the list_pnotice_names command for available names.

Object pnotice name?
: You are asked to provide primary names of notices, without the .pnotice suffix, for protection of object and executable. When done, type "q". Use the list_pnotice_names command for available names.

Pathname of source archive?
: You are asked to provide an archive pathname of the source archive. The ".archive" suffix is not required, but can be given.

Pathname of object archive?
You are asked to provide an archive pathname of the object archive. The suffix .archive is not required, but can be given.

These two archives need reside neither in the same directory nor in the working directory.

*

Name: get_dir_quota

SYNTAX AS A COMMAND

```
get_dir_quota {paths} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

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

FUNCTION

prints information about the directory quota and pages used by inferior directories.

ARGUMENTS

paths
are pathnames of directories for which you want quota information. If one of the paths is -working_directory (-wd), your working directory is used. If you don't supply paths, your working directory is assumed. The star convention is allowed.

CONTROL ARGUMENTS

-long, -lg
includes the cumulative time-page product for the current accounting period. The active function doesn't accept it.

-quota
returns the terminal quota on each directory. (Default: to return 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.

-records_used, -rec_used
returns the number of pages used in each directory.

ACCESS REQUIRED

You require status permission on each directory for which you want quota. 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) prints the number of pages of quota used by the segments in that directory and in any inferior directories charging against that quota. The output is prepared in tabular format, with a total, when you specify more than one pathname; when you give only one, 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 you last updated this number. Thus, you can see what secondary storage charges your accounts are accumulating. If you have inferior directories with nonzero quotas, you need print this product for all these directories in order to obtain the charge.

NOTES ON ACTIVE FUNCTION

Supply only one directory in the active function. The star convention is not allowed.

You can specify any of -quota, -records_left, or -records_used; the default is -quota.

Name: get_effective_access, gea

SYNTAX AS A COMMAND

```
gea paths {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[gea paths {-control_args}]
```

FUNCTION

returns the specified user's effective access on the given path (see Section 6 of the Programmer's Reference Manual).

ARGUMENTS

paths
are the pathnames of segments or directories for which effective access is to be determined. You can use the star convention.

CONTROL ARGUMENTS

-interpret_as_extended_entry, -inaee
interprets the selected entries as extended entry types.

-interpret_as_standard_entry, -inase
interpret the selected entries as standard entry types.

-ring RING
gets the effective access assuming the user is in the specified ring. (Default: user's validation level)

-user USER
finds the effective access for USER. You can use the star convention. (Default: user's own User_id)

Name: get_ips_mask

SYNTAX AS A COMMAND

```
get_ips_mask {-control_args}
```

FUNCTION

prints the current state of the IPS mask for the calling process.

CONTROL ARGUMENTS

-brief, -bf
prints the names of masked signals. If no IPS signals are masked, it prints nothing.

-long, -lg
prints a more descriptive message about the status of IPS signals, masked or unmasked. (Default)

NOTES

If all undefined IPS signals are either masked or unmasked, they are not mentioned. If, however, some are masked and others are not, an octal list is printed. This can only happen when you have supplied an invalid (probably reinitialized) value in a call to set that mask.

Name: get_library_segment, gls

SYNTAX AS A COMMAND

```
gls seg_names {-control_args}
```

FUNCTION

finds source or object segments in the Multics system libraries and copies them into your current working directory. You can specify which system libraries are to be searched and the order of the search. You can also search for user libraries that may not be organized like the Multics system libraries. (See "Operation" below.)

This command has functionally been replaced by library_fetch.

ARGUMENTS

seg_names
are the names of the segments to be found, including any language suffix.

CONTROL ARGUMENTS

-brief, -bf
does not print pathnames. (Default)

-control path, -ct path
looks in the directory specified by path to find the control segments. The path argument can be -working_directory (-wd) to specify the current working directory (see "Operation" below). If -control is not specified, the command looks in the directory >ldd to find its control segments.

-long, -lg
prints the pathname of the segment from which each segment is copied.

-rename new_name, -rn new_name
copies the immediately preceding seg_name into your process directory and then into a segment in the working directory. The new_name can be an equal name, in which case the equal convention is applied to the seg_name; otherwise, the segment created in the working directory is named new_name. The new_name cannot be a pathname.

-sys lname
uses the control segment "lname.control".

NOTES

If you don't give -sys, get_library_segment uses all the control segments specified in the root directory, whose default is >ldd. For a complete list of the control segments, type

```
! list -pn >ldd -all **.control
    hard
    standard
    unbundled
    auth_maint
    network
    languages
    tools
```

You can give multiple -sys in the same command invocation. If so, all the control segments referenced by the lnames in these arguments are searched. The order in which the control segments are processed and searched is determined by the order in which the lnames appear in the command and the directores referenced by each lname appear in the lname control segment.

Control arguments and segment names can be interspersed throughout the command invocation.

NOTES ON USER LIBRARIES

You can supply -control to extract segments from a user library, causing the command to use a control segment with the pathname path><keyword>.control. This allows you to search your own library structure, using your own search procedure or one of the Multics system library search procedures listed below.

OPERATION

If you don't select -control, gls searches for segments in one or more of the Multics system libraries. From each keyword given in a -sys, it constructs a pathname of the form >ldd><keyword>.control. It uses this as the pathname of a control segment. This control segment tells gls which directories are to be searched and how to search them.

Each control segment contains one or more lines of the form

```
directory_path: search_procedure;
```

where

directory_path
is the absolute pathname of a directory to be searched.

search_procedure
is the name of a procedure that searches the directory to find seg_name. This name can have the form

segment_name

or

segment_name$entry_name

For each directory_path specified in the control segment, gls initiates the search_procedure and calls it to search the directory. The calling sequence for search_procedure is

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

call search_procedure (directory_path, seg_name, containing_seg, code);
```

where

directory_path
is the absolute pathname of a directory to be searched. (Input)

seg_name
is the name of the segment to be found, including any language suffix. (Input)

containing_seg
is the name of the segment in directory_path in which seg_name is found. This name is either the same as seg_name or the name of an archive containing seg_name. (Output)

code
is a standard storage system status code. (Output)
0 seg_name is found in directory_path>containing_seg
1 seg_name is not found.

If code is 0 and the final eight nonblank characters of containing_seg are the archive suffix, gls issues the command

```
archive x directory_path>containing_seg seg_name
```

to extract the segment into the current working directory. If you specify -rename for seg_name, the segment is extracted and given the new name.

If code is 0 and the final eight nonblank characters of containing_seg are not the archive suffix, gls calls copy_seg_ to copy directory_path>seg_name into the current directory, unless you have given a -rename, in which case the segment is copied into directory_path>new_name.

If code is 1, gls continues the search with the next directory_path in the current control segment. If the current control segment contains no more directory_paths, the search continues with the first directory_path in the next control segment you specify. If the segment has not been found after all control segments have been exhausted, gls prints an error message and begins searching for the next seg_name.

If search_procedure returns a code that is neither 0 nor 1, the error message corresponding to the error code is printed and search_procedure continues the search as if code were 1.

The get_primary_name_ procedure is used to find segments in the Multics system libraries.

If you don't give -sys, gls uses all the control segments in >ldd.

EXAMPLES

The command line

```
! gls abc.pl1 -sys tools -sys sss random.alm
```

copies abc.pl1 and random.alm from the directories specified in >ldd>tools.control and >ldd>sss.control if they exist.

```
! gls -sys lang xyz.pl1 -sys os -sys hard
```

searches for xyz.pl1 in the directories specified by the set of control segments in >ldd.

```
! gls gorp.pl1 -rename glop.pl1
```

searches the default group of directories for segment gorp.pl1 and copies it into your working directory with the name glop.pl1.

```
! gls fortran_blast_ bound_parse_.bind -sys lang.o
```

searches for the segment fortran_blast_ and the bind segment bound_parse_.bind in the directories specified in >ldd>lang.o.control.

You can use gls to extract a copy of the source program alpha.pl1 from a private library archive with the command

```
! gls -ct >udd>Project_id>Person_id -sys source alpha.pl1
```

if >udd>Project_id>Person_id>source.control contains the line

```
   >udd>Project_id>Person_id>library: get_primary_name_;
```

and if alpha.pl1 is a component of some archive segment >udd>Project_id>Person_id>library, having alpha.pl1 as one of its names.

Name: get_mode

SYNTAX AS A COMMAND

```
get_mode MODE_STR MODE_NAME
```

SYNTAX AS AN ACTIVE FUNCTION

```
[get_mode MODE_STR MODE_NAME]
```

FUNCTION

extracts the value of a mode from a mode string.

ARGUMENTS

MODE_STR
is a mode string, as used with an I/O module.

MODE_NAME
is the name of a mode contained in the mode string.

NOTES

For a boolean mode, the value returned is "true" or "false", otherwise it is the character string value of the mode.

Name: get_pathname, gpn

SYNTAX AS A COMMAND

```
gpn {-control_arg} arg
```

SYNTAX AS AN ACTIVE FUNCTION

```
[gpn {-control_arg} arg]
```

FUNCTION

returns the absolute pathname of the segment designated by a specified reference name or segment number. If the reference name or segment number is not in use, an error message is printed.

ARGUMENTS

arg
is a reference name or octal segment number known to this process.

CONTROL ARGUMENTS

-name, -nm
indicates that arg (which happens to look like an octal segment number) is to be interpreted as a reference name. If this control argument is not specified, the system assumes arg is a reference name only if arg is not a valid octal number.

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 you want quota information. If one of the paths is -working_directory (-wd), your working directory is used. If you don't supply paths, your working directory is assumed. The star convention is allowed.

CONTROL ARGUMENTS

-long, -lg
includes the cumulative time-page product for the current accounting period and the corresponding price according to the rate structure of the current process.

-nonzero, -nz
lists directories with nonzero quota used only.

-quota
returns the terminal quota on each directory. (Default: to return 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.

-records_used, -rec_used
returns the number of pages used in each directory.

-sort
sorts directories by the requested quota value or by records used if you request more than one value. The largest value is printed first.

-total, -tt
returns, for quota used, the total quota used by the subtree. Master directories in a subtree are not included in its total.

-zero
lists directories with zero quota used only.

ACCESS REQUIRED

You require status permission on each directory for which you want quota. 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) prints the number of pages of quota used by the segments in that directory and in any inferior directories charging against that quota. The output is prepared in tabular format, with a total, when you specify more than one pathname; when you give only one, a single line of output is printed.

The long form of output gives the quota and pages-used information provided in the short output and prints the logical volume identifier of segments, the time-record product in units of record days, and the date you last updated this number; thus you can see what secondary storage charges your accounts are accumulating. If you have inferior directories with nonzero quotas, you need print this product for all these directories to obtain the charge.

NOTES ON ACTIVE FUNCTION

Supply only one directory in the active function. You can't use the star convention.

The active function doesn't accept -long, -nonzero, -sort, and -zero.

You can specify any of -quota, -records_left, or -records_used; the default is -quota.

Name: get_system_search_rules, gssr

SYNTAX AS A COMMAND

```
gssr
```

FUNCTION

prints the definitions of site-defined search rule keywords acceptable to the set_search_rules command.

NOTES

This command prints a list of standard search rule keywords and directories, each one followed by one or more site-defined keywords. If you include a site-defined keyword in the search segment accepted by set_search_rules, the site-defined keyword expands into its definition in the order printed by get_system_search_rules.

See add_search_rules, delete_search_rules, print_search_rules, and set_search_rules.

Name: greater

SYNTAX AS A COMMAND

```
greater STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[greater STRA STRB]
```

FUNCTION

returns true if STRA is greater than STRB according to the ASCII collating sequence, otherwise returns false.

NOTES

The strings are compared character by character according to their ASCII code value: if the first character in each string has the same ASCII code value, compare the second character; if their values are identical, compare the third character; etc.

To make numeric comparisons of strings see the descriptions of ngreater and nless.

Name: hash_table, ht

SYNTAX AS A COMMAND

```
ht path nb
```

FUNCTION

used to create a hash table and to insert, delete, and search for entries in it. It uses the hash_ subroutine.

ARGUMENTS

path
specifies the name of a segment, which is an existing hash table.

nb
is the (optional) number of buckets with which the hash table is to be created. If you don't give nb or if it is out of range (0 < nb <= 6552), then a default is assigned to it.

LIST OF REQUESTS

The command operates in response to the following requests given by you. Each request code must be the first character of the line and followed by one or more arguments separated by any number of blanks (a blank before the first argument is optional).

```
a       add
d       delete
q       quit
s       search
```

The a Request

SYNTAX

```
a name1 value1...nameN valueN
```

FUNCTION

inserts an entry into the hash table for namei and its corresponding valuei.

ARGUMENTS

namei
is a character string less than, or equal to, 32 characters.

valuei
is a decimal number you associate with namei to indicate its location in the corresponding data table. It can be array subscript.

The d Request

SYNTAX

```
d name1...nameN
```

FUNCTION

deletes the entry namei from the hash table and prints the value it was associated with.

ARGUMENTS

namei
is a character string less than, or equal to, 32 characters.

The q Request

SYNTAX

```
q
```

FUNCTION

returns control to command level.

The s Request

SYNTAX

```
s name1...nameN
```

FUNCTION

searches the hash table for namei and prints its corresponding value. You can then locate namei in your data table by using valuei.

ARGUMENTS

namei
is a character string less than, or equal to, 32 characters.

NOTES

If the hash table ever becomes full or inefficient, the number of buckets is doubled or assigned the maximum, the hash table is rehashed, and a message is printed.

Name: have_mail

SYNTAX AS A COMMAND

```
have_mail mbx_specification {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[have_mail {mbx_specification} {-control_args}]
```

FUNCTION

returns "true" if there is mail in the specified mailbox.

ARGUMENTS

mbx_specification
specifies the mailbox to be examined. If not given, your default mailbox (>udd>Project_id>Person_id>Person_id.mbx) is used.

LIST OF MBX SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table (see "Notes on Mailbox Selection by User" below).

STR
is any noncontrol argument and is first interpreted as -mailbox STR. If no mailbox is found, it is then interpreted as -save. If no savebox is found, it is then interpreted as -user STR.

CONTROL ARGUMENTS

-interactive_messages, -im
returns "true" if there are any interactive messages in the mailbox. (Default)

-mail, -ml
returns "true" if there is any mail in the mailbox. (Default)

-no_interactive_messages, -nim
returns "true" only if there is mail in the mailbox, ignoring whether there are any interactive messages present.

-no_mail, -nml
returns "true" only if there are interactive messages in the mailbox, ignoring whether there is any mail in the mailbox.

NOTES ON MAILBOX SELECTION BY USER

The user's default mailbox is specified in the form Person_id.Project_id. For an entry in the mail table, STR is usually in the form of Person_id (the mail table is fully described in the *Extended Mail System User's Guide,* CH23).

If STR contains one period and no white space, it is interpreted as a User_id that specifies the user's default mailbox; otherwise, it is interpreted as the name of an entry in the mail table. For example,

```
-user DBuxtehude.SiteSA
```

is interpreted as a User_id that identifies a default mailbox. On the other hand,

```
-user "George G. Byron"
-user L.v.Beethoven
-user Burns
```

are all interpreted as the names of entries in the mail table: the first because it contains white space; the second because it contains more than one period; the third because it contains no period.

When interpreted as a User_id, the STR cannot contain any angle brackets (<>) and must have the form Person_id.Project_id, where "Person_id" cannot exceed 28 characters and "Project_id" 32 characters. In this case, "-user STR" is equivalent to the mbx_specification -mailbox >udd>Project_id>Person_id>Person_id.mbx.

When interpreted as the name of a mail table entry, STR cannot contain any commas, colons, semicolons, backslashes (\), parentheses, angle brackets, braces ({}), quotes, commercial at-signs (@), or white space other than spaces. The query of the mail table is performed in a case-insensitive manner. Use the display_mailing_address command to determine the actual address corresponding to the STR. The address in the mail table must identify a mailbox.

ACCESS REQUIRED

If you give either -no_interactive_messages or -no_mail, you must have rs extended access to the mailbox; otherwise, you only need s extended access.

EXAMPLES

You can use the following statement in your start_up.ec to invoke read_mail only when there is mail present in your mailbox:

```
&if [have_mail -no_interactive_messages] &then read_mail -list
```

Name: have_messages

SYNTAX AS A COMMAND

```
have_messages mbx_specification {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[have_messages {mbx_specification} {-control_args}]
```

FUNCTION

returns "true" if there are any interactive messages in the specified mailbox.

ARGUMENTS

mbx_specification
specifies the mailbox to be examined. If not given, your default mailbox (>udd>Project_id>Person_id>Person_id.mbx) is used.

LIST OF MBX_SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table (see "Notes on Mailbox Selection by User" below).

STR
is any noncontrol argument and is first interpreted as -mailbox STR. If no mailbox is found, it is then interpreted as -save. If no savebox is found, it is then interpreted as -user STR.

CONTROL ARGUMENTS

-interactive_messages, -im
returns "true" if there are any interactive messages in the mailbox. (Default)

-mail, -ml
returns "true" if there is any mail in the mailbox. (Default)

-no_interactive_messages, -nim
returns "true" only if there is mail in the mailbox, ignoring whether there are any interactive messages present.

-no_mail, -nml
returns "true" only if there are interactive messages in the mailbox, ignoring whether there is any mail in the mailbox.

NOTES ON MAILBOX SELECTION BY USER

The user's default mailbox is specified in the form Person_id.Project_id. For an entry in the mail table, STR is usually in the form of Person_id (the mail table is fully described in the *Extended Mail System User's Guide,* CH23).

If STR contains one period and no white space, it is interpreted as a User_id that specifies a user's default mailbox; otherwise, it is interpreted as the name of an entry in the mail table. For example,

```
-user DBuxtehude.SiteSA
```

is interpreted as a User_id that identifies a default mailbox. On the other hand,

```
-user."George G. Byron"
-user L.v.Beethoven
-user Burns
```

are all interpreted as the names of entries in the mail table: the first because it contains white space; the second because it contains more than one period; the third because it contains no period.

When interpreted as a User_id, the STR cannot contain any angle brackets (<>) and must have the form Person_id.Project_id, where "Person_id" cannot exceed 28 characters and "Project_id" 32 characters. In this case, "-user STR" is equivalent to the mbx_specification -mailbox >udd>Project_id>Person_id>Person_id.mbx.

When interpreted as the name of a mail table entry, STR cannot contain any commas, colons, semicolons, backslashes (\), parentheses, angle brackets, braces ({}), quotes, commercial at-signs (@), or white space other than spaces. The query of the mail table is performed in a case-insensitive manner. Use the display_mailing_address command to determine the actual address corresponding to the STR. The address in the mail table must identify a mailbox.

ACCESS REQUIRED

You must have rs extended access to the mailbox; however if you give -mail but give no -no_interactive_messages, you only need s extended access.

Name: have_queue_entries

SYNTAX AS A COMMAND

```
have_queue_entries ms_paths
```

SYNTAX AS AN ACTIVE FUNCTION

```
[have_queue_entries ms_paths]
```

FUNCTION

returns "true" if there are messages in the specified queue message segments.

ARGUMENTS

ms_paths
: are relative pathnames or star names for queue message segments to be examined. The suffix ms is added if not present. You can use the star convention.

ACCESS REQUIRED

You need s extended access to the queue message segments.

EXAMPLES

```
[have_queue_entries >ddd>idd>x1200_3]
```

returns "true" if there are any requests in queue 3 of the I/O daemon "x1200" request type.

```
[have_queue_entries >udd>idd>printer_*]
```

returns "true" if there are any requests in any of the I/O daemon printer queues.

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. The command selects this information from segments maintained online, which are in a special format, called information segments (info segs).

ARGUMENTS

info_names
: specify the information to be printed. The suffix .info is assumed. If you supply a pathname, it identifies the info seg to be printed; otherwise help searches for segments matching an entryname using the info_segments search list. For subroutines you can include an entry point name in the info_name (e.g., subroutine_$entry_point). You can use the star convention except when you specify an entry point name or when you give -entry_point. (See "Notes on Star Convention" below.)

  If you select no info_names, help prints the default info seg (help_system.gi.info), which gives a brief introduction to the help facility.

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

LIST OF CONTROL ARGUMENTS BY FUNCTION

The control arguments are arranged below functionally. Detailed descriptions follow the list, in the same order.

help

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.
-list_entry_points, -lep
    prints entry points.
-title
    prints section titles.

Starting Paragraph

-search STRs, {-case_sensitive} {-non_case_sensitive},
-srh STRs {-cs} {-ncs}
    selects by words in paragraph.
-section STRs {-case_sensitive} {-non_case_sensitive},
-scn STRs {-cs} {-ncs}
    selects by section title.

Paragraph Grouping

-maxlines J
    sets maximum paragraph grouping size.
-minlines I
    sets minimum paragraph size.

Info-Seg Selection

-entry_point, -ep
    selects main subroutine entry point.
-pathname path, -pn path
    selects an info segment.

CONTROL ARGUMENTS FOR INFORMATION SELECTION

The following control arguments select the kind of information that help prints. If you specify no control arguments from this group, help prints a long heading line, followed by the first paragraph of info. At the end of each paragraph, help asks you 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/active functions, a list of control arguments and/or other keywords used by the command.

-brief_header, -bfhe
prints a brief heading line (which consists of the heading and line count), followed by either information selected by the other information selection control arguments or by the first paragraph if you specify no other control arguments from this group.

-case_sensitive, -cs
when used with either -section or -search, performs the requested action in a case-sensitive manner.

-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.

-header, -he
prints only a long heading line consisting of the pathname of the info seg, heading, and line count. It conflicts with all other information selection control arguments. (Default)

-list_entry_points, -lep
lists the entry points in a subroutine info segment.

-non_case_sensitive, -ncs
when used with either section or search, performs the requested action in a non-case-sensitive manner. This is the default.

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

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

CONTROL ARGUMENTS FOR SELECTING A STARTING PARAGRAPH

Normally, help begins printing the first info paragraph. The control arguments below can select a particular section and/or paragraph at which printing is to start.

-search STRs {case_sensitive} {-non_case_sensitive},
-srh STRs {-cs} {-ncs}
begins printing with the first paragraph containing STRs. All the strings must appear in the selected paragraph, but they can appear in any order. You can type STRs in lowercase since case is ignored when matching, unless -es is specified. All arguments and control arguments following -srh are treated as STRs, so should be put -srh at the end of the command line. An exception is that if -cs or -ncs is the last argument of the string, it will be interpreted as a control argument and not one of the STRs. The search usually begins with the first paragraph, but when you also specify -scn it begins with the matching section and continues to the last paragraph (i.e., without wraparound).

-section STRs {-case_sensitive} {-non_case_sensitive},
-scn STRs {-cs} {-ncs}
begins printing the section whose title contains STRs. The entire section title is not required. The first section whose title matches STRs is selected. You can put STRs in any order in the section title. You can type them in lowercase since case is ignored during matching operations, unless -cs is specified. All arguments following -scn until the next control argument are treated as STRs.

When you supply -srh or -scn and no matching paragraph is found in any of the info segs selected by an info_name or info seg selection control argument, that info seg is passed over without comment. Thus, the starting-paragraph control arguments serve as a secondary info selection mechanism.

You can use the control arguments from this group with any of the information selection control arguments, but their effect differs depending upon which of them are used. When -srh or -scn is used with -he, help prints the heading lines, not the matching paragraph, for infos containing a matching paragraph. When you use them with -bf or -ca, help prints a heading line and then the information selected by -bf or -ca; the matching paragraph is not printed.

When -srh or -scn is used with the -cs control argument, help will print only exact matches. If the -ncs control argument is used, help will match to both uppercase and lowercase values. The default is -ncs.

When you use -srh or -scn with -bfhe, help prints a brief heading line preceding the matching paragraph. When you use them with -title, help prints a heading line, then the list of section titles, and finally the matching paragraph. When you use them with -a, the entire info is printed for infos containing a matching paragraph.

CONTROL ARGUMENTS FOR PARAGRAPH GROUPING

The following control arguments determine how much information help prints before asking if you want to see more.

-minlines I
sets the minimum paragraph size to I lines. Paragraphs smaller than this size are printed with preceding paragraphs. (Default: 4)

-maxlines J
sets the maximum paragraph grouping size to J lines. (Default: 15)

Paragraphs that you have seen are not grouped with unseen paragraphs. Paragraphs at the end of one section are not grouped with those beginning another section. Paragraphs are not grouped when you give -srh or -scn.

CONTROL ARGUMENTS FOR SELECTING INFO SEGS

-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 you omit -ep and specify no entry point name by an info_name identifying a subroutine info segment, help prints the info describing the general purpose of the subroutine.

-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 your working directory or when the pathname begins with a minus sign. For subroutines you can include an entry point name with the final entryname of path; for example,

```
! help -pn >udd>Project_id>Person_id>info>subr_$entry_pt
```

A suffix of .info is assumed if you give none. You can use the star convention except when you give an entry point name or -ep. (See "Notes on Star Convention.")

LIST OF RESPONSES

The responses accepted when help questions you 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.

.
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.

?
prints a list of available responses.

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

help

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. You can specify EP_NAME as entry_point_name or subroutine_$entry_point_name; if you omit it, help skips to the description of the subroutine_$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.

list_entry_points, lep
lists the entry points in a subroutine info segment.

list_requests {STRs}, lr {STRs}
prints information about available help requests.

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} {-all_entrypoints}, r {-scn} {-aep}
prints the rest of the info seg without intervening questions. If you choose -scn, help prints only the rest of the current section without questions. When the section has been printed, help then asks whether you want to see the next section.
If -all_entrypoints is specified, help will print the rest of the remaining entry points.

search {STRs} {-top} {-case_sensitive} {-non_case_sensitive},
srh {STRs} {-t} {-cs} {-ncs}
skips to, and prints, the next paragraph containing STRs. Paragraph selection is performed as described above for -srh. If you give -t, searching starts at the beginning of the info seg. If the -cs argument is used, the search will be case-sensitive and find only exact matches. If -ncs is used, the search will match both uppercase and lowercase values. The default is -ncs. If STRs are omitted, help uses the strings from the previous search response or -srh. If the search fails, help prints the message:

```
No matching paragraph found.
```

and repeats the previous question.

section {STRs} {-top} {-case_sensitive} {-non_case_sensitive},
scn {STRs} {-t} {-cs} {-ncs}
skips to the next section whose title contains STRs. Title matching is performed as described above for -scn. If you supply -t, title searching starts at the beginning of the info. If you omit STRs, help uses the search strings from the previous section response or -scn. If the -cs argument is used, the search will be case-sensitive and find only those section titles that are exact matches. If the -ncs argument is used, help will match both uppercase and lowercase values. The default is -ncs.
If the search fails, help prints the message:

```
No matching section found.
```

and repeats the previous question.

skip {-section} {-rest} {-seen} {-entry_point}, s {-scn} {-r} {-seen} {-ep}
skips the next paragraph and asks whether you want to see the paragraph following it. If you select -scn, help skips all paragraphs of the current section. If you supply -r or -ep, help skips the rest of this info seg or subroutine entry point description, continuing with the next. If you give -seen, help skips to the next paragraph that you haven't seen. You can use only one of these control arguments at a time.

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

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

yes, y
prints the next paragraph of information, then asks whether you want more help.

This command remembers which paragraphs you have seen and which you have skipped or not yet reached. It asks you to "Review" paragraphs seen before and asks if "More help" is needed for unseen paragraphs. It stops printing if you have seen all paragraphs when you reach the end of the info. If you skipped any paragraphs, however, help asks if you want to see them; if you answer "yes," the first unseen paragraph is printed. You can then answer "skip -seen" to view subsequent unseen paragraphs. The question/answer dialogue continues until all the information is printed or you reply "no."

CONTENTS OF INFO SEGS

Each segment contains one or more blocks of information that describe a particular command/active function, subroutine, or topic. To validate the format of info segs, use validate_info_seg.

An info seg begins with a heading line, consisting of a date on which it was last modified and a brief title identifying it. For command/active function info segs the program name, including any short name, is used as the title; for subroutine info segs the subroutine name is used.

Information in an info seg is divided into paragraphs, separated from one another by two blank lines. The help command uses this separation to determine where one paragraph ends and the next begins.

Each paragraph contains a logically complete unit of information. Control arguments and responses are available to search for, and print, a particular paragraph. To avoid printing unnecessary information when you perform such searches, paragraphs are short (1 to 15 lines long) and deal with only a single subject.

The paragraphs describing a given topic are grouped together into a section. The first paragraph of each section begins with a title that names the topic described in that section. Section titles are short, usually consisting of one or two words followed by a colon (:).

Standard section titles are used in info segs provided with the Multics system so that users can search for a particular information topic. For command/active function info segs the standard section titles in their proper order are:

Syntax As a Command:
shows how the program is invoked. Arguments are given a generic name (e.g., paths indicates that one or more pathnames are allowed). Optional arguments are shown in braces (e.g., {paths}). If the program allows control arguments, they are shown as -control_args in the syntax line.

Function:
gives a brief description of what the program does.

Arguments:
gives a brief description of each argument.

Control Arguments:
gives a brief description of each control argument.

Notes:
gives comments, clarifications, or any special-case information.

The descriptions of arguments and control arguments are formatted in a special way so that "help -bf" can print a list of all argument and control argument names and "help -ca" can find and print the description of an individual argument or control argument. Each description begins with a line naming the argument or control argument, including the short name, and any operands it requires. This naming line begins in column 1. The description continues on subsequent lines by defining the meaning and function of the argument or control argument. These lines are indented three spaces from the left margin.

Subroutines are described by info segs containing a series of specially formatted information blocks, one describing each subroutine entry point. The first block describes the general purpose of the subroutine and can include control information and notes common to all entry points. It includes the following sections:

Function:
describes the overall function performed by the subroutine. The heading is optional, the description is not.

Entry points in SUBROUTINE:
causes help to list the entry points defined in the subroutine. Precede this line by two blank lines and make it the last line (followed by two blank lines) before the first entry point description.

Entry point descriptions are separated from the first block and each other by the following line:

```
:Entry: yyy: 12/07/86  SUBROUTINE_$ENTRYPOINT
```

The date is the date-last-modified. Use of -ep causes help to search through these lines for a ENTRYPOINT matching STR. Always precede this line by two blank lines.

Section titles in their standard order for subroutine entry point descriptions are as follows:

Function:
gives a brief description of what the entry point does.

Usage:
gives the PL/I declare and call statements for the entry point. A sample description from the cu_$arg_count entry point follows:

```
Usage:
declare cu_$arg_count entry (fixed bin, fixed bin (35));
call cu_$arg_count (nargs, code);
```

Arguments:
lists the arguments shown in the call statement along with a brief description of their functions.

Notes:
are general notes that apply to the entry point.

To keep info segs concise, avoid tutorial notes and examples except in special cases.

NOTES ON SEARCH LIST

The help command uses the "info_segments" (or "info_segs" or "info") search list. The default info seg directories contain info segs provided by your site and those supplied with the system. Type "print_search_paths info_segments" to see what the current info segments search list is. For more information about search lists, see the search facility commands, especially add_search_paths.

help

NOTES ON STAR CONVENTION

When you use the star convention, help performs the following steps:

1. The info segs whose entrynames match any of the star names are alphabetized within their directory and scanned in that order.

2. When you give -srh and -scn help scans the matching info seg until the desired paragraph and/or section is found. If a matching paragraph is found, help prints it, then asks you whether to print remaining paragraphs. Any section and search responses given at this point scan only the current info seg. If a matching paragraph is not found in one of the info segs selected by a star name, that info seg is passed over without comment. Thus, it is possible to scan all info segs and print only those containing certain section titles or certain words.

3. When you supply no -srh and -scn, help begins printing the first paragraph of each info seg that matches any of the starnames. Then help asks you whether to print the remaining paragraphs.

4. The -a, -bf, -ca, and -title control arguments apply to each info seg selected by the starnames and -srh/-scn string matching. Section titles, a brief summary, or particular control argument descriptions are printed before the matching paragraph. When you combine -a with -srh or -scn, the entire info seg selected by the string matching is printed without questions.

5. The no, rest, skip, and yes responses operate on the next selected paragraph. This paragraph can be the first paragraph of the next selected info seg or even the first paragraph that matches the -srh and -scn criteria in the next selected info seg.

6. If you issue a quit signal, you can use program_interrupt to reenter the interactive help environment. The question asked prior to the quit is repeated.

INFO SEG 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 PL/I compiler command is called pl1.info.

Information about changes made to a command/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 online. To find information about a particular area of the system, use list_help or -he with an entryname containing stars to list the names of available infos.

USER-CREATED INFO SEGMENTS

You can create info segs describing your own commands, exec_coms, and application programs. To create proper info segs see "Contents of Info Segs" above.

Name: hexadecimal, hex

SYNTAX AS A COMMAND

```
hex values
```

SYNTAX AS AN ACTIVE FUNCTION

```
[hex values]
```

FUNCTION

returns one or more values in hexadecimal.

ARGUMENTS

value
is a value to be processed. The last character of the value indicates its type. Acceptable types are binary (b), quaternary (q), octal (o), hexadecimal (x), or unspec (u).

Any valid PL/I real value is allowed. The absence of any specifier means decimal. The unspec value is limited to eight characters.

EXAMPLES

```
! string [hex 377o]
ff

! hex abcu
184c463
```

Name: high

SYNTAX AS A COMMAND

```
high N
```

SYNTAX AS AN ACTIVE FUNCTION

```
[high N]
```

FUNCTION

returns a specified number of copies of the last (highest) character in the ASCII character set, the PAD character of 177 octal.

Name: high9

SYNTAX AS A COMMAND

```
high9 N
```

SYNTAX AS AN ACTIVE FUNCTION

```
[high9 N]
```

FUNCTION

returns a specified number of copies of the last (highest) 9-bit bit pattern, 777 octal.

Name: history_comment, hcom

SYNTAX AS A COMMAND

```
hcom operation path {args} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[hcom operation path {args} {-control_args}]
```

FUNCTION

adds, checks, displays, formats, and updates software change history comments within a given source module.

ARGUMENTS

operation
designates the operation to be performed.

path
is the name of a source code program that requires history comments. Include the language suffix.

args
are optional arguments appropriate to the particular operation to be performed.

CONTROL ARGUMENTS

are optional control arguments that vary, depending on the particular operation to be performed.

HISTORY COMMENT FORMAT

Following is a PL/I history comment example. Other languages will have different comment delimiters.

```
/**^  HISTORY COMMENTS:
1)  change(85-05-12, DOppenheimer), approve(85-05-25, MCR2355),
    audit(85-05-26, EBlau), install(85-05-30, MR11.0-3382):
    Increased size of test_array to eliminate subscript error.
2)  change(85-05-28, MLee), approve(85-05-29 MCR2356),
    audit(85-06-05, TYoffe), install(85-06-10, MR11.0-3384):
    Added the -brief and -long control arguments.
                                        END HISTORY COMMENTS /
```

NOTE: To determine if prior history comments exist in the module, the source module is checked for a line containing the history comment block beginning, i.e., a line beginning with the appropriate comment delimiter and "HISTORY COMMENTS:". If found, the program then checks for the history comment block ending, i.e., a line containing "END HISTORY COMMENTS."

LIST OF HISTORY COMMENT FIELDS

The fields within a given history comment are identified as follows:

```
NO) change (CHANGE_DATE, CHANGE_PERSON_ID),
    approve (APPROVE_DATE, APPROVE_ID),
    audit (AUDIT_DATE, AUDIT_PERSON_ID),
    install (INSTALL_DATE, INSTALL_ID): SUMMARY
```

The fields in a history comment are named as described below. The sample validation routine hcom_default_validate_ validates field formats used by the Multics Development Center as described below. Each site, however, can provide its own validation routines to tailor the contents of the user-settable field values.

NO
is the number of the history comment. Comments are numbered sequentially in chronological order, starting with 1. (Supplied by hcom)

CHANGE_DATE
is the date (yy-mm-dd) on which the history comment was first added to the source module. (Supplied by hcom)

CHANGE_PERSON_ID
is the Person_id of whoever added the history comment. (Supplied by hcom)

APPROVE_DATE
is the date (yy-mm-dd) on which an approval value was supplied for a history comment. (Supplied by hcom)

APPROVE_ID
is the identifier authorizing the change. The default validation routine expects an identifier in the form "TYPEnnnn" for Multics change requests (MCRs), post-installation bug fix (PBFs) associated with MCRnnnn, or Multics emergency change request (MECRs) (e.g., MCR6734, PBF6734, MECR0102). For critical fixes the identifier should be in the form of fix_nnnn or fix_nnnn.ds. The maximum length of this field is 24 characters. (Supplied by user)

AUDIT_DATE
is the date (yy-mm-dd) audit field added to the history comment. (Supplied by hcom)

AUDIT_PERSON_ID
is the Person_id of whoever audited the source module. (Supplied by hcom)

INSTALL_DATE
is the date (yy-mm-dd) install field added to the history comment. (Supplied by hcom)

INSTALL_ID
is the value identifying either a specific installation or the installer of a critical fix. The default validation routine expects an identifier in the form "MRrel-nnnnn", consisting of a release number and installation sequence count (e.g., MR12.0-00234). For a critical fix the validation routine expects a Person_id naming the person who installed the fix. The maximum length of this field is 24 characters. (Supplied by user)

SUMMARY
is a brief description of the change made to the module. This field contains text (up to 2000 characters) and is not validated. (Supplied by user)

NOTES

The following is a typical usage pattern expected for the various operations of the command:

- You make a change to the source module. You can add a new history comment by hand (perhaps using an Emacs extension to prompt for field values). Or, after adding the change, you can use the hcom add operation to add a new comment. A typical command line might be

```
hcom add prog.pll
```

- You may not have had approval for the change at the time the history comment was added. When approval is gained, you can use the hcom add_field operation to add the approve field. For example,

```
hcom af prog.pll -approve MCR7235
```

- You can display the history comments in a program or even compare the comments in a modified version of a program with those in the library copy of the program. For example,

```
hcom display prog.pll new -orig [lpn prog.pll]
```

displays the new history comments in the source module, while

```
hcom compare prog.pll -orig [lpn prog.pll]
```

displays the differences between the source module and the original module.

- When the change is audited, the auditor uses the hcom add_field operation to supply an audit field for all new or incomplete history comments. For example,

```
hcom af prog.pll -audit
```

- When you are ready to submit the change for installation, you use the hcom check operation to ensure that all comment fields except the install field have been supplied in each changed module. Since you have a site-defined validation routine called hcom_site_validate_ in your object search rules, this routine is used to fully validate the fields of all comments.

```
hcom check prog.pll -orig [lpn prog.pll]
```

- When the installer receives the modules in an installation, he uses the hcom install operation to ensure that new history comments describing the changes are present. This operation also adds an identifier to each new comment, indicating in which installation it was installed. The installer can use a special library-defined validation routine to perform special field validations. Here, the library validation routine is called hcom_mdc_validate_:

```
hcom install prog.pll -vdt hcom_mdc_validate_ -install
   MR12.0-0023 -orig [lpn prog.pll]
```

VALIDATION ROUTINE CALLING SEQUENCE

A site can define a site-wide history comment validation routine to validate the contents of the APPROVE_ID and INSTALL_ID fields of history comments. This routine is called hcom_site_validate_. If it is found in your object search rules, hcom uses this validation routine instead of using hcom_default_validate_. The -validate control argument allows use of a user-supplied validation subroutine, which can have any name, to validate the APPROVE_ID and INSTALL_ID fields.

The calling sequence of both the hcom_site_validate_ subroutine and user-written routines is shown below.

```
dcl hcom_site_validate_ entry (char() var, char() var,
    char() var, bit(1), char() var, char() var, char(100) var);

call hcom_site_validate (caller, field_name, input_value,
     result_bit, canonical_value, field_type, error_msg);
```

where:

caller
is the name of the calling program on whose behalf the validation routine should report errors, ask questions, etc. (Input)

field_name
is the name of the field being validated. It can have a value of either APPROVAL_FIELD_NAME or INSTALL_FIELD_NAME. These named constants are declared in hcom_field_names.incl.pl1. (Input)

input_value
is the field value you supply. (Input)

result_bit
is either "1"b if the input value is valid or "0"b if the input value is invalid. (Output)

canonical_value
is the canonical text form of the field_name and input_value. (Output)

field_type
is the canonical text form of the field_name for use in error messages. (Output)

error_msg
is the text of the error message. (Output)

Operation: add

SYNTAX AS A COMMAND

```
hcom add path {-control_args}
```

FUNCTION

adds a new history comment to the requested module. The summary field is required; all other fields are optional.

ARGUMENTS

path
is the name of a source code program that requires history comments. Include the language suffix. You can give an archive pathname.

CONTROL ARGUMENTS FOR FIELD INPUT

-approve APPROVE_ID, -apv APPROVE_ID
specifies the APPROVE_ID field. The maximum length of this field is 24 characters. (See "List of History Comment Fields" above for a description of valid APPROVE_IDs.)

-fill, -fi
sets fill mode on for the summary field. In fill mode text, words are moved from line to line in such a way that the last word does not extend past the right margin. (Default)

-input_approve, -iapv
prompts for an APPROVE_ID. This is a single-line field value. (Default)

-input_install, -iin
prompts for the INSTALL_ID. This is a single-line field.

-input_summary, -ism
prompts you for the summary field. This is a multiline field. (See "Notes" below.) (Default)

-install INSTALL_ID, -in INSTALL_ID
specifies an identifier associated with installing the changed module into execution libraries. See "List of History Comment Fields" above for a description of valid INSTALL_IDs. The maximum length of this field is 24 characters.

-no_approve, -napv
specifies that an APPROVE_ID is not being entered.

-nofill, -nfi
sets the fill mode off for the summary field.

-no_install, -nin
suppresses the prompt for INSTALL_ID. (Default, since the installation ID is usually specified when the module is being installed rather than when the history comment is first added)

-summary TEXT, -sm TEXT
gives text describing the change. Enclose the text within quotes if it contains spaces, quotes, parentheses, etc.

CONTROL ARGUMENTS

-critical_fix, -cfix
specifies that critical-fix history comments are allowed in the program. All comments following the first that contains critical-fix field values must also contain critical-fix field values.

-validate RTN, -vdt RTN
validates user-supplied fields in the history comment, using a user-supplied validation routine. RTN must be a virtual entrypoint name acceptable to cv_entry_. If you give no -vdt, the default is to validate using the hcom_site_validate_ subroutine, if your site has provided it, or the hcom_default_validate_ subroutine provided with hcom.

NOTES

For multiline fields all input is treated as text until reading a line with just a period. Input lines beginning with ".." are treated as Multics command lines, rather than as part of the field value. After the command line is executed, you can continue answering the prompt or can replace input text typed so far with a new answer. Optional field values answered with a period omit the field from the history comment.

For single-line fields, input ends when you type a carriage return. If the input line begins with "..", the text that follows is treated as a Multics command line. After the command line is executed, you are prompted again for the question. Optional field values answered with a carriage return omit the field from the history comment.

Operation: add_field, af

SYNTAX AS A COMMAND

```
hcom af path {comment_specs} {-control_args}
```

FUNCTION

inserts missing fields in selected comments.

ARGUMENTS

path
is the name of a source code program that requires history comments. Include the language suffix. You can give an archive pathname.

comment_specs
specify which history comment(s) are to be updated. (See "List of Comment Specifiers" below.) (Default: to select comments that are missing the fields given by the "Control Arguments for Field Input")

CONTROL ARGUMENTS FOR FIELD INPUT

-approve APPROVE_ID, -apv APPROVE_ID
inserts the missing APPROVE_ID field. The maximum length of this field is 24 characters. (See "List of History Comment Fields" above for a description of valid APPROVE_IDs.)

-audit, -aud
inserts the user's Person_id in the AUDIT_PERSON_ID field.

-input_approve, -iapv
prompts for a new APPROVE_ID. This is a single-line field value. (Default, if you give no field input control arguments)

-input_install, -iin
prompts for the INSTALL_ID. This is a single-line field.

-install INSTALL_ID, -in INSTALL_ID
specifies an identifier associated with installing the changed module into execution libraries. The maximum length of this field is 24 characters. (See "List of History Comment Fields" above for a description of valid INSTALL_IDs.)

-no_approve, -napv
does not replace the APPROVE_ID field nor prompts for missing approve fields. (Default, if you supply any field input control arguments)

-no_audit, -naud
does not add the AUDIT_PERSON_ID field. (Default)

-no_install, -nin
does not set the INSTALL_ID field nor prompts for missing install fields. (Default)

CONTROL ARGUMENTS

-critical_fix, -cfix
specifies that critical-fix history comments are allowed in the program. All comments following the first that contains critical-fix field values must also contain critical-fix field values.

-validate RTN, -vdt RTN
validates user-supplied fields in the history comment, using a user-supplied validation routine. RTN must be a virtual entrypoint name acceptable to cv_entry_. If you give no -vdt, the default is to validate using the hcom_site_validate_ subroutine, if your site has provided it, or the hcom_default_validate_ subroutine provided with hcom.

LIST OF COMMENT SPECIFIERS

I, I:J
selects the comment(s) by a comment number or a range of numbers. You can use the keywords "first" (f) and "last" (l) to identify the first and last comments.

all, a
selects all comments.

approved, apv
selects comments that have an approve field.

audited, aud
selects comments that have an audit field.

complete, cpt
selects comments that include all fields.

incomplete, icpt
selects comments that are missing the approve, audit, or install field.

installed, in
selects comments that have an install field.

new
selects, when you give -original, comments that do not appear in the original (earlier) version of the program.

old
selects, when you give -original, comments that appear in both the original and new versions of the program.

unapproved, unapv
selects comments that do not have an approve field.

unaudited, unaud
selects comments that do not have an audit field.

uninstalled, unin
selects comments that do not have an install field.

NOTES

If you provide no control args, the default is to print selected history comments and to prompt you for missing approve fields.

Operation: check, ck

SYNTAX AS A COMMAND

```
hcom ck path {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[hcom ck path {-control_args}]
```

FUNCTION

looks for one or more incomplete (or new if you give -original) history comments and verifies that their summary, approve, and audit fields are given while the install field is missing. The active function returns true if the check succeeds (the history comments are ready for submission), false otherwise.

ARGUMENTS

path
: is the name of a source code program that has history comments. Include the language suffix. You can give an archive pathname.

CONTROL ARGUMENTS

-errors, -er
: displays history comments that failed check. (Default)

-no_errors, -ner
: suppresses display of history comments that failed check. (Default, for the active function)

-original orig_path, -orig orig_path
: specifies that the current version of the source program is to be cross-checked with an earlier version (given as orig_path) to ensure that there are new history comments in the current module. You can give an archive pathname and use the equal convention. (Default: to check for incomplete history comments in the given source program)

-validate RTN, -vdt RTN
: validates user-supplied fields in the history comment, using a user-supplied validation routine. RTN must be a virtual entrypoint name acceptable to cv_entry_. If you give no -vdt, the default is to validate using the hcom_site_validate_ subroutine, if your site has provided it, or the hcom_default_validate_ subroutine provided with hcom.

NOTES

The presubmission check is run by developers to ensure that at least one history comment has been added to describe modifications to the source module. These history comments will be incomplete because they will not have an install field. Supply all other fields prior to submission.

Operation: compare, cmp

SYNTAX AS A COMMAND

```
hcom cmp path -control_args
```

SYNTAX AS AN ACTIVE FUNCTION

```
[hcom cmp path -control_args]
```

FUNCTION

displays any differences between the source module and the original module. The active function returns true if the comments in the source and original modules are identical, false otherwise.

ARGUMENTS

path
: is the name of a source code program that has history comments. Include the language suffix. You can give an archive pathname.

CONTROL ARGUMENTS

-original orig_path, -orig orig_path
: specifies the pathname of an earlier version of the module. You can give an archive pathname and use the equal convention. (Required)

-validate RTN, -vdt RTN
: validates user-supplied fields in the history comment, using a user-supplied validation routine. RTN must be a virtual entrypoint name acceptable to cv_entry_. If you give no -vdt, the default is to validate using the hcom_site_validate_ subroutine, if your site has provided it, or the hcom_default_validate_ subroutine provided with hcom.

Operation: display, ds

SYNTAX AS A COMMAND

```
hcom ds path {comment_specs} {-control_args}
```

FUNCTION

displays selected history comments. Optionally, compares history comments in a program with those in an earlier version of the program, displaying old comments (which appear in both versions) or new comments (which do not appear in the earlier version).

ARGUMENTS

path
is the name of a source code program. Include the language suffix. You can give an archive pathname.

comment_specs
select which history comment(s) to display. (See "List of Comment Specifiers" under the add_field operation.) (Default: to display new comments if you specify -original or all comments if you omit -original)

CONTROL ARGUMENTS

-original orig_path, -orig orig_path
specifies the pathname of an earlier version of the module. You can give an archive pathname and use the equal convention.

-validate RTN, -vdt RTN
validates user-supplied fields in the history comment, using a user-supplied validation routine. RTN must be a virtual entrypoint name acceptable to cv_entry_. If you give no -vdt, the default is to validate using the hcom_site_validate_ subroutine, if your site has provided it, or the hcom_default_validate_ subroutine provided with hcom.

Operation: exists

SYNTAX AS A COMMAND

```
hcom exists path {comment_specs} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[hcom exists path {comment_specs} {-control_args}
```

FUNCTION

prints or returns true if any history comments matching all the comment_specs are found in every selected module, false otherwise.

ARGUMENTS

path
is the name of a source code program that has history comments. Include the language suffix. You can give an archive pathname.

comment_specs
select which history comment(s) to print. (See "List of Comment Specifiers" under the add_field operation.) (Default: "all," to check whether any comments exist in the source module)

CONTROL ARGUMENTS

-original orig_path, -orig orig_path
specifies the pathname of an earlier version of the module. You can give an archive pathname and use the equal convention.

-validate RTN, -vdt RTN
validates user-supplied fields in the history comment, using a user-supplied validation routine. RTN must be a virtual entrypoint name acceptable to cv_entry_. If you give no -vdt, the default is to validate using the hcom_site_validate_ subroutine, if your site has provided it, or the hcom_default_validate_ subroutine provided with hcom.

Operation: format, fmt

SYNTAX AS A COMMAND

```
hcom fmt path {comment_specs} {-control_args}
```

FUNCTION

reformats selected history comments in a program, including putting date fields into standard "yy-mm-dd" format, filling lines of all comment entries to a 79-character width, validating field values, etc.

ARGUMENTS

path
is the name of a source code program whose history comments are to be reformatted. Include the language suffix. You can give an archive pathname.

comment_specs
select which history comment(s) to reformat. (See "List of Comment Specifiers" under the add_field operation.) (Default: "all," to check whether any comments exist in the source module)

CONTROL ARGUMENTS

-fill, -fi
sets fill mode on for the summary field. In fill mode text, words are moved from line to line in such a way that the last word does not extend past the right margin. (Default)

-nofill, -nfi
sets the fill mode off for the summary field.

-no_renumber, -nrnb
prints an error if history comment numbers are out of sequence. (Default)

-renumber, -rnb
specifies that the history comments within the current module can be renumbered if they are out of sequence.

-validate RTN, -vdt RTN
validates user-supplied fields in the history comment, using a user-supplied validation routine. RTN must be a virtual entrypoint name acceptable to subroutine cv_entry_. If you give no -vdt, the default is to validate using the hcom_site_validate_ subroutine, if your site has provided it, or the hcom_default_validate_ subroutine provided with hcom.

Operation: get

SYNTAX AS A COMMAND

```
hcom get path comment_specs -control_args
```

SYNTAX AS AN ACTIVE FUNCTION

```
[hcom get path comment_specs -control_args]
```

FUNCTION

prints or returns given field values from selected history comments.

ARGUMENTS

path
is the name of the source code program whose history comments fields are to be returned. Include the language suffix. You can give an archive pathname.

comment_specs
specify from which history comment(s) field values are extracted. Give at least one specifier. (See "List of Comment Specifiers" under the add_field operation.)

CONTROL ARGUMENTS

-field_name FIELDS, -fn FIELDS
specify which fields from the selected history comments are to be returned or printed. All arguments following -fn up to the first argument that begins with a hyphen are considered field names. See "List of Field Names" below. (Default: to return or print all fields of matching entries)

-original orig_path, -orig orig_path
specifies the pathname of an earlier version of the module. You can give an archive pathname and use the equal convention.

-validate RTN, -vdt RTN
validates user-supplied fields in the history comment, using a user-supplied validation routine. RTN must be a virtual entrypoint name acceptable to cv_entry_. If you give no -vdt, the default is to validate using the hcom_site_validate_ subroutine, if your site has provided it, or the hcom_default_validate_ subroutine provided with hcom.

LIST OF FIELD NAMES

You can supply the following values with -fn to specify which field to return.

approve_date, apvdt
is the date on which the approve field was entered.

approve_id, apvi
is the identifier from the approve field.

audit_date, auddt
is the date on which the audit field was entered.

audit_person_id, audpi
is the Person_id of whoever audited the source module.

change_date, cdt
is the date on which the history comment was first entered.

change_person_id, cpi
is the Person_id of whoever entered the history comment.

install_date, indt
is the date on which the install field was entered.

install_id, ini
is the identifier from the install field.

summary, sm
is the summary field from the history comment.

NOTES

If several history comments are selected, specified fields from the first selected comment are returned or printed, followed by fields from the second selected comment, etc. If the selected field is not present in a given history comment, then a null string is returned for that field. Multiline field values are returned in a single line (with newline characters replaced by a space) as a quoted string.

Operation: install

SYNTAX AS A COMMAND

```
hcom install path -control_args
```

SYNTAX AS AN ACTIVE FUNCTION

```
[hcom install path -control_args]
```

FUNCTION

performs a preinstallation check on modules being installed, as specified by system integration personnel. It performs a variety of steps, including checking that new history comments exist and are properly filled in. If the check succeeds, an installation ID is placed in the comment. The active function returns true if the check succeeds (the history comments are ready for installation), false otherwise.

ARGUMENTS

path
: is the name of a source code program that requires history comments. Include the language suffix. You can give an archive pathname.

CONTROL ARGUMENTS FOR FIELD INPUT

-approve APPROVE_ID, -apv APPROVE_ID
: specifies the APPROVE_ID field to be assigned to all history comments that are missing an approve field. An error occurs if you give -apv but no comments are missing the approve field. (See "List of History Comment Fields" above for valid APPROVE_IDs.) This control argument is used when only the installer knows what the approval identifier is; e.g., only he knows what the MECR number is because this number is assigned at installation time. The maximum length of this field is 24 characters. If the AUDIT_DATE and AUDIT_PERSON_ID fields are missing when you use -apv, an error message is issued but processing continues.

-input_approve, -iapv
: prompts for an APPROVE_ID. This is a single-line field value.

-input_install, -iin
: prompts for the installation identifier. (Default)

-install INSTALL_ID, -in INSTALL_ID
specifies an identifier associated with installing the changed module into execution libraries. This identifier is placed in all history comments that are missing the install field. An error occurs if every comment has an install field. See "List of History Comment Fields" above for valid INSTALL_IDs. The maximum length of this field is 24 characters.

-no_approve, -napv
specifies that an APPROVE_ID is not being entered. (Default)

CONTROL ARGUMENTS

-critical_fix, -cfix
specifies that critical-fix history comments are allowed in the program. All comments following the first that contains critical-fix field values must also contain critical-fix history comments.

-errors, -er
displays history comments that fail the installation checks. (Default)

-no_errors, -ner
suppresses display of failing history comments. (Default, for the active function)

-original orig_path, -orig orig_path
specifies the pathname of an earlier module copy that is already installed in the software library. This library copy is compared with the version being submitted (see "Notes" below). You can give an archive pathname and use the equal convention.

-validate RTN, -vdt RTN
validates user-supplied fields in the history comment, using a user-supplied validation routine. RTN must be a virtual entrypoint name acceptable to cv_entry_. If you give no -vdt, the default is to validate using the hcom_site_validate_ subroutine, if your site has provided it, or the hcom_default_validate_ subroutine provided with hcom.

NOTES

The install operation performs the following steps:

1. Make a working copy of history comments in the new module.

2. If -original is given, check comments in the original module against those in the working copy:

   a) check for comments in the original that do not appear in the working copy. This indicates changes that have been backed out. If any are found, print an error and stop further checking.

   b) copy the install identifier from comments in the original module into corresponding comments in the working copy that are missing this identifier. This may occur when the developer makes changes to a modified version of the program before that version is installed in the libraries.

3. If -approve or -input_approve is given, check for comments in the working copy that are missing the approve field. If none are found, report an error and stop further checking. If the AUDIT_DATE and AUDIT_PERSON_ID fields are missing, an error message is issued but processing continues.

4. If -install or -input_install is given, check for comments in the copy working that are missing the install field. If none are found, report an error and stop further checking. This indicates that the developer forgot to add a history comment when he modified the module.

5. Check for completeness of summary and audit fields in all history comments. If the AUDIT_DATE and AUDIT_PERSON_ID fields are missing, an error message is issued but processing continues. If incomplete or erroneous entries are found, report an error and stop further checking.

6. If -approve or -input_approve is given, place the approve identifier in the working copy's new history comments. If -install or -input_install is given, place the installation identifier in the working copy's new history comments.

7. Reformat the new history comments in the working copy.

8. If no error occurred, replace history comments in the new module with the working comments built by the install operation.

Operation: replace_field, rpf

SYNTAX AS A COMMAND

```
hcom rpf path comment_specs -control_args
```

FUNCTION

replaces existing comment fields in selected history comments.

ARGUMENTS

path
is the name of a source code program that requires history comments. Include the language suffix. You can give an archive pathname.

comment_specs
specify which history comment(s) are to be updated. (See "List of Comment Specifiers" under the add_field operation.)

CONTROL ARGUMENTS FOR FIELD INPUT

Give at least one of the following control arguments:

-approve	-input_summary
-audit	-install
-input_approve	-summary
-input-install	

-approve APPROVE_ID, -apv APPROVE_ID
replaces the APPROVE_ID field. The maximum length of this field is 24 characters. (See "List of History Comment Fields" above for valid APPROVE_IDs.)

-audit, -aud
puts the user's Person_id in the AUDIT_PERSON_ID field.

-fill, -fi
sets fill mode on for the summary field. In fill mode text, words are moved from line to line in such a way that the last word does not extend past the right margin. (Default)

-input_approve, -iapv
prompts for a new APPROVE_ID. This is a single-line field value.

-input_install, -iin
prompts for the INSTALL_ID. This is a single-line field.

-input_summary, -ism
prompts you for a new summary field. This is a multiline field.

-install INSTALL_ID, -in INSTALL_ID
specifies an identifier associated with installing the changed module into execution libraries. (See "List of History Comment Fields" above for a description of valid INSTALL_IDs.) The maximum length of this field is 24 characters.

-no_approve, -napv
does not replace the APPROVE_ID field nor prompts for missing approve fields. (Default)

-no_audit, -naud
    does not add the AUDIT_PERSON_ID field. (Default)

-nofill, -nfi
    sets the fill mode off for the summary field.

-no_install, -nin
    does not set the INSTALL_ID field nor prompts for missing install fields. (Default)

-no_summary, -nsm
    does not replace the summary field. (Default)

-summary TEXT, -sm TEXT
    replaces the text describing the change. If the text contains spaces, quotes, parentheses, etc., enclose it within quotes.

CONTROL ARGUMENTS

-critical_fix, -cfix
    specifies that critical-fix history comments are allowed in the program. All comments following the first that contains critical-fix field values must also contain critical-fix history comments.

-original orig_path, -orig orig_path
    specifies the pathname of an earlier version of the module. You can give an archive pathname and use the equal convention.

-validate RTN, -vdt RTN
    validates user-supplied fields in the history comment, using a user-supplied validation routine. RTN must be a virtual entrypoint name acceptable to cv_entry_. If you give no -vdt, the default is to validate using the hcom_site_validate_ subroutine, if your site has provided it, or the hcom_default_validate_ subroutine provided with hcom.

NOTES

If several history comments are selected, specified fields from the first selected comment are returned or printed, followed by fields from the second selected comment, etc. If the selected field is not present in a given history comment, then a null string is returned for that field. Multiline field values are returned in a single line (with newline characters replaced by a space) as a quoted string.

Name: home_dir, hd

SYNTAX AS A COMMAND

```
hd
```

SYNTAX AS AN ACTIVE FUNCTION

```
[hd]
```

FUNCTION

returns the pathname of your home directory (usually of the form >user_dir_dir>Project_id>Person_id).

Name: hour

SYNTAX AS A COMMAND

```
hour {time_string} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[hour {time_string} {-control_arg}]
```

FUNCTION

returns the one- or two-digit number of an hour of the day, from 0 to 23. The format string to produce this is "^Z9Hd".

ARGUMENTS

time_string
: indicates the hour about which information is desired. If you supply no time_string, the current hour is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-zone STR
: STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default zone. Use the display_time_info command to display a list of all acceptable zone values.

Name: how_many_users, hmu

SYNTAX AS A COMMAND

```
hmu {-control_args} {optional_args}
```

FUNCTION

tells you 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. Use it only together with one of the optional arguments.

-long, -lg
prints additional information including the name of the installation, the time the system was brought up, the time of the next scheduled shutdown, the time of the last shutdown or crash, and load information on absentee users.

LIST OF OPTIONAL ARGUMENTS
list only selected users 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 Person_id and the project name Project_id.

NOTES

In addition to how many users are currently logged in, hmu prints the name of the system, the current load on the system, the maximum load, and, if the absentee facility is running, the number of absentee users and the maximum number of absentee users.

If you invoke this command without any arguments, basic summary information is printed (see "Examples.")

When you select hmu with optional arguments, absentee counts are denoted by an asterisk (*).

You are permitted up to 20 classes of selected users.

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 additional information, 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 Smith, type

```
! hmu -bf Smith
  Smith = 1 + 1*
```

Name: hunt

SYNTAX AS A COMMAND

```
hunt name {path} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[hunt name {path} {-control_args}]
```

FUNCTION

searches a specified subtree of the hierarchy for all occurrences of a named segment that is either freestanding or included in an archive file.

ARGUMENTS

name
is the name of a segment for which hunt is to search. The star convention is allowed.

path
is the pathname of a directory to be interpreted as the root of the subtree in which to search for the specified segment(s). If you don't supply path, the subtree rooted at the current working directory is searched.

CONTROL ARGUMENTS

-all, -a
reports on finding links, directories, and segments.

-archive, -ac
looks inside archives for components whose names match the name argument. (Default)

-first
stops searching as soon as the first occurrence of the selected segment is found. (Default: to return all occurrences)

-no_archive, -nac
does not look inside archives and is therefore faster.

NOTES

This command displays the type of entry found (segment, directory, or link), followed by the entry itself, and a total of the number of occurrences found.

If archive components are being examined, the matching components are reported before added names on the archive segment.

NOTES ON ACTIVE FUNCTION

As an active function, hunt returns a string of pathnames separated by spaces. Archive components are returned as archive_path::component_name.

Name: hunt_dec

SYNTAX AS A COMMAND

```
hunt_dec {path} {-control_args}
```

FUNCTION

searches a specified subtree of the hierarchy for all PL/I object segments that are either freestanding or included in an archive file.

ARGUMENTS

path
: is the pathname of a directory to be interpreted as the root of the subtree in which to search and classify PL/I object segments. If you don't specify path, the working directory is assumed.

CONTROL ARGUMENTS

-aligned_decimal path, -ad path
: creates the ASCII segment listing the absolute pathnames of PL/I object segments and archive segments containing components classified as "aligned decimal" with path suffixed with "hd".

-unaligned_decimal path, -ud path
: creates the ASCII segment listing the absolute pathnames of PL/I object segments and archive segments containing components classified as "unaligned decimal" with path suffixed with "hd".

NOTES

Each PL/I object segment is classified according to its use of arithmetic decimal instructions and how these instructions access the data. The three classes are "no decimal", "aligned decimal", and "unaligned decimal".

This command aids you when PL/I programs compiled using "unaligned decimal" are to be recompiled using the newer PL/I compiler implementing packed decimal, which was part of Multics Release 8.0. This was an incompatible change because the layout of variables containing the unaligned and decimal attributes was changed. Therefore, find those PL/I programs that used "unaligned decimal" so that the appropriate program and data base changes can be made before recompiling the program using the new compiler.

If you specify no control arguments, two ASCII segments are created in the working directory. One segment, aligned_decimal.hd, is a list of the absolute pathnames of PL/I object segments and archive segments containing PL/I object segments classified as "aligned decimal". The absolute pathname of the archive segment is followed by a

space, then by the name of the archive's component classified as "aligned decimal". This occurs for each component of the archive that is classified as such. Another segment, unaligned_decimal.hd, is created in the working directory for the class "unaligned decimal". No segment is created for the class "no decimal".

This command uses the following algorithm to classify PL/I object segments. The text section is scanned for EIS decimal arithmetic instructions generated by the PL/I compiler. If none are found, the object segment is classified as "no decimal". If some are found, they and their descriptors are examined for address modification and nonzero digit offsets. If either is present, the object segment is classified as "unaligned decimal"; otherwise, it is classified as "aligned decimal".

The validity of the classification algorithm rests upon knowledge of how the PL/I compiler generates machine code. Below is a table listing the reliability of the algorithm for the different classifications.

CLASSIFICATION	RELIABILITY
aligned decimal	Always correct.

unaligned decimal	Fails when an unaligned decimal variable falls on a word boundary. For example,
	
	`dcl 1 record aligned,`
	`2 item1 fixed bin(17),`
	`2 item2 fixed dec(3) unaligned;`
	
	The variable, item2, is unaligned decimal. But, since it is located one word from the beginning of the structure, the instruction accessing it appears to be accessing aligned decimal data.

no decimal	If fixed decimal variables are present in the source program but are never referenced or do not have the initial attribute, no EIS fixed decimal instructions are generated by the compiler.

Most of the time hunt_dec identifies correctly PL/I object segments that use unaligned decimal data while letting a few segments slip by misclassified as aligned decimal or no decimal.

This command forces access to all segments in its search path. If unable to access a segment, it bypasses the segment without classifying it.

if

Name: if

SYNTAX AS A COMMAND

```
if [EXPR] -then LINE1 {-else LINE2}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[if [EXPR] -then STR1 {-else STR2}]
```

FUNCTION

conditionally executes one of two command lines depending on the value of an active string. As an active function, returns one of two character strings to the command processor depending on the value of an active string.

ARGUMENTS

EXPR
is the active string, which must evaluate to either "true" or "false".

LINE1
is the command line to execute if EXPR evaluates to "true". If the command line contains any command processor characters, enclose it in quotes.

STR1
is returned as the value of the if active function if the EXPR evaluates to "true".

LINE2
is the command line to execute if EXPR evaluates to "false". If omitted and EXPR is "false", no additional command line is executed. If the command line contains any command processor characters, enclose it in quotes.

STR2
is returned as the value of the if active function if the EXPR evaluates to "false". If omitted and the EXPR is "false", a null string is returned.

Name: immediate_messages, im

SYNTAX AS A COMMAND

```
im {mbx_specification}
```

FUNCTION

restores the immediate printing of interactive messages and notifications.

ARGUMENTS

mbx_specification
specifies the mailbox on which the printing of messages is to be restored. If not given, the user's default mailbox (>udd>Project>Person>Person.mbx) is used.

LIST OF MBX SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table.

STR
is any noncontrol argument and is first interpreted as -mailbox STR; if no mailbox is found, STR is then interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

NOTES

This command cancels defer_messages, but does not cancel any options that may have been specified by accept_messages (see accept_messages and print_messages).

Name: indent, ind

SYNTAX AS A COMMAND

```
ind oldpath {newpath} {-control_args}
```

FUNCTION

improves the readability of a PL/I source segment by indenting it according to a set of standard conventions described below.

ARGUMENTS

oldpath
is the pathname of the input source segment. Source segments with suffixes for PL/I, create_data_segment, and reductions are recognized. If the segment does not have a recognized suffix, indent uses a suffix of .pl1, name.cds, or name.rd, in that order.

newpath
is the pathname of the output source segment. The output segment must have the same suffix as the input segment. If you omit newpath, the indented copy of the program replaces the original one in oldpath. If errors are detected during indentation and you don't give newpath, however, the original copy is not replaced; instead, the pathname of the temporary file containing the indented copy is printed in an error message.

CONTROL ARGUMENTS

-brief, -bf
suppresses warning comments on invalid or non-PL/I characters found outside of a string or comment; such characters are never removed. When you select -bf, those errors whose warning messages are suppressed do not prevent the original copy from being replaced.

-comment STR, -cm STR
sets the comment column to STR. Comments are lined up in this column unless they occur in the beginning of a line or are preceded by a blank line. (Default: 61, if you omit -cm)

-indent STR, -ind STR
sets indentation for each level to STR. Each do, begin, proc, and procedure statement indents additional STR spaces until the matching end statement is encountered. (Default: five, if you omit -ind)

-lmargin STR, -lm STR
sets the left margin (indentation for normal program statements) to STR. (Default: 11, if you omit -lm)

NOTES ON CONVENTIONS

Declaration statements are indented five spaces for dcl declarations and 10 for declare declarations. Identifiers appearing on different lines of the same declare statement are lined up under the first identifier on the first line of the statement. Structure declarations are indented according to level number; after level two, each additional level is indented two additional spaces.

An additional level of indentation is also provided for the then clause of an if statement; else clauses are lined up with the corresponding if. Statements continuing over more than one line have an additional five spaces of indentation for the second line and all succeeding ones.

Multiple spaces are replaced by a single space except within strings or for nonleading spaces and tabs in comments. Trailing spaces and tabs are removed from all lines. Spaces are inserted before left parentheses, after commas, and around the constructs =, ->, <=, >=, and ^=. Spaces are deleted if they are found after a left parenthesis or before a right parenthesis. Tabs are used wherever possible to conserve storage in the output segment.

Parentheses are counted and balanced at every semicolon. If they do not balance or if the input segment ends in a string or comment, a warning message is printed. Language keywords (do, begin, end, etc.) are recognized only at parenthesis level zero, and most keywords are recognized only if they appear to begin a statement.

This command treats comments that begin with /****^ as unindentable. These comments are copied directly into the indented source program without reformatting or indentation. This follows the format_pl1 command convention for identifying comments that are not to be reformatted.

NOTES ON RESTRICTIONS

The only case in which indent splits a line is when lines are longer than 350 characters, since they overflow indent's buffer size.

Labeled end statements do not close multiple open do statements.

This command assumes that the identifiers begin, end, procedure, proc, declare, and dcl are reserved words when they appear at the beginning of a statement. If the input contains a statement like

```
do = do + 1;
```

indent interprets it to mean that the statement delimits a do group and does not indent correctly.

Structure level numbers greater than 99 do not indent correctly.

Name: index

SYNTAX AS A COMMAND

```
index STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[index STRA STRB]
```

FUNCTION

returns an integer representing the character position in STRA where STRB begins. If STRB does not occur in STRA, 0 is returned.

EXAMPLES

```
! string [index abcdefhgij ef]
  5
! string [index "Now is the time" hte]
  0
```

Name: index_set

SYNTAX AS A COMMAND

```
index_set {F1} B1 {I1}...Fn Bn In
```

SYNTAX AS AN ACTIVE FUNCTION

```
[index_set {F1} B1 {I1}...Fn Bn In]
```

FUNCTION

returns one or more integers, separated from each other by spaces.

ARGUMENTS

F
: is the first number of a set and must be an integer. This argument is optional (see "Examples").

B
: is a bound on the set and must be an integer.

I

is the increment between the numbers of a set, either a positive or negative integer. If F > B, then I is assumed to be a negative integer. Otherwise, I is assumed to be positive. This argument is optional.

NOTES

If more than one F-B-I triple is specified, F, B, and I must be specified in each triple. If only one F-B-I triple is specified, I or both I and F can be omitted. I and F are assumed to be 1 if omitted.

EXAMPLES

The following interactions illustrate the index_set active function:

```
! string [index_set 6]
  1 2 3 4 5 6
! string [index_set 5 21 3]
  5 8 11 14 17 20
! string [index_set 0 6 2]
  0 2 4 6
! string [index_set 4 0]
  4 3 2 1 0
! string [index_set 0]
  1 0
! create file_([picture 99 [index_set 5 21 3]])

! list file_*

  Segments = 6, Lengths = 0.

  r w    0  file_20
  r w    0  file_17
  r w    0  file_14
  r w    0  file_11
  r w    0  file_08
  r w    0  file_05
```

The following interactions illustrate command usage:

```
! index_set 4 20 5
  4 9 14 19
! index_set 4 20 5 8 30 6
  4 9 14 19 8 14 20 26
! index_set 5
  1 2 3 4 5
! index_set 5 2
  5 4 3 2
```

Name: initiate, in

SYNTAX AS A COMMAND

```
in path {ref_names} {-control_args}
```

FUNCTION

initiates segments or multisegment files (MSFs).

ARGUMENTS

path
is the pathname of a segment or MSF. You can't use the star convention.

ref_names
are optional reference names by which to initiate the file. If you specify no ref_names, the file is initiated by the entryname portion of path.

CONTROL ARGUMENTS

-all, -a
initiates the file by all its names.

-brief, -bf
does not print a message giving the segment number. (Default)

-chase
used with -a on a link pathname, initiates the target file by all the names on the target segment. (Default)

-force, -fc
terminates each reference name first if it is already known.

-long, -lg
prints a message giving the segment number assigned.

-no_chase
used with -a on a link pathname, initiates the target file by all the names on the link.

-no_force, -nfc
prints an error message if a ref_name is already known. (Default)

ACCESS REQUIRED

Nonnull.

NOTES

When you use initiate to explicitly make known a segment by some reference name, the first reference to that name accesses the initiated segment instead of searching among the search directories for a segment by that name. (For a discussion of search rules, see the Programmer's Reference Manual.)

If you give no ref_names, the segment is made known by the entryname part of the pathname; if you give ref_names, the entryname of the segment is not initiated, but the specified reference names are. If the pathname is a single-element name, the directory assumed is your working directory.

Initiating a MSF involves initiating component 0 of the MSF with the reference names specified.

This page intentionally left blank.

If you cannot initiate a ref_name, an error message is printed and the command continues initiating the other names.

To make a segment known, you must have nonnull access to that segment.

EXAMPLES

The command line

```
! in >udd>Demo>JKeats>gamma x y
```

makes the segment >udd>Demo>JKeats>gamma known, initiating the names x and y.

The command line

```
! in pop
```

makes the segment pop in your working directory known and initiates it with the reference name pop.

The command line

```
! in xx u v -long
```

makes the segment xx in your working directory known, initiates the reference names u and v, and prints out the assigned segment number.

Name: io_call, io

SYNTAX AS A COMMAND

```
io operation switchname {args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io operation switchname {args}]
```

FUNCTION

performs diverse operations on specified I/O switches and returns a result.

ARGUMENTS

operation
: designates the operation to be performed. See "List of Operations" below for a description of each operation, its command syntax line, and specific application.

switchname
is the name of the I/O switch through which the operation is performed.

args
can be one or more arguments, depending on the particular operation to be performed.

LIST OF OPERATIONS

Unless otherwise specified, if a control block for the I/O switch does not already exist, an error message is printed on error_output and the operation is not performed. If the requested operation is not supported for the switch's attachment and/or opening, an error message is printed on error_output.

Differences between command and active function invocation are described under the individual operations.

The explanations of the operations cover only the main points of interest and, in general, treat only the cases where the I/O switch is attached to a file or device. For details see the descriptions of the iox_ subroutine and the I/O modules in the Subroutines manual, and the section on I/O facilities in the Programmer's Reference manual.

Operation: attach

SYNTAX AS A COMMAND

```
io attach switchname attach_description
```

FUNCTION

attaches the I/O switch using the designated I/O module. If a control block for the I/O switch does not already exist, one is created.

ARGUMENTS

attach_description
is the concatenation of modulename and args separated by blanks. It must conform to the requirements of the I/O module. If the I/O modulename is specified by a pathname, it is initiated with a reference name equal to the entryname. If the entryname or reference name does not contain a dollar sign, the attachment is made by calling modulename$modulenameattach. If you supply a $, the entry point specified is called. (See "Entry Point Names" in the Programmer's Reference manual.)

Operation: attach_desc

SYNTAX AS A COMMAND

```
io attach_desc switchname
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io attach_desc switchname {-control_arg}]
```

FUNCTION

prints or returns the attach description of the switch, quoted unless you give -no_quote.

CONTROL ARGUMENTS

-no_quote, -nq
does not enclose the returned data in quotes.

Operation: attached

SYNTAX AS A COMMAND

```
io attached switchname
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io attached switchname]
```

FUNCTION

prints or returns true if the switch is attached, false otherwise.

Operation: close

SYNTAX AS A COMMAND

```
io close switchname
```

FUNCTION

closes the I/O switch.

Operation: close_file

SYNTAX AS A COMMAND

```
io close_file switchname {args}
```

FUNCTION

closes the I/O switch with the specified description. The close_file description is the concatenation of all arguments separated by blanks. It must conform to the requirements of the I/O module.

ARGUMENTS

args
: can be one or more arguments, depending on what is permitted by the particular I/O module.

Operation: closed

SYNTAX AS A COMMAND

```
io closed switchname
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io closed switchname]
```

FUNCTION

prints or returns true if the switch is closed, false otherwise.

Operation: control

SYNTAX AS A COMMAND

```
io control switchname order {args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io control switchname order {args}]
```

FUNCTION

applies only when the I/O switch is attached via an I/O module that supports the control I/O operation. The exact format of the command line depends on the order being issued and the I/O module being used. For more details, see "Control Operations from Command Level" in the appropriate I/O module. If the I/O module supports the control operation and the paragraph just referenced does not appear, assume that only control orders that do not require an info_structure can be performed with the io_call command. This is true because this command/active function uses a null info_ptr. (See the iox_$control entry point in the Subroutines manual and "Performing Control Operations from Command Level" and the I/O module description in the Programmer's Reference Manual.)

The active function returns a value that depends on the I/O module and the order specified.

ARGUMENTS

order
: is one of the orders accepted by the I/O module used in the attachment of the I/O switch.

args
: are additional arguments dependent upon the order being issued and the I/O module being used.

Operation: delete_record, delete

SYNTAX AS A COMMAND

```
io delete switchname
```

FUNCTION

deletes the current record in the file to which the I/O switch is attached. The current record is determined as in rewrite_record.

Operation: destroy_iocb

SYNTAX AS A COMMAND

```
io destroy_iocb switchname
```

FUNCTION

destroys the I/O switch by deleting its control block. Be sure the switch is detached before using this command. Any pointers to the I/O switch become invalid.

Operation: detach

SYNTAX AS A COMMAND

```
io detach switchname {args}
```

FUNCTION

detaches the I/O switch with the specified description. The detach description is the concatenation of all arguments separated by blanks. It must conform to the requirements of the I/O module.

ARGUMENTS

args
: can be one or more arguments, depending on what is permitted by the particular I/O module.

NOTES

If there are no arguments after switchname, this request is synonymous with the detach_iocb request. This means that if you supply no detach description on the command line, detach acts essentially as a short name for detach_iocb.

Operation: detach_iocb

SYNTAX AS A COMMAND

```
io detach_iocb switchname
```

FUNCTION

detaches the I/O switch.

Operation: detached

SYNTAX AS A COMMAND

```
io detached switchname
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io detached switchname]
```

FUNCTION

prints or returns true if the switch is detached, false otherwise.

Operation: find_iocb

SYNTAX AS A COMMAND

```
io find_iocb switchname
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io find_iocb switchname]
```

FUNCTION

prints or returns the location of the control block for the I/O switch. If it does not already exist, the control block is created.

Operation: get_chars

SYNTAX AS A COMMAND

```
io get_chars switchname {N} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io get_chars switchname {N} {-control_args}]
```

FUNCTION

reads the next N characters from the file or device to which the I/O switch is attached.

ARGUMENTS

N
: is a decimal number greater than zero specifying the number of characters to be read.

CONTROL ARGUMENTS

-allow_newline, -alnl
does not add to, nor delete from, the end of the line any newline character. (Default, when you select -segment)

-append_newline, -apnl
adds a newline character to the end of the line if one is not present. (Default, when you don't choose -segment)

-lines
specifies that the offset, if given, is measured in lines rather than in characters. This control argument has meaning only if you also supply -segment; you can't use it with the active function.

-no_quote, -nq
returns the data unquoted. (Default for active function only)

-remove_newline, -rmnl
deletes the newline character, if present, from the end of the line. (Default for active function)

-segment path {offset}, -sm path {offset}
specifies that the data read from the I/O switch is to be stored in the segment given by path. You can optionally describe the location at which to begin writing in path with the offset parameter. This is normally specified as a character offset (i.e., the number of characters to skip over before storing the new data in the segment). For example, an offset of 0 causes the new data to overwrite the entire file. When you also give -lines, then offset is a line offset (i.e., the number of lines to skip over before storing the new data in the segment). For example, an offset of 1 line begins storing data at the second line of the file. If you omit offset, new data is appended to the end of the segment. You can't use this control argument with the active function.

NOTES

The characters read are written on user_output if you specify no -segment or stored in a segment if you specify -segment.

The active function returns the data read as a quoted string, unless you specify -no_quote. A trailing newline character is deleted. If you don't specify the maximum number of characters N, the maximum segment size is assumed.

Operation: get_line

SYNTAX AS A COMMAND

```
io get_line switchname {N} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io get_line switchname {N} {-control_args}]
```

FUNCTION

reads the next line from the file or device to which the I/O switch is attached.

ARGUMENTS

N
is a decimal number greater than zero specifying the maximum number of characters to be read.

CONTROL ARGUMENTS

-allow_newline, -alnl
does not add to, nor delete from, the end of the line any newline character. (Default, when you select -segment)

-append_newline, -apnl
adds a newline character to the end of the line if one is not present. (Default, when you choose no -segment)

-lines
specifies that the offset, if given, is measured in lines rather than in characters. This control argument has meaning only if you also supply -segment; you can't use it with the active function.

-no_quote, -nq
returns the data unquoted. (Default for active function only)

-remove_newline, -rmnl
deletes the newline character, if present, from the end of the line. (Default for active function)

-segment path {offset}, -sm path {offset}
specifies that the data read from the I/O switch is to be stored in the segment given by path. You can optionally describe the location at which to begin writing in path with the offset parameter. This is normally specified as a character offset (i.e., the number of characters to skip over before storing the new data in the segment). For example, an offset of 0 causes the new data to overwrite the entire file. When you also give -lines, then offset is a line offset (i.e., the number of lines to skip over before storing the new data in the segment). For example, an offset of 1 line begins storing data at the second line of the file. If you omit offset, new data is appended to the end of the segment. You can't use this control argument with the active function.

NOTES

If you give N and the line is longer than N, then only the first N characters are read. The active function returns the data read as a quoted string, unless you give -no_quote. A trailing newline character is deleted. If you don't give the maximum number of characters N, the maximum segment size is assumed.

If you select no -segment, the line read is written onto the I/O switch user_output, with a newline character appended if one is not present and if you have selected neither -alnl nor -rmnl. If you select -segment, the line is stored in the segment specified by path; if this segment does not exist, it is created. The bit count of the segment is always updated to a point beyond the newly added data. If the segment contains a trailing newline and you haven't selected -rmnl, that newline remains; if the segment does not contain a trailing newline and you haven't selected -apnl, no newline is appended.

Operation: io_module

SYNTAX AS A COMMAND

```
io io_module switchname
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io io_module switchname]
```

FUNCTION

prints or returns the name of the I/O module through which the switch is attached.

Operation: look_iocb

SYNTAX AS A COMMAND

```
io look_iocb switchname
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io look_iocb switchname]
```

FUNCTION

prints, on user_output, the location of the control block for the I/O switch; if this switch does not exist, an error is printed. The active function returns true if the specified iocb exists, false otherwise.

This page intentionally left blank.

Operation: modes

SYNTAX AS A COMMAND

```
io modes switchname {string} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io modes switchname {string}]
```

FUNCTION

sets only new modes specified in string and then prints the old modes on user_output; applies only when the I/O switch is attached via an I/O module that supports modes. The active function performs the specified modes operation and returns the old modes.

ARGUMENTS

string
is a sequence of modes separated by commas. The string must not contain blanks. See the description of the I/O module attached to the switch for a list of acceptable modes.

CONTROL ARGUMENTS

-brief, -bf
suppresses printing of the old modes.

NOTES

If the switch name is user_i/o, the command refers to the modes controlling your terminal.

Operation: move_attach

SYNTAX AS A COMMAND

```
io move_attach switchname switchname2
```

FUNCTION

moves the attachment of the first I/O switch (switchname) to the second I/O switch (switchname2). The original I/O switch is left detached.

ARGUMENTS

switchname2
is the name of a second I/O switch.

Operation: open

SYNTAX AS A COMMAND

```
io open switchname mode
```

FUNCTION

opens the I/O switch with the specified opening mode.

ARGUMENTS

mode
is one of the following opening modes:

direct_input, di	sequential_output, sqo
direct_output, do	sequential_input_output, sqio
direct_update, du	sequential_update, squ
keyed_sequential_input, ksqi	stream_input, si
keyed_sequential_output, ksqo	stream_output, so
keyed_sequential_update, ksqu	stream_input_output, sio
sequential_input, sqi	

Operation: open_desc

SYNTAX AS A COMMAND

```
io open_desc switchname
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io open_desc switchname]
```

FUNCTION

prints or returns the current open description (stream_input, etc.), quoted.

Operation: open_file

SYNTAX AS A COMMAND

```
io open_file switchname mode {args}
```

FUNCTION

opens the I/O switch with the specified opening mode and description. The open_file description is the concatenation of all arguments separated by blanks. It must conform to the requirements of the I/O module.

ARGUMENTS

mode
is one of the opening modes listed under open.

args
can be one or more arguments, depending on what is permitted by the particular I/O module.

Operation: opened

SYNTAX AS A COMMAND

```
io opened switchname
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io opened switchname]
```

FUNCTION

prints or returns true if the switch is open, false otherwise.

Operation: position

SYNTAX AS A COMMAND

```
io position switchname type
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io position switchname type]
```

FUNCTION

positions the file to which the I/O switch is attached.

ARGUMENTS

type
can be one of the following:

bof
sets position to beginning of file.

eof
sets position to end of file.

forward N, fwd N, f N
sets position forward N records or lines (same as reverse -N).

reverse N, rev N, r N
sets position back N records (same as forward -N records). You can give any other numeric argument or pair of numeric arguments, but its function depends on the I/O module being used and cannot be implemented for all I/O modules.

reverse N, rev N, r N

NOTES

If type is bof, the file is positioned to its beginning, so that the next record is the first record (structured files) or the next byte is the first byte (unstructured files). If type is eof, the file is positioned to its end; the next record (or next byte) is at the end-of-file position. If type is forward or reverse, the file is positioned forwards or backwards over records (structured files) or lines (unstructured files). The number of records or lines skipped is determined by the absolute value of N. The active function returns true if it succeed, false otherwise.

In the case of unstructured files, the next-byte position after the operation is at a byte immediately following a newline character (or at the first byte in the file or at the end of the file). The number of newline characters moved over is the absolute value of N.

If the I/O switch is attached to a device, you are only allowed forward skips; this discards the next N lines input from the device.

Operation: print_iocb

SYNTAX AS A COMMAND

```
io print_iocb switchname
```

FUNCTION

prints, on user_output, all the data in the control block for the I/O switch, including all pointers and entry variables.

Operation: put_chars

SYNTAX AS A COMMAND

```
io put_chars switchname {string} {-control_args}
```

FUNCTION

outputs a character string or an entire segment to a specified I/O switch.

ARGUMENTS

string
can be any character string.

CONTROL ARGUMENTS

-allow_newline, -alnl
does not add to, nor delete from, the end of the line any newline character.

-append_newline, -apnl
adds a newline character to the end of the line if one is not present. (Default)

-lines
specifies that the offset and length operands of -segment are measured in lines rather than in characters. This control argument has meaning only if you also supply -segment.

-remove_newline, -rmnl
does not append a newline character to the end of the output string or segment even if one is not present at the end.

-segment path {{offset} length}, -sm path {{offset} length}
specifies that the data for the output operation is to be found in the segment specified by path. You can optionally describe the location and length of the data with offset and length parameters. These are normally specified as a character offset (i.e., 0 identifies the first character of the segment) and character length. When you also give -lines, they are specified as a line offset and line count. If you give no offset, 0 is assumed. If you give no length and offset, the entire segment is used.

-string STR, -str STR
specifies an output string that can have a leading hyphen.

NOTES

The string argument and -segment are mutually exclusive. If you supply a string, the contents of the string are output to the I/O switch. If you supply -segment, the data is taken from the segment specified by path, at the offset and length given.

If the I/O switch is attached to a device, io_call transmits the characters from the string or the segment to the device. If the I/O switch is attached to an unstructured file, the data is added to the end of the file.

Operation: read_key

SYNTAX AS A COMMAND

```
io read_key switchname
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io read_key switchname {-control_arg}]
```

FUNCTION

prints, on user_output, the key and record length of the next record in the indexed file to which the I/O switch is attached. The file's position is not changed. The active function returns the value of the key, quoted, unless you select -no_quote.

CONTROL ARGUMENTS

-no_quote, -nq
: does not enclose the returned data in quotes. Data containing spaces is quoted by default.

Operation: read_length

SYNTAX AS A COMMAND

```
io read_length switchname
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io read_length switchname]
```

FUNCTION

prints, on user_output, the length of the next record in the structured file to which the I/O switch is attached. The file's position is not changed. The active function returns the length of the next record, in bytes.

Operation: read_record, read

SYNTAX AS A COMMAND

```
io read switchname {N} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io read switchname {N} {-control_args}]
```

FUNCTION

reads the next record from the file to which the I/O switch is attached into a buffer of length N.

ARGUMENTS

N
is a decimal integer greater than zero specifying the size of the buffer to be be used.

CONTROL ARGUMENTS

-allow_newline, -alnl
does not add to, nor delete from, the end of the line any newline character. (Default for command)

-append_newline, -apnl
adds a newline character to the end of the line if one is not present.

-lines
specifies that the offset, if given, is measured in lines rather than in characters. This control argument has meaning only if you also supply -segment; you can't use it with the active function.

-no_quote, -nq
returns the data unquoted. (Default for active function only)

-remove_newline, -rmnl
deletes the newline character, if present, from the end of the line. (Default for active function)

-segment path {offset}, -sm path {offset}
specifies that the data read from the I/O switch is to be stored in the segment given by path. You can optionally describe the location at which to begin writing in path with the offset parameter. This is normally specified as a character offset (i.e., the number of characters to skip over before storing the new data in the segment). For example, an offset of 0 causes the new data to overwrite the entire file. When you also give -lines, then offset is a line offset (i.e., the number of lines to skip over before storing the new data in the segment). For example, an offset of 1 line begins storing data at the second line of the file. If you omit offset, new data is appended to the end of the segment. You can't use this control argument with the active function.

NOTES

The characters read are written on user_output if you specify no -segment or stored in a segment if you specify -segment.

The active function returns the data read as a quoted string, unless you give -no_quote. A trailing newline character is deleted. If you don't give the maximum number of characters N, the maximum segment size is assumed.

Operation: rewrite_record, rewrite

SYNTAX AS A COMMAND

```
io rewrite switchname {string} {-control_args}
```

FUNCTION

replaces the current record in the file to which the I/O switch is attached.

ARGUMENTS

string
is any character string.

CONTROL ARGUMENTS

-allow_newline, -alnl
does not add to, nor delete from, the end of the line any newline character. (Default, when you select -segment)

-append_newline, -apnl
adds a newline character to the end of the line if one is not present.

-lines
specifies that the offset and length operands of -segment are measured in lines rather than in characters. This control argument has meaning only if you also supply -segment.

-no_quote, -nq
returns the data unquoted. (Active function usage only)

-remove_newline, -rmnl
deletes the newline character, if present, from the end of the line. (Default if you give no -segment)

-segment path {{offset} length}, -sm path {{offset} length}
specifies that the data for the output operation is to be found in the segment specified by path. You can optionally describe the location and length of the data with offset and length parameters. These are normally specified as a character offset (i.e., 0 identifies the first character of the segment) and character length. When you also give -lines, they are specified as a line offset and line count. If you give no offset, 0 is assumed. If you give no length and offset, the entire segment is used.

-string STR, -str STR
specifies an output string that can have a leading hyphen.

NOTES

The string argument and -segment are mutually exclusive. If you supply a string, the contents of the string are output to the I/O switch. If you supply -segment, the data is taken from the segment specified by path, at the offset and length given.

The current record must have been defined by a preceding read_record, seek_key, or position operation as follows:

read_record
the current record is the last record read.

seek_key
the current record is the record with the specified key.

position
the current record is the one preceding the record to which the file was positioned.

Operation: seek_key

SYNTAX AS A COMMAND

```
io seek_key switchname key
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io seek_key switchname key]
```

FUNCTION

positions the indexed file to which the I/O switch is attached to the record with the given key. The record's length is printed on user_output. Trailing blanks in the key are ignored. The active function returns true if the key exists, false otherwise.

ARGUMENTS

key
: is a string of no more than 256 ASCII characters. You can use the null string ("") as a key.

NOTES

If the file does not contain a record with the specified key, it becomes the key for insertion. A following write_record operation adds a record with this key.

Operation: test_mode

SYNTAX AS A COMMAND

```
io test_mode switchname mode
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io test_mode switchname mode]
```

FUNCTION

performs a modes operation and prints or returns true if mode appears in the mode string, false if ^mode appears.

Operation: valid_mode

SYNTAX AS A COMMAND

```
io valid_mode switchname mode
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io valid_mode switchname mode]
```

FUNCTION

performs a modes operation and prints or returns true if either mode or ^mode appears in the mode string, false otherwise.

Operation: valid_op

SYNTAX AS A COMMAND

```
io valid_op switchname operation
```

SYNTAX AS AN ACTIVE FUNCTION

```
[io valid_op switchname operation]
```

FUNCTION

prints or returns true if the operation is defined on the switch.

List of Operations

close	move_attach
control	open
delete_record	position
destroy_iocb	put_chars
detach_iocb	read_key
find_iocb	read_length
get_chars	read_record
get_line	rewrite_record
look_iocb	seek_key
modes	write_record

Operation: write_record, write

SYNTAX AS A COMMAND

```
io write switchname {string} {-control_args}
```

FUNCTION

adds a record to the file to which the I/O switch is attached.

ARGUMENTS

string
is any character string.

CONTROL ARGUMENTS

-allow_newline, -alnl
does not add to, nor delete from, the end of the line any newline character. (Default when you select -segment)

-append_newline, -apnl
adds a newline character to the end of the line if one is not present.

-lines
specifies that the offset and length operands of -segment are measured in lines rather than in characters. This control argument has meaning only if you also supply -segment.

-no_quote, -nq
returns the data unquoted. (Active function usage only)

-remove_newline, -rmnl
deletes the newline character, if present, from the end of the line. (Default if you give no -segment)

-segment path {{offset} length}, -sm path {{offset} length}
specifies that the data for the output operation is to be found in the segment specified by path. You can optionally describe the location and length of the data with offset and length parameters. These are normally specified as a character offset (i.e., 0 identifies the first character of the segment) and character length. When you also give -lines, they are specified as a line offset and line count. If you give no offset, 0 is assumed. If you give no length and offset, the entire segment is used.

-string STR, -str STR
specifies an output string that can have a leading hyphen.

NOTES

The string argument and -segment are mutually exclusive. If you supply a string, the contents of the string are output to the I/O switch. If you supply -segment, the data is taken from the segment specified by path, at the offset and length given.

If the file is sequential, the record is added at the end of the file. If the file is indexed, the record's key must have been defined by a preceding seek_key operation.

is_component_pathname

Name: is_component_pathname, icpn

SYNTAX AS A COMMAND

```
icpn path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[icpn path]
```

FUNCTION

returns true if the path is a valid pathname that refers to an archive component (pathname).

This page intentionally left blank.

EXAMPLES

```
! icpn >udd>Proj>Myname>start_up.ec
  false

! icpn >udd>Multics>Library>Source>bound_command_demos_.s::program.pl1
  true
```

Name: kermit

SYNTAX AS A COMMAND

```
kermit {-control_args}
```

FUNCTION

invokes the Multics implementation of the Kermit file transfer program. The Multics Kermit program provides the capability to transfer files between a Multics system and a remotely located system (e.g., a personal computer) using the KERMIT protocol. Once invoked, Multics Kermit prompts you for the various file transfer requests. Multics Kermit has been implemented with a server feature that permits you to login to Multics from a remote site and specify file transfer operations without having to escape back and forth between the Multics system and the remote system.

CONTROL ARGUMENTS

-abbrev, -ab
: enables abbreviation expansion of request lines.

-io_switch STR, -iosw STR
: specifies that communication with the remote system be done over the I/O switch whose name is STR. If you omit it, the user_i/o switch is assumed.

-no_abbrev, -nab
: does not enable abbreviation expansion of request lines. (Default)

-no_prompt, -npmt
: suppresses the prompt for request lines in the request loop.

-no_request_loop, -nrql
: does not enter the request loop after performing any operations given by -request.

-no_start_up, -nsu, -ns
: does not execute the start_up exec_com.

-profile PATH, -pf PATH
specifies the pathname of the profile to use for abbreviation expansion. The suffix "profile" is added to PATH if you don't include it explicitly on the command line. This control argument implies -abbrev.

-prompt STR, -pmt STR
sets the request loop prompt to STR. (Default: ^/Multics-Kermit^[(^d)^]:^2x)

-quit
exits the Kermit program after performing any operations given by -request.

-request STR, -rq STR
executes STR as a Kermit program request line before entering the request loop.

-request_loop, -rql
enters the Kermit program request loop after performing any operations given by -request. (Default)

-start_up, -su
executes the Kermit program start_up exec_com start_up.kermit. The users home directory, the project directory, and >site are searched, in that order, for the start_up. The exec_com is executed before the request_string and before entering the subsystem request_loop. (Default)

LIST OF REQUESTS

The following is a summary of requests used to respond to prompts from the Kermit program. In this summary "-ca" is used as shorthand for "-control_args". For a complete description of any request, issue the Kermit request:

help request_name

prints a line describing the current invocation of the Kermit program.

?
prints a list of requests available in the Kermit program.

abbrev {-ca}, ab {-ca}
controls abbreviation processing of request lines.

do rq_str {args}, [do rq_str args]
executes/returns a request line with argument substitution.

exec_com ec_path {ec_args}, ec ec_path {ec_args}
[exec_com ec_path {ec_args}], [ec ec_path {ec_args}]
executes a file of Kermit program requests that can return a value.

execute cmd_line, e cmd_line
[execute active_str], [e active_str]
executes a Multics command line or evaluates a Multics active string.

finish
sends a request to a remote server to shut down server operation and return the remote system to its request's loop.

get remote_source_path {local_destination_path}
sends a request to a remote server requesting that the named file(s) be sent from the remote system.

help {topics} {-ca}
prints information about Kermit program requests and other topics. If you supply no topics, methods for getting help are listed.

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

list_requests {STRs} {-ca}, lr {STRs} {-ca}
prints a brief description of selected Kermit program requests. You can use STR to specify that only specific requests be listed.

log {PATH} {-ca}
directs the Kermit program to start logging transactions.

logout
sends a request to the remote server directing it to log you out from the remote system.

quit, q
exits the Kermit program.

quit_log
directs the Kermit program to stop logging transactions.

receive {PATH}, r {PATH}
receives a file or file group from the other system.

send local_source_path {remote_destination_path}
s local_source_path {remote_destination_path}
sends a file or file group to the other system.

server
instructs the Kermit program to cease taking commands from the keyboard and to receive all further instruction in the form of Kermit packets.

set mode {STR}
establishes or modifies various modes for file transfers.

show {modes}
displays mode values.

statistics, st
shows statistics about the most recent file transfer.

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

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

The following list of modes are recognized by the Kermit program and the set and show Kermit requests. The values associated with each mode are also given.

LIST OF MODES AFFECTING FILE STORAGE

file_type STR
indicates the type of file being transferred. STR can be either binary or ascii.

file_warning STR
indicates the action to be taken when an incoming file name has the same name as an existing file name in the default directory when receiving files. STR can be either on or off. If file_warning is on, the Kermit program renames the file to avoid overwriting the preexisting one; if file_warning is off, the incoming file replaces the preexisting one. If logging transactions, the log indicates the name of the file in which the data was stored. (Default: on)

incomplete STR
indicates the action to be taken if a file was not completely transferred. STR can be either keep or discard. If you specify keep, all incomplete files are saved; if you give discard, incomplete files are discarded. (Default: keep)

LIST OF MODES AFFECTING FILE TRANSFER

control_prefix CHR, cp CHR
is the character to use for quoting of control characters, where CHR is any character in the range ! through > or ` through ~, but different from eight_bit_prefix and repeat_prefix. (Default: #)

eight_bit_prefix CHR, ebp CHR
is the ASCII character Multics Kermit program uses, when transmitting binary files via a 7-bit connection, to quote characters that have the 8th bit set. CHR is one of the following, but different from control_prefix and repeat_prefix:

Y
characters with the 8th bit set are quoted if the remote system requests it.

N
characters with the 8th bit set are not quoted.

&
or any character in the range ! through > or ` through ~. Use the specified character for quoting characters with the 8th bit set. If the Multics Kermit program's eight_bit_prefix character is different from the remote program's, then no 8th bit prefixing is done.

The value of this mode is ignored if line_byte_size is 8. (Default: &)

end_of_packet CHR, eop CHR
is the character the Multics Kermit program uses as a line terminator for incoming packets, where CHR is an ASCII character between SOH (001 octal) and US (037 octal) inclusive and different from the start_of_packet character. (Default: carriage return, 015 octal)

line_byte_size N
indicates whether data is being transmitted via a 7-bit or 8-bit connection, where N can be either 7 or 8. A 7-bit connection is desirable when transferring ASCII files or when the 8th bit of each transmitted byte is required for parity or changed by intervening communications equipment. Use an 8-bit connection for transferring binary files, as it decreases protocol overhead. If you can't use an 8-bit connection for a binary file transfer, then you can use a 7-bit connection with the eight_bit_prefix mode enabled to transfer binary files. (Default: 7)

packet_length N, pl N
is the maximum packet length the Multics Kermit program can receive, where N is an integer between 20 and 94 (decimal). (Default: 80)

parity STR
used for communicating with systems or networks that require the 8th bit for character parity. The parity used must be the same for Kermit programs on both the local and remote system. STR can be one of

none
eight data bits and no parity.

mark
seven data bits with the parity bit set to 1.

space
seven data bits with the parity bit set to 0.

even
seven data bits with the parity bit set to make the overall parity even.

odd
seven data bits with the parity bit set to 1 to make the overall parity odd.

The value of the mode is ignored if line_byte_size is 8. (Default: none)

repeat_prefix CHR, rp CHR
is the character the Multics Kermit program uses to indicate a repeated character, where CHR can be any character in the range ! through > or ' through ~, but different from the control_prefix and eight_bit_prefix. Space " " denotes no repeat count processing is to be done. If the Multics Kermit program repeat_prefix character is different from the remote system's, then no repeat prefixing is done. (Default: ~).

retry_threshold N, rt N
specifies how many times to try sending or receiving a particular packet before giving up, where N is an integer between 5 and 15 inclusive. (Default: 5)

start_of_packet CHR, sop CHR
is the character to be used for the start of packet, where CHR is an ASCII character between NUL (000 octal) and US (037 octal) inclusive. The start_of_packet character must be the same for Kermit programs on both the local and remote system, but different from the end_of_packet character. (Default: SOH, octal 001)

timeout N
specifies how many seconds the Multics Kermit program wants the remote system to wait for a packet from Multics before trying again, where N is an integer value between 5 and 20. (Default: 15)

NOTES ON KERMIT DEVELOPMENT

The KERMIT protocol was developed at Columbia University. Many implementations of KERMIT are avaiable from the KERMIT group at Columbia. Direct inquiries about KERMIT to

```
KERMIT Distribution
Columbia University Center for Computing Activities
7th Floor, Watson Laboratory
612 West 115th Street
New York, New York 10025
```

NOTES ON REMOTE SYSTEMS

The Multics Kermit program supports the transfer of 7-bit ASCII files and 8-bit binary files. You can transfer ASCII files between any two systems, whereas you can only transfer binary files between systems that are able to retain the original value of the data byte. For example, sending a Multics binary file in which all bits of the 9-bit byte are used to a system that uses 8-bit bytes results in the loss of the most significant bit (i.e., the transferred file on the remote system differs from the original file sent). However, a binary file received by the Multics Kermit program from a remote system that uses 8-bit bytes can then be sent by Multics Kermit to a second such remote system. The resulting file on the second system is identical to the original file sent.

NOTES ON FILE TRANSFER

For transmission between systems, you must assign files to one of two categories--ASCII or binary. On systems with 8-bit bytes, ASCII files have the high-order bit of each byte set to zero, whereas binary files use the high-order bit of each byte for data, in which case its value can vary from byte to byte and must be preserved. Binary file transmission is permissible as long as the two Kermit programs involved can control the value of the 8th bit (i.e., it is not being used for parity or changed by intervening communications equipment). In that case the 8th bit of a transmitted character matches that of the original data byte without any special 8th bit prefixing. For example, to send or receive a binary file of 8-bit bytes when an 8-bit connection is possible, set line_byte_size to 8, set file_type to binary, and start the transfer. If an 8-bit connection is not possible, then you can send binary files via a 7-bit connection using the eight_bit_prefix. For example, set file_type to binary, set line_byte_size to 7, set parity to str, set eight_bit_prefixing to chr, and start the transfer. To send or receive an ASCII file, set file_type to ascii, set line_byte_size to 7, set any other desired modes, and start the transfer.

The Multics Kermit program does not support the transfer of 9-bit bytes when the most significant bit is used for data. Thus sending a Multics binary file to a second Multics site results in the loss of the most significant bit of each byte.

PROCEDURE FOR USING KERMIT: MULTICS/PERSONAL COMPUTER

Use the following procedure to transfer files between Multics and a personal computer using Kermit:

1. Start Kermit on the personal computer.

2. Set any desired modes.

3. Connect to Multics via the connect command. Once connected, the standard Multics banner is displayed.

4. Login to Multics.

5. Start Kermit on Multics. It responds with the prompt "Multics-Kermit:".

6. Set any desired modes.

7. Execute either a send or receive request, specifying the file or file group.

8. Use the appropriate escape sequence to get back to Kermit command level on the personal computer.

9. Execute the corresponding request on the personal computer. For example, if you issue the send request on Multics, execute the receive request on the personal computer or vice versa.

10. File transfer begins. The personal computer displays the status of the file transfer.

11. To transfer more files, connect back to Multics Kermit and enter a carriage return to get the "Multics-Kermit:" prompt. Go to step 7.

12. Exit Multics Kermit by issuing the quit request and logout.

13. Use the appropriate escape sequence to get back to Kermit command level on the personal computer.

PROCEDURE FOR USING KERMIT: MULTICS SERVER/PERSONAL COMPUTER

Use the following procedure to transfer files between Multics and a personal computer using the Kermit server:

1. Start Kermit on the personal computer.

2. Set any desired modes.

3. Connect to Multics via the connect command. Once connected, the standard Multics banner is displayed.

4. Login to Multics.

5. Start Kermit on Multics. It responds with the prompt "Multics-Kermit:".

6. Set any desired modes.

7. Execute the server request.

8. Use the appropriate escape sequence to get back to Kermit command level on the personal computer.

9. Execute the Kermit server request on the personal computer for sending or receiving files.

10. File transfer begins. The personal computer displays the status of the file transfer.

11. To transfer more files, go to step 9.

12. Exit Multics Kermit by issuing the Kermit server quit request on the personal computer.

13. Connect back to Multics and logout.

14. Use the appropriate escape sequence to get back to Kermit command level on the microcomputer.

PROCEDURE FOR USING KERMIT: MULTICS/MULTICS

Use the following procedure to transfer files between two Multics systems using Kermit:

1. Login to the local Multics.
2. Connect to the remote Multics via the dial_out command.
3. Login to the remote Multics.
4. Start Kermit on the remote Multics.
5. Set any desired modes.
6. Execute the server request.
7. Use the appropriate escape sequence to start up Kermit on the local Multics (e.g., e kermit -iosw [switch_name])
8. Execute the Multics send request to send files to the remote system, or the get request to receive files from the remote system.
9. To transfer more files, go to step 8.
10. When done, execute the finish request to exit the remote server and quit from the local kermit to reconnect to the remote Multics, or execute the logout request to logout from the remote Multics and then quit the local Kermit.

EXAMPLES

```
r 14:24 3.546 25
dial_out e.h024.* p25...

Multics Op. System M...

l NCopernicus Multics
Password:

You are protected from...

r 14:26 14.354 243
kermit -prompt "^/SysM-Kermit^[ (^d)^]:  "

SysM-Kermit: server
 e kermit -iosw [switch_name]

Multics-Kermit: send x.pl1 test.pl1

Sending 1 file(s)...
```

```
Transaction completed: 1 file(s) sent.

Multics-Kermit: get test.pl1 x2.pl1

Receiving...

Successfully received 1 file(s).

Multics-Kermit: logout

Multics-Kermit: q

dial_out: connection closed.
r 14:32 53.659 863
```

Name: l6_ftf

SYNTAX AS A COMMAND

```
l6_ftf channel_name {-control_args}
```

FUNCTION

allows a process to handle file transfer requests from a DPS 6 using the DPS 6 File Transfer Facility (FTF) protocol (referred to as L6 TRAN; see the *DPS 6 & Level 6 to Level 66 File Transmission Facility User's Guide* (CZ60).

ARGUMENTS

channel_name
: is the name of a polled VIP subchannel over which the file transfers take place. It must have the x prefix (i.e., b.h217.x01).

CONTROL ARGUMENTS

-long, -lg
: prints a line describing each file transfer as it starts and as it completes. (Default: not to print this information)

-target_dir -td
: restricts the pathnames of any files to be transferred to be relative to the target directory. You can specify the root as ">", which allows you to give absolute pathnames. (Default: your working directory)

ACCESS REQUIRED

You must have rw access to the access control segment (ACS) of the specified channel name and the dialok attribute turned on in the project master file (PMF). The polled VIP subchannel must have the slave attribute in the channel master file (CMF).

NOTES

This command continues to listen for, and carry out, DPS 6 requests until you explicitly tell it to stop ("quit," "q") or the channel disconnects. You can type the quit request at any time, but it only takes effect before any file transfer has started or between two file transfers. You can only transfer sequential ASCII or sequential binary files to or from the DPS 6. ASCII files on Multics are assumed to be stream files when sending and are stored as stream files when receiving. Binary files on Multics have a special format.

Interrupting and releasing a file transfer may result in aborting the operation inconsistently and hanging the DPS 6 task.

The polled VIP subchannel must be defined with a terminal type that assigns max_message_len to a value of 1009 in its additional_info statement.

Blank lines in a DPS 6 file actually have some characters on them, usually a space or tab. These characters end up in the Multics file. The command transmits blank lines from Multics files to the DPS 6 by sending a line containing a single-space character.

Each sequential binary record on Multics is assumed to have the following format:

```
dcl 1 binary_record aligned based,
      2 num_sextets fixed bin(35) aligned,
      2 sextets (0 refer binary_record.num_sextets) fixed bin(6)
        unsigned unaligned;
```

Each binary record is stored in a vfile_ record of size currentsize(binary_record) * 4.

Name: last_message, lm

SYNTAX AS A COMMAND

```
lm {mbx_specification}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lm {mbx_specification}]
```

FUNCTION

returns the text of the last message received from send_message. See accept_messages, last_message_sender, last_message_time, and send_message.

ARGUMENTS

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

LIST OF MBX SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table.

STR
is any noncontrol argument and is first interpreted as -mailbox STR; if no mailbox is found, STR is then interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

Name: last_message_destination, lmds

SYNTAX AS A COMMAND

```
lmds {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lmds {-control_arg}]
```

FUNCTION

returns the User_id of the last destination to which a message was sent by send_message.

CONTROL ARGUMENTS

-inhibit_error, -ihe
prints/returns a null string if there is no last message destination.

NOTES

See accept_messages, last_message_sender, last_message_time, and send_message.

Name: last_message_sender, lms

SYNTAX AS A COMMAND

```
lms {mbx_specification}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lms {mbx_specification}]
```

FUNCTION

returns the sender of the last message received (from send_message) in the form "Person_id.Project_id" (e.g., GBShaw.Demo).

ARGUMENTS

mbx_specification
specifies the mailbox of the sender of the last message. If not given, the user's default mailbox (>udd>Project>Person>Person.mbx) is used.

LIST OF MBX SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table.

STR
is any noncontrol argument and is first interpreted as -mailbox STR; if no mailbox is found, STR is then interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

NOTES

You are cautioned against using the active function when in polite mode. In this mode the system holds all messages until you finish typing a line (i.e., until the carriage is at the left margin); therefore it is possible that while you are sending a message, your process receive another message from another user--a message not yet seen. By using the active function in this situation, you can inadvertently attribute a message to the wrong person.

See accept_messages, last_message, last_message_time, and send_message.

EXAMPLES

Assume that a user has just received the following message:

```
From AFrance.Demo 11/19/86 1231.7 mst Wed: Need access to xy
```

A reply can be sent as follows:

```
! sm [lms] Sorry for the oversight.  You have access now.
```

Name: last_message_time, lmt

SYNTAX AS A COMMAND

```
lmt {mbx_specification}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lmt {mbx_specification}]
```

FUNCTION

returns the date and time the last message (from send_message) was received.

ARGUMENTS

mbx_specification
specifies the mailbox from which the last message was received. If not given, the user's default mailbox (>udd>Project>Person>Person.mbx) is used.

LIST OF MBX SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table.

STR
is any noncontrol argument and is first interpreted as -mailbox STR; if no mailbox is found, STR is then interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

NOTES

See accept_messages, last_message, last_message_sender, and send_message.

Name: length, ln

SYNTAX AS A COMMAND

```
ln STR
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ln STR]
```

FUNCTION

returns an integer representing the number of characters in STR.

ARGUMENTS

STR
is any string of alphanumeric characters. If STR contains blanks or other command language characters, enclose it in quotes.

EXAMPLES

```
string [ln "A multiple-word string"]
22
```

The following example from an exec_com segment tests for a string that is greater than 27 characters.

```
&if [nless [ln &1] 27] &then &goto OK
&print Entry name too long.  &1.info
&quit
&label OK
ec exec_com2 &1.info
```

Name: less

SYNTAX AS A COMMAND

```
less STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[less STRA STRB]
```

FUNCTION

returns true if strA is less than strB according to the ASCII collating sequence; otherwise it returns false.

NOTES

The strings are compared character by character according to their ASCII code value (i.e., if the first character in each string has the same ASCII code value, compare the second character; if their values are identical, compare the third character; etc.). See the descriptions of nless and ngreater for a way to compare numeric strings.

Name: library_descriptor, lds

SYNTAX AS A COMMAND

```
lds key {arguments}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lds key {arguments}]
```

FUNCTION

prints or returns information about library descriptors and controls their use by the other library descriptor commands (see the It can print the pathname of the directory* or archive associated with a library root, can print detailed information about one or more library roots, can set and print the name of the default library descriptor used by the other library descriptor commands, and can print the default library and search names associated with each library descriptor command.

A library descriptor is a data base that associates directories or archives in the Multics storage system with the roots of a logical library structure.

LIST OF KEYS

name, nm
returns the name of the default library descriptor that is currently being used.

Usage:
```
lds name
[lds name]
```

set
sets the name of the default library descriptor.

Usage:
```
lds set {desc_name}
```

where

desc_name
is the pathname or reference name of the new default library descriptor. If you give a reference name, the descriptor is searched for according to the search rules. If you omit desc_name, the default library descriptor is set as the descriptor for the Multics system libraries.

pathname, pn
returns the pathname of the library root(s) that is identified by one or more library names. Using pathname, you can invoke the command as an active function.

Usage:
```
lds pathname library_names {-control_args}
[lds pathname library_names {-control_args}]
```

where

library_names
are the names of the libraries whose pathnames are to be returned. You can use the star convention to identify a group of libraries, and you can give up to 30 library names.

control_args
can be

-descriptor desc_name
gives the pathname or reference name of the library descriptor defining the library roots whose pathnames are to be returned. If you don't choose -descriptor, the default library descriptor is used.

-library library_name, -lb library_name
identifies a library name that begins with a minus to distinguish the library name from a control argument. There are no other differences between library_names and those given with -library. You can supply one or more -library control arguments.

default, dft
prints the default library name(s) and search name(s) associated with one or more of the library descriptor commands.

Usage:
```
lds default {command_names} {-control_arg}
```

where

command_names
are the names of the library descriptor commands whose default library and search names are to be printed. If you select no command names, the defaults for all the library descriptor commands are printed.

control_arg
can be

-descriptor desc_name
gives the pathname or reference name of the library descriptor defining the library roots whose pathnames are to be returned. It can appear anywhere after the key. If you don't choose -descriptor, the default library descriptor is used.

root, rt
prints detailed information about library roots, including the names on each library root, its pathname, and its type.

Usage: `lds root library_names {-control_args}`

where

library_names
are the names of the libraries whose pathnames are to be returned. You can use the star convention to identify a group of libraries, and you can give up to 30 library names.

control_args
select from the following:

-descriptor desc_name
gives the pathname or reference name of the library descriptor defining the library roots whose pathnames are to be returned. If you don't choose -descriptor, the default library descriptor is used.

-library library_name, -lb library_name
identifies a library name that begins with a minus to distinguish the library name from a control argument. There are no other differences between library_names and those given with -library. You can supply one or more -library control arguments.

-match
prints all library root names that match any of the library names. (Default)

-name, -nm
prints all the names on each library root.

-primary, -pri
prints the primary name on each library root.

Name: library_fetch, lf

SYNTAX AS A COMMAND

```
lf  {search_names} {-control_args}
```

FUNCTION

fetches entries from a library. It functionally replaces get_library_segment. You can fetch segments, archives, archive components, multisegment files (MSFs), and MSF components.

ARGUMENTS

search_names
are star names identifying the library entries to be fetched. Defaults differ for each library descriptor. You can supply up to 1000 search names; if you give none, any default search names specified in the library descriptor are used.

CONTROL ARGUMENTS

-all, -a
includes in the output file complete status information for fetched entries.

-all_matches, -amch
fetches all matching entries. (Default, if you supply more than one search name, any star names, or -component)

-brief, -bf
suppresses printing of information about fetched entries. (Default)

-chase
fetches through links.

-components
fetches library entries contained in, or with, a matching entry; it implies -all_matches.

-container
fetches the library entry containing a matching library entry, rather than the matching entry itself.

-default, -dft
includes in the output file default status information for fetched entries.

-entry, -et
fetches matching library entries only. (Default)

-descriptor desc_name
specifies the descriptor defining the libraries to be searched.

-first_match, -fmch
fetches the first matching entry only. (Default, if you specify only one search name)

-into path
fetches library entries into the specified pathname (absolute or relative). The directory portion of the pathname identifies the directory into which each library entry is copied; the final entryname of the pathname renames each library entryname being placed on the copy. You can give an equal name as the final entryname of the path. Use -into only once in a command line. If not given, matching entries are copied into your working directory and no renaming occurs.

-library library_name, -lb library_name
fetches entries in the specified library. You can use star names. You can supply up to 100 -library control arguments; if you give none, any default library names specified in the library descriptor are used.

-long, -lg
prints information about fetched entries.

-match
puts entrynames that match a search name on the fetched entry. (Default)

-name, -nm
puts all the entrynames on the fetched library entry.

-no_chase
does not fetch through links. (Default)

-omit
does not fetch library entries awaiting deletion. (Default)

-output_file filename, -of filename
appends status of fetched library entries to named file.

-primary, -pri
puts first entryname on fetched library entry.

-retain, -ret
fetches library entries awaiting deletion.

-search_name search_name
gives search_name that looks like a control argument. You can supply one or more -search_name control arguments.

NOTES

This command uses a library descriptor and library search procedures, as described in the *Multics Library Maintenance SDN*, (Order No. AN80). The initial default descriptor describes the Multics system libraries and allows library_fetch to extract source programs, object segments and bind files, include files and info segments, and compilation listings from the system libraries. (See the library_descriptor command.)

You can give any combination of the control arguments governing naming (-name, -primary, and -match). However, select only one from each group of the following: -chase or -no_chase; -long or -brief; -container, -components, or -entry; -retain or -omit; and -all or -default.

You must use -output_file if you give -all or -default. The particular status information recorded in the output file for -default is controled by the library search program. It includes the information deemed most important for the type of entry contained in the library.

If the file given in -output_file does not exist, it is created by library_fetch; if it does, new status information is appended to the end of the file preserving any previously recorded status. This feature allows you to build a history of the entries copied out of a library. When using -into, ensure that the equal name included in the pathname can be applied to all names to be placed on each of the copied entries. Name duplications can easily result when more than one library entry matches the search names.

The -container and -components control arguments facilitate copying all the library entries included in a given bound segment or related to a given subsystem. For example, by identifying a component of the source archive for a bound segment and using -container, the entire source archive is copied into your directory; similarly, by identifying a directory in the library containing all the component entries of a subsystem and using -components, each component is copied into your directory.

When you use -container, -components, or -chase, it may happen that none of the entrynames on a copied library entry matches any of the search names. Because you may have requested that only matching names be placed on the copies, the library search program places the first entryname on the copy when you select one of these control arguments, in addition to any names you requested.

You are given re access to copied object segments and rw access to all other segments.

EXAMPLES

The command line

```
lf abbrev.pl1 -into >udd>Multics>user>new_=.=
```

copies the source segment abbrev.pl1 into the directory >udd>Multics>user, renaming the copy new_abbrev.pl1.

```
lf bound_qedx_.** -library online
```

copies all the segments in the online libraries whose names begin with bound_qedx_ into your working directory. This might include the source archive, bindable object archive, bound object segment, and bind listing.

```
lf bound_qedx_.** -library online.source -components
```

copies all the source components from the source archive for bound_qedx_ into your directory.

```
lf qedx.pl1 -components
```

copies all the source components in the archive containing qedx.pl1 into your working directory.

```
lf *.alm -lb network.source -into new_=.alm
```

copies all ALM source segments from the network source library into your working directory and adds the prefix new_ to the names placed on each segment.

```
lf pl1_status.info -nm -lb infof
```

copies the pl1_status.info segment from the info segment libraries into your working directory, copying all entrynames from the library entry onto the copy.

```
lf **.ec -library online.??????
```

copies all exec_com segments from the online source and object libraries into your working directory.

```
lf -lb supervisor.bc bound_sss_wired_.*
```

copies the bind segment from the bindable object archive called bound_sss_wired_.archive. Although the object archive itself matches the search name you gave, only the matching archive component is copied because you didn't supply -container.

```
lf -lb include stack_frame.incl.*
```

copies the stack frame declaration include segments for all source languages from the include library into your working directory.

Name: line_length, ll

SYNTAX AS A COMMAND

```
ll {maxlength}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ll {maxlength}]
```

FUNCTION

allows you to control the maximum length of a line output to the device (usually your terminal) that your process is connected to through the user_output I/O switch.

ARGUMENTS

maxlength
is a positive decimal number greater than four that specifies the maximum number of characters that can henceforth be printed on a single line using the I/O switch named user_output. If you don't give it, the current line length is printed. Output lines longer than maxlength are folded.

NOTES

As an active function, the current line length setting is returned and a new one set if you supply maxlength.

Name: link, lk

SYNTAX AS A COMMAND

```
lk path1 {path2...path1N path2N} {-control_args}
```

FUNCTION

creates a storage system link with a specified name in a specified directory pointing to a specified segment, directory, or link (for a discussion of links, see the Programmer's Reference Manual).

ARGUMENTS

path1
specifies the pathname of the storage system entry to which path2N is to point. The star convention is allowed. Give the pathnames in pairs.

path2
specifies the pathname of the link to be created. If omitted (in the final argument position of a command line only), a link to path1 is created in your working directory with the entryname portion of path1N as its entryname. The equal convention is allowed.

CONTROL ARGUMENTS

-chase
creates a link to the ultimate target of path1 if path1 is a link. The default is to create a link to path1 itself.

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

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

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

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

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

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

ACCESS REQUIRED

You 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 introduces a duplication of names within the directory and if the old entry has only one name, you are asked whether to delete the entry with the old name. If you answer "no," the link is not created. If the old entry has multiple names, the conflicting name is removed and a message is issued to you. In either case, since the directory in which the link is to be created is being changed, you must have modify permission for that directory.

See the create_dir and create commands for the creation of directories and segments.

EXAMPLES

The command line

```
! lk >my_dir>beta alpha >dictionary>grammar
```

creates two links, alpha and grammar, in the working directory: the first points to the segment beta in the directory >my_dir, the second to the segment grammar in the directory >dictionary.

Name: linkage_editor, le

SYNTAX AS A COMMAND

```
le paths {-control arguments}
```

FUNCTION

joins a series of object segments into a single execution unit resolving external references to the explicitly named segment(s) within the named libraries.

ARGUMENTS

paths
are specifications of the input binaries that are to be included in the output binary. Valid formats for a path specification are:

Archive
Library archives are unbound archives containing object segments that you can use to resolve external references. To specify a library archive, include the suffix .archive in the name.

Directory
You can specify library directories that contain loose object segments to which external references can be resolved. Any archives found in the directory are expanded and all the components included.

Pathname
You can specify pathnames to specific segments or archive components.

Starname
You can specify starnames specifying groups of loose segments within a directory or an archive component starname specifying a set of components within a particular archive. A starname that includes archives expands each archive.

CONTROL ARGUMENTS

-abort_severity severity, -asv severity
specifies the error severity at which the execution should be terminated; it must be from 0 to 3 (see "Notes on Severity Values" below). Severity 0 errors generate the tables and attempt to resolve links, but stop before generating any output. Severity 4 errors are terminated immediately. (Default: 3, if you give no -asv)

-automatic_segnames, -asn
specifies that entrypoint names are to be automatically added to the containing block as segnames, so that a reference to the entrypoint name alone resolves to the correct block. (Default)

-component_size pages, -compsz pages
specifies the maximum number of pages that a single component of the output object should contain. This value is used to determine at what point the transition is made from single-segment to multisegment file (MSF) and the size of an individual MSF component. (Default: 255, if you supply no -compsz)

-delete {entrypoint}, -dl {entrypoint}
deletes the given entrypoint from the output binary. If you give no entrypoint name, all entrypoints are deleted except the "main_" definition and its associated segnames, if present. You can remove this definition by using "-delete main_".

-display_severity severity, -dsv severity
sets the display severity to the given value. The display severity is the minimum severity error that displays a message; it must be from 1 to 5. (Default: 1)

-force, -fc
suppresses the query before overwriting a nonobject segment or an object segment created by a translator.

-library library_specification, -lb library_specification
specifies one or more object library routines that the linkage editor uses to resolve external references. The library specification is in the same form, and is evaluated in the same manner, as the input path specification.

-list, -ls
creates a listing specifying what segments were involved in the creation of the linked segment, the disposition of each input component, and a list of the retained links and definitions. The name of the listing segment is the same as the output binary with a .list suffix appended.

-map
creates a map of the input components and where they were placed. The name of the listing segment is the same as the output binary with a .list suffix appended.

-no_automatic_segnames, -nasn
does not add entrypoint names to the containing block as segname definitions. This control argument has the same behavior as the binder, and requires that you reference entrypoints with the name of the containing module (i.e., segment$entrypoint), to be resolved internally.

-no_force, -nfc
queries before overwriting a nonobject segment or an object created by a translator. (Default)

-no_list, -nls
does not produce a listing segment. (Default)

-no_version, -nvers
does not print out the version of the linkage editor.

-output_file pathname, -of pathname
specifies the pathname of the output binary to be created. If you give no -of, the output binary is created in your working directory with the name a.out. Before generating the output, linkage_editor checks to insure that the target either is nonexistent or was a bound object prior to overwriting. If the target is not an object, or was not created by linkage_editor, you are queried before it is overwritten unless you give -fc.

-retain entrypoint, -ret entrypoint
specifies an entrypoint that should be retained. The entrypoint is given in the form segname$entryname. The entrypoint is found by conventional methods, and that definition and all segname definitions for that block are retained in the final output. If you give no -ret or -dl, all segnames and entrypoints are retained. If you give any -ret, all other entrypoint and segname definitions are deleted except the "main_" definition, which is always retained, along with its associated segnames, unless deleted by using "-delete main_".

-version, -vers
prints out the version of the linkage editor before linking. (Default)

NOTES

When specifying an entrypoint with -ret or -dl, the entrypoint name given is treated as a pair of starnames: one starname for the segname portion and one for the entrypoint name portion. Parts of the entrypoint not given are assumed to be **. If you supply -ret but no -dl, a global -dl is assumed; otherwise a global -ret is assumed.

When determining which -ret or -dl to apply to any given definition, the most and least specific rules are applied:

1. The following list shows the order from the most specific to the least specific rule. The most specific rule is applied first.

```
most specific: explicit segname and entrypoint
               star segname and explicit entrypoint
               any segname and explicit entrypoint
               explicit segname and star entrypoint
               star segname and entrypoint
               any segname and star entrypoint
               explicit segname and any entrypoint
               star segname and any entrypoint
least specific: any segname and entrypoint
```

where "any" name refers to a star name that matches any name, "star" name refers to a star name that is ambiguous (i.e., contains * or ? characters) but does not match all names, and "explicit" name refers to a name with no * or ? characters.

2. Entrypoint specs with the same class above are ordered based on the order they occurred in the input. If two star names both match a given entrypoint, the first you give on the command line is used.

Here are some examples:

```
-retain bar$foo          = -retain bar$foo
-retain foo              = -retain **$foo
-retain bar$             = -retain bar$**
-retain                  = -retain **$**
-retain b??$             = -retain b??$**
```

NOTES ON SEVERITY VALUES

This command associates the following severity values to be used by the severity active function:

Value	*Meaning*
0	No error
1	Warning
2	Correctable error
3	Fatal error
4	Unrecoverable error

Name: links

SYNTAX AS A COMMAND

```
links star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[links star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of links that match one or more star names.

ARGUMENTS

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

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: list, ls

SYNTAX AS A COMMAND

```
ls {entrynames} {-control_args}
```

FUNCTION

prints information about entries contained in a single directory. There are five entry types supported by this command: segments, multisegment files (MSFs), data management (DM) files, directories, and links. Segments, DM files, and MSFs are referred to collectively as files; segments, MSFs, DM 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 you specify entrynames, only entries having at least one name matching an entryname argument are listed; if you supply no entryname argument, all entries (of the types given by control arguments) in the directory are listed. You can specify a pathname 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 "Control Arguments for Directory.")

Except where otherwise noted in the descriptions of the control arguments, entrynames and control_args can appear anywhere on the command line.

CONTROL ARGUMENTS

Control arguments enable you 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. For convenience, control arguments have been arranged by function.

directory
-pathname path, -pn path

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

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

multiple-name entries
-match
-primary, -pri

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

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

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 you give no directory, your working directory is assumed. This command can list only one directory at a time.

-pathname path, -pn path
lists entries in the directory specified by path.

You can specify the directory to be listed by giving a pathname instead of an entryname. The difference between the two methods is that the entire pathname after -pathname is taken to be that of a directory whose entries are to be listed, while a pathname not preceded by -pathname is separated into its directory and entryname portions; the former specifies the directory, the latter 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, data management files, directories, and links.

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

-data_management_file, -dmf
lists data management files.

-directory, -dr
lists information about directories.

-file, -f
lists information about files (i.e., segments, multisegment files, and data management files, in that order). (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 control argument is inconsistent with and more expensive than -date_time_entry_modified.

-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).

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

-length, -ln
prints the current length computed from the bit count. This control argument is inconsistent with and less expensive than -record. (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 command except when you explicitly request only totals information.

-record, -rec
prints the records used.

If you supply no control arguments from this category, 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 you give -brief, -mode, -record, -length, or -name, only the names column plus those columns explicitly selected by control arguments is 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 you select neither -no_header nor -total, 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 the primary name of each entry, not its secondary names. It does not suppress the printing of any other columns.

These control arguments are applicable only to entries that have more than one name. If you give neither one, all the names of the specified entries are printed.

CONTROL ARGUMENTS FOR ENTRY ORDER

-reverse, -rv
prints entries in the reverse order in which they are found in the directory. If you also supply -sort, 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 (see "Notes on Sorting" below):

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

date_time_contents_modified, dtcm
sort entries by the date and time the contents of the entry were last modified, the most recent ones first. This argument is inconsistent with -date_time_entry_modified. If you choose -date_time_contents_modified and no key follows -sort, that control 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, the most recent ones first. This argument is inconsistent with -date_time_contents_modified. If you supply -date_time_contents_modified and no key follows -sort, that control argument is implied as the default sort key.

date_time_used, dtu
sort entries by the date and time used, the most recent ones first.

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

mode, md
sort entries by access mode in this 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, the largest ones first. This argument is inconsistent with -length. If you specify record and the size column is being printed, the value printed in that column is records used, rather than length

If you use neither -reverse nor -sort, entries are printed in the order in which they are found in the directory.

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 entries having at least one name that matches the specified entryname. The star convention is allowed. The entryname specified in -exclude and any names specified in the entrynames argument to the command operate together to limit the entries that are listed.

-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 had you not given -first.

The following two control arguments exclude entries according to 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 having a date value before the one specified by D.

-to D
does not list any entries having a date value after the one specified by D.

The D value after -from or -to must be a string acceptable to convert_date_to_binary_. If the date-time string contains spaces, enclose the string in quotes. The D value specifies both a date and a time; if you give only a date, convert_date_to_binary_ uses the current time as the default.

If you supply both -from and -to, the -from value must be earlier than the -to value.

CONTROL ARGUMENTS FOR OUTPUT FORMAT

-brief, -bf
if just totals information is being printed, -brief abbreviates and prints on a single line the totals information for all selected entry types; otherwise, it does not print the default columns when you don't explicitly name them in control arguments. For example, typing:

```
ls -dtu -bf
```

prints the names and date-time-used columns, but not the access-mode and length columns.

-short, -sh
prints link pathnames starting two spaces after their entrynames, instead of aligning them in column position 35.

If you select neither -brief nor -short, the output format is not changed.

NOTES

The set of possible columns is different for branches and links. For branches the set (with column 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; 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, these are the items printed for branches: 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 you are listing links and give either modification-date value for printing, sorting, or entry exclusion (using -from and -to), the date-time-entry-modified value of links is used.

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 you omit the sort column XX, 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.

You can only sort links by the name, modification-date, or count columns. If you specify sorting by any other column, 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" above.)

NOTES ON THE DEFAULT

If you invoke the command without any arguments, it lists all segments and multisegment files in the working directory thus: access mode, length of each, and name(s). 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. Within each entry type, entries are listed in the order in which they are found in the directory.

EXAMPLES

The command line

```
! ls -pri -ct
```

lists all files in your working directory (default); the names column contains only the primary names of all entries; the total number of names (for entries having more than one) is printed after the primary name; besides the names column, the access-mode and length columns are printed.

The command line

```
! ls -ex *.*
```

lists all the files in your 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 your working directory having two-component names whose second component is not pl1, printing the default columns.

The command line

```
! ls -dtem -sr
```

lists all files in your working directory, sorted by the date-time-entry-modified column. The date-time-entry-modified and the default columns are printed.

The command line

```
! ls -nm -sr dtm
```

lists all files in your working directory, sorted by the date-time-entry-modified value. Only the names column is printed. The argument dtm is a synonym for dtem.

The command line

```
! ls -sm -nm -pri -nhe
```

lists only the primary name of each segment in your 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 your working directory.

The command line

```
! ls -tt -to "7/1/84 000.0" -dtu -rec
```

prints the totals for all files that have not been used since the end of June 1984. 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 you don't supply path, all pathnames are selected. This argument is incompatible with -entry.

CONTROL ARGUMENTS

-absolute_pathname, -absp
prints the full pathname of each request selected, rather than the entryname.

-admin {User_id}, -am {User_id}
selects the requests of all users or of the user specified by User_id. If you don't choose -am, only your own requests are selected. (See "Notes.")

-all, -a
searches all queues and prints the totals for each nonempty queue whether or not any requests are selected from it. If you specify no -a, nothing is printed for queues from which no requests are selected. This control argument is incompatible with -q.

-brief, -bf
does not print the state and comment of each request. This control argument is incompatible with -lg and -tt.

-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. You can use the star convention. Directory portions of request pathnames are ignored when selecting requests.

-foreground, -fg
searches only the foreground queue and prints the totals for this queue whether or not any requests are selected from it (see -q).

-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 excludes 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; but it includes requests deferred because of load control or resource unavailability, because those conditions could change at any time. (See -psn.)

-long, -lg
prints all the information pertaining to an absentee request, including the long request identifier and the full pathname. If you omit -lg, only the short request identifier, entryname, state, and any comment, are printed.

-long_id, -lgid
prints the long form of the request identifier. If you supply no -lgid, the short form of the request identifier is printed.

-pathname, -pn
prints the full pathname of each selected request, just as -absp does.

-position, -psn
prints the position within its queue of each selected request. When used with -tt, it prints a list of all the positions of the selected requests. When used with -im, it considers only immediate requests when computing positions.

-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 you give no -q, 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" (fg) following -q is equivalent to -fg.

-resource {STR}, -rsc {STR}
selects only requests having a resource requirement. If you specify STR, only requests whose resource descriptions contain that string are chosen. This control argument also prints the resource descriptions of the selected requests, even when you supply no -lg. (See the reserve_resource command for details on resource description specification.)

-restarted
prints only requests that have been restarted by the system after a system crash (see "Notes").

-sender STR
lists only requests from sender STR. You must specify one or more request identifiers. 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 you choose -psn. If the queue is empty, it is not listed.

-truncate, -tc
prints only the requests that have the -tc absout option (see "Notes").

-truncate_restarted, -tc_restarted
prints only those requests that have the -tc absout option and that also have been restarted after a system crash (see "Notes").

-user User_id
selects only requests entered by the specified user (see "Notes").

ACCESS REQUIRED

You must have o access to the queue(s). You must have r extended access to the queue(s) to use -am, -psn, or -user, since it is necessary to read all requests in the queue(s) to select those entered by a specified user or to compute the positions of the chosen requests.

NOTES

All queues are searched for your requests. The request identification, entryname, state, and any comment of each request is printed. If you supply no arguments, only your own requests are selected. Nothing is printed for queues from which no requests are chosen. (See enter_abs_request.)

When you give a user name with -am or -user, a proxy request is chosen if either your name or the proxy user's one on whose behalf you entered the request matches the specified user name.

The entry portion of the pathname argument, the entryname given with -et, and the RJE station name specified after -sender can be star names.

The User_id arguments selected after -am 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 -am with no User_id following it
```

Use "enter_abs_request -tc" if you want the absout file of an absentee request to be truncated before the job is run. Truncation occurs as the job starts up, unless a system crash has occurred. If the request is restarted, the truncation indicator is ignored and the output of the interrupted job is appended to the absout file.

Name: list_accessible, lac

SYNTAX AS A COMMAND

```
lac {path {User_id}} {-control_args}
```

FUNCTION

scans a directory and lists segments, multisegment files (MSFs), and directories with a specified access for a given User_id.

ARGUMENTS

path
is the pathname of the directory to be scanned. If you omit it, or if you supply -working_directory (-wd), your working directory is scanned.

User_id
is an access name. If you omit it, the User_id of the calling process with a star tag is assumed. It can have null components. You can use the star convention for access names (see the set_acl command).

CONTROL ARGUMENTS

If you select no control arguments, all the segments and directories to which the named user(s) has nonnull access are listed.

-dir_mode STR
lists directories to which the named user(s) has any of the modes stated in STR, where STR can be any or all the letters sma.

-seg_mode STR
lists segments to which the named user(s) has any of the modes indicated in STR, where STR can be any or all the letters rew.

ACCESS REQUIRED

You must have s permission on the directory.

NOTES

You can't use User_id unless you have first supplied a path.

If there are more than one User_id (i.e., the specified User_id has null components), the modes for each matched User_id are listed on a per-entry basis.

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}]
```

list_acl

FUNCTION

lists the access control lists (ACLs) of nonlink entries in a directory.

ARGUMENTS

path
: is the pathname of an entry. If it is -working_directory (-wd), your working directory is assumed. You can use the star convention.

This page intentionally left blank.

User_ids
are access control names of the form Person_id.Project_id.tag. All ACL entries with matching names are listed. If you don't give User_ids, the entire ACL is listed.

CONTROL ARGUMENTS

-brief, -bf
suppresses the message "User name not on ACL of path." If you invoke list_acl as an active function and User_id is not on the ACL, the null string is returned.

-chase
chases links matching a star name. Links are always chased when path is not a star name.

-directory, -dr
lists the ACLs of directories only (see "Notes" below). (Default: segments, multisegment files, and directories)

-interpret_as_extended_entry, -inaee
interpret the selected entry as an extended entry type.

-interpret_as_standard_entry, -inase
interpret the selected entry as a standard entry type.

-no_chase
does not chase links. (Default)

-ring_brackets, -rb
lists the ring brackets. Not valid in the active function.

-segment, -sm
lists the ACLs of segments and multisegment files only.

-select_entry_type STR, -slet STR
affects only entries of the entry type selected by STR, which is a comma-delimited list of file system entry types. Use the list_entry_types command to obtain a list of valid entry type values.

ACCESS REQUIRED

You need status permission on the directory.

NOTES

This command provides effective access information only when discretionary access control is being used (regulated by an ACL). If either nondiscretionary access control (regulated by the AIM) or intraprocess access control (regulated by the ring structure) is in operation, use the status command to determine actual access.

The -directory, -segment, and -select_entry_type control arguments are used to resolve an ambiguous choice that may occur when path is a star name.

If you invoke list_acl with no arguments, it lists the entire ACL of your working directory.

For a description of ACLs and ring brackets, see the Programmer's Reference Manual. For a description of the matching strategy, see set_acl.

EXAMPLES

The command line

```
! la notice.runoff .Faculty. Milton
```

lists, from the ACL of notice.runoff, all entries with Project_id Faculty and the entry for Milton.*.*.

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_daemon_requests, ldr

SYNTAX AS A COMMAND

```
ldr {path} {-control_args}
```

FUNCTION

lists requests in the I/O daemon queues. The request identifier and entryname of each request are printed.

ARGUMENTS

path
is the pathname of a request to be listed. The star convention is allowed. Only requests matching this pathname are selected. If you give no path, all pathnames are selected. This argument is incompatible with -entry.

CONTROL ARGUMENTS

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

-admin {User_id}, -am {User_id}
selects the requests of all users, or of the user specified by User_id. If you don't choose -admin, only your own requests are selected. This control argument is incompatible with -user. (See "Access Required" and "Notes" below.)

-all, -a
searches all queues and prints the totals for each nonempty queue whether or not any requests are selected from it. If you supply no -all, the default queue is searched. This control argument is incompatible with -queue.

-brief, -bf
does not print the state and comment of each request. This control argument is incompatible with -long and -total.

-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.

-id ID
selects only requests whose identifier matches the specified ID.

-immediate, -im
selects only requests that can be run immediately and skips requests deferred by the I/O daemon.

-long, -lg
prints all the information about each selected request including the long request identifier and the full pathname. If you omit -long, only the short request identifier, entryname, and state are printed.

-long_id, -lgid
prints the long the request identifier.

-position, -psn
prints the position within its queue of each selected request. When used with -total, it prints a list of all the positions of the selected requests. (See "Access Required" and "Notes.")

-queue N, -q N
searches only queue N. If you don't select -queue, all queues are searched but nothing is printed for queues from which no requests are selected.

-request_type STR, -rqt STR
specifies that requests are to be found in the queue for the request type identified by STR. If you give no -request_type, the default is "printer". List request types with print_request_types.

-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 you choose -position. If the queue is empty, it is not listed.

-user User_id
selects only requests entered by the specified user. (See "Access Required" and Notes.")

ACCESS REQUIRED

You must have o access to the queue(s). You must have r extended access to the queue(s) to use -admin, -position, or -user, since it is necessary to read all requests in the queue(s) to select those entered by a specified user or to compute the positions of the chosen requests.

NOTES

The User_id arguments specified after -admin or -user can have any of the following forms:

```
Person_id.Project_id   matches that user only
Person_id.*            matches that person on any project
Person_id              same as Person_id.*
*.Project_id           matches any user on that project
.Project_id            same as *.Project_id
*.*                    same as -admin with no User_id following it.
```

The state is printed only if it is deferred and you don't supply -brief.

Name: list_dir_info

SYNTAX AS A COMMAND

```
list_dir_info path {-control_arg}
```

FUNCTION

lists the contents of a directory information segment created by save_dir_info.

ARGUMENTS

path
is the pathname of the directory information segment. If path does not end in the suffix dir_info, it is assumed.

CONTROL ARGUMENTS

-brief, -bf
produces a short output.

-long, -lg
produces a long output.

NOTES

If you provide neither -long nor -brief, an intermediate verbosity level is used.

The output of this command is written on the user_output I/O switch. For each entry, a series of lines of the form

```
item_name: value
```

is written. Entries are separated by a blank line.

See the list_dir_info_ subroutine for information on the items printed for each verbosity level.

Name: list_emacs_ctls

SYNTAX AS A COMMAND

```
list_emacs_ctls {terminal_type}
```

FUNCTION

produces a list of all known Emacs terminal types, or verifies the existence in your search rules of specified Emacs terminal controllers.

ARGUMENTS

terminal_type
is a terminal type name. It can be a single starname.

NOTES

When given with no arguments, this command lists all Emacs terminal types.

When given with the name of a terminal type as an argument, list_emacs_ctls either verifies the existence of any Emacs terminal controller in your search rules that matches the starname or prints the message "No Emacs terminal controllers found."

Name: list_entry_types, lset

SYNTAX AS A COMMAND

```
lset
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lset]
```

FUNCTION

prints or returns a list of all the file system entry types that can be found using your search rules.

Name: list_external_variables, lev

SYNTAX AS A COMMAND

```
lev names {-control_args}
```

FUNCTION

prints information about variables managed by the system for the user, including FORTRAN common and PL/I external static variables whose names do not contain dollar signs. The default information is the location and size of each specified variable.

ARGUMENTS

names
are names of external variables, separated by spaces.

CONTROL ARGUMENTS

-all, -a
prints information for each variable the system is managing.

-long, -lg
prints how and when the variables were allocated.

-no_header, -nhe
suppresses the header.

-unlabeled_common, -uc
is the name for unlabeled (or blank) common.

Name: list_fortran_storage, lfs

SYNTAX AS A COMMAND

```
lfs
```

FUNCTION

lists the extended storage segments assigned to the process, grouped according to the storage to which they are extensions.

list_heap_variables

Name: list_heap_variables, lhv

SYNTAX AS A COMMAND

```
lhv names {-control_args}
```

FUNCTION

prints information concerning heap variables. Only variables at the specified execution level(s) are printed. The default information is the location and size of each specified variable. A level description is printed for each execution level specified. The heap variables are displayed starting at the lowest execution level specified.

ARGUMENTS

names
are names of external variables, separated by spaces.

CONTROL ARGUMENTS

-all, -a
prints information for all heap levels. Starting at execution level 0 and ending with the current execution level.

-brief, -bf
prints out the variable name, size, and where it is allocated. (Default)

-from level
specifies what execution level to start printing variables at. If not present, execution level 0 is assumed.

-header, -he
prints the header. (Default)

-long, -lg
prints how and when the variables were allocated.

-no_header, -nhe
suppresses printing of the header.

-to level
specifies what execution level to stop printing variables at. If not present, the current execution level is assumed.

NOTES

Use -from and -to together to specify a range of execution levels to be printed. If you give none, the current execution level is assumed.

Name: list_help, lh

SYNTAX AS A COMMAND

```
lh topics {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lh topics {-control_args}]
```

FUNCTION

displays the names of all info segments (info segs) pertaining to a given topic.

ARGUMENTS

topics
are strings to be searched for in info seg names.

CONTROL ARGUMENTS

-absolute_pathname, -absp
prints or returns full pathnames of info segs, rather than entrynames.

-all, -a
displays the names of all info segs. (Default: to display the names of only those info segs whose names match the topics specified)

-brief, -bf
does not display the alternate names of the info segs. You can't use -brief in the active function. (Default: to display them)

-no_sort
does not sort the output. (Default)

-pathname path, -pn path
specifies the pathname of a directory to search for applicable segments. Multiple -pathname control arguments are allowed. (See "Notes.") (Default: to search the directories in the info_segments search list)

-sort
sorts the output in ascending alphabetic order using as key the primary name of the info segs. If you give -absolute_pathname, -sort uses the entry name part of it as primary name.

NOTES

An info seg is considered to pertain to a given topic if the topic name appears in (i.e., is a substring of) the info seg name. The active function returns the selected names separated by spaces. For information on info segs, see the help command.

This page intentionally left blank.

The default info seg directories contain info segs provided by the site and those supplied with the system. Type "print_search_paths info_segments" to see what the current info segs search list is. For information about search lists, see the search facility commands--add_search_paths, in particular.

Name: list_iacl_dir, lid

SYNTAX AS A COMMAND

```
lid {path} {User_ids} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lid {path} {User_ids} {-control_args}]
```

FUNCTION

lists some or all the entries on a directory initial access control list (initial ACL) of a specified directory.

ARGUMENTS

path
: specifies the directory in which the directory initial ACL should be listed. If path is -working_directory (-wd) or omitted, your working directory is assumed; if omitted, you cannot specify User_ids. The star convention is allowed.

User_ids
: are access control names of the form Person_id.Project_id.tag. All access names matching the given User_ids are listed. If you don't give User_id, the entire initial ACL is listed.

CONTROL ARGUMENTS

-brief, -bf
: suppresses the message "User name not on ACL of path." If you invoke lid as an active function and User_id is not on the initial ACL, the null string is returned.

-chase
: chases links matching a star name. (Default: to chase a link only when specified by a nonstarred pathname)

-no_chase
: does not chase links.

-ring N, -rg N
identifies the ring number whose directory initial ACL is to be listed. It can appear anywhere on the line and affects the whole line. If present, follow it by N (where 0 <= N <= 7). If omitted, your ring is assumed.

ACCESS REQUIRED

You require status permission on the containing directory.

NOTES

If you invoke list_iacl_dir without any arguments, the entire initial ACL for your working directory is listed.

A directory initial ACL contains the ACL entries to be placed on directories created in the specified directory.

For information on initial ACLs, see the Programmer's Reference Manual. For a description of the matching strategy for User_ids, see the set_acl command.

EXAMPLES

The command line

```
! lid all_runoff .Faculty Erasmus..
```

lists, from the directory initial ACL of the all_runoff directory, all entries ending in Faculty.* and all entries with the Person_id Erasmus.

The command line

```
! lid -wd -rg 5
```

lists entries in the ring 5 directory initial ACL of the working directory.

Name: list_iacl_seg, lis

SYNTAX AS A COMMAND

```
lis {path} {User_ids} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lis {path} {User_ids} {-control_args}]
```

FUNCTION

lists some or all the entries on a segment initial access control list (initial ACL) in a specified directory.

ARGUMENTS

path
specifies the directory in which the directory initial ACL should be listed. If path is -working_directory (-wd) or omitted, then your working directory is assumed. If omitted, you can't specify User_ids. The star convention is allowed.

User_ids
are access control names of the form Person_id.Project_id.tag. All access names matching the given User_ids are listed. If you don't give User_id, the entire initial ACL is listed.

CONTROL ARGUMENTS

-brief, -bf
suppresses the message "User name not on ACL of path." If you invoke lid as an active function and User_id is not on the initial ACL, the null string is returned.

-chase
chases links matching a star name. (Default: to chase a link only when specified by a nonstarred pathname)

-no_chase
does not chase links.

-ring N, -rg N
identifies the ring number whose directory initial ACL is to be listed. It can appear anywhere on the line and affects the whole line. If present, follow it by N (where 0 <= N <= 7). If omitted, your ring is assumed.

ACCESS REQUIRED

You need status permission on the containing directory.

NOTES

If you invoke list_iacl_seg without any arguments, the entire initial ACL for your working directory is listed.

A segment initial ACL contains the ACL entries to be placed on segments created in the specified directory.

For information on initial ACLs, see the Programmer's Reference Manual. For a description of the matching strategy for User_ids, see the set_acl command.

EXAMPLES

The command line

```
! lis all_runoff .Faculty. Whitehead
```

lists, from the segment initial ACL of the all_runoff directory, all entries with the Project_id Faculty and the entry for Whitehead.*.*.

The command line

```
! lis -wd -rg 5
```

lists entries in the ring 5 segment initial ACL of the working directory.

Name: list_mdir, lmd

SYNTAX AS A COMMAND

```
lmd volume_names {-control_args}
```

FUNCTION

prints logical volume and master directory quotas.

ARGUMENTS

volume_names
are the names of the logical volumes.

CONTROL ARGUMENTS

-account User_ids
specifies a list of quota account names for which information is desired, where each User_id is of the form Person_id.Project_id. You can't use asterisks when specifying quota account names. Asterisks in account names only match quota account names that contain asterisks.

-all, -a
prints information about all users of the logical volume.

-brief, -bf
suppresses header and shortens the output lines.

-directory, -dr
prints only master directory information.

-long, -lg
prints additional information, including the quota account for each directory.

-owner User_ids
specifies a list of directory owners for which information is desired, where each User_id is of the form Person_id.Project_id. You can use an asterisk when specifying either component of an owner name. If you omit -owner, information is printed only for directories owned by you.

-quota
prints only quota information.

ACCESS REQUIRED

You must have e access to the logical volume to use -account, -all, and -owner. The volume need not be mounted.

NOTES

If you specify neither -quota nor -directory, information about both quotas and directories is printed.

If you give -all, you can't supply -owner and -account. If you use both -owner and -account, information is printed only for directories that match both conditions.

EXAMPLES

In the example below, the user with Person_id Jones on the Fed project requests information on the logical volume named work. The system responds with information about Jones's master directories.

```
! lmd work
  QUOTA  PATHNAME
  100    >udd>Fed>Jones>sub
  250    >udd>Fed>Jones>sub1>sub2
  350 records of quota assigned, 500 available.
```

In the example below, Jones requests brief information on the logical volume named work. The system responds with information about Jones' master directories.

```
! lmd work -bf
  >udd>Fed>Jones>sub
  >udd>Fed>Jones>sub1>sub2
  Quota=500, used=350.
```

In the example below, Jones requests brief, quota information about the logical volume named work.

```
! lmd work -quota -bf
  Quota=500, used=350.
```

In the example below, Jones requests information about all users of the logical volume named work. The use of the -all control argument requires "e" access on the logical volume.

```
lmd work -all
OWNER              QUOTA      PATHNAME
Jones.Fed          100        >udd>Fed>Jones>sub
Jones.Fed          250        >udd>Fed>Jones>sub1>sub2
Smith.Fed          900        >udd>Fed>Smith>work

ACCOUNT                       USED       VOLUME QUOTA
Smith.Fed          900        1000
*.Fed                         350        500

Total volume quota: 1500
Total quota used: 1250
```

Name: list_not_accessible, lnac

SYNTAX AS A COMMAND

```
lnac {path} {User_id} {-control_args}
```

FUNCTION

scans a directory and lists segments and directories to which a specified User_id does not have a given access condition.

ARGUMENTS

path
: is the pathname of the directory to be scanned. If you omit path or give -working_directory (-wd), your working directory is scanned.

User_id
: is an access name that can have null components. If you omit it, your process's User_id is assumed. The star convention for access names is allowed. (See the set_acl command.)

CONTROL ARGUMENTS
: If you select no control arguments, all segments and directories to which the named user(s) has null access are listed.

-dir_mode STR
: lists directories to which the named user(s) does not have any of the modes specified in STR, where STR can be any or all of the letters sma.

-seg_mode STR
lists segments to which the named user(s) does not have any of the modes specified in STR, where STR can be any or all of the letters rew.

ACCESS REQUIRED

You must have status permission on the directory.

EXAMPLES

```
! lnac >udd>work>Smith
  null  Smith.mbx

! lnac >udd>work>Smith -dir_mode m
  s     index
  null  Smith.mbx
  s     newindex
  s     garyl
  s     stuff
```

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
choose it from the following. If you specify no request_identifier, 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. It requires r extended access to the queue(s) to read other users' requests. (Default: to list your own requests)

-all, -a
searches all queues.

-brief, -bf
does not print the request state in normal (not -long) mode. It is incompatible with -long and -total.

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

-long, -lg
prints all information about each selected request, including the long request_id and full pathname (see "Notes"). (Default: to print the short request_id and entryname)

-long_id, -lgid
prints the long request_id.

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

-queue N, -q N
searches only queue N. If you supply no -queue, 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.")

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

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

-request_type STR, -rqt STR
searches the I/O daemon queues belonging to the specified request type (see "Notes.")

-total, -tt
prints only the total number of selected requests and the total number in the queue.

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

NOTES

You can only choose printer, punch, or plotter generic request types with -request_type when you give -long. The print_request_types command gives you a list of these request types.

The -print, -punch, -plot, and -request_type control arguments are mutually exclusive. If you give none, lor lists the default request type used by eor -print (as displayed by print_request_types).

Name: list_pnotice_names

SYNTAX AS A COMMAND

```
list_pnotice_names {-control_arg}
```

FUNCTION

displays the primary names of all protection notice templates.

CONTROL ARGUMENTS

-all, -a
specifies that the entire pnotice search list is to be processed and all templates, including duplicates, are to be listed.

-check, -ck
specifies that the entire pnotice search list is to be processed and that all templates, including duplicates, are to be listed. Checks are also made to the text of each template, and any errors encountered are reported.

NOTES

Default copyright and trade secret notices are indicated. Names displayed by this command are shown as they should be input to the add_pnotice and generate_pnotice commands. If no control arguments are used, names are output in search list order, omitting duplicates.

Name: list_ref_names, lrn

SYNTAX AS A COMMAND

```
lrn paths {-control_args}
```

FUNCTION

lists the reference names associated with a given segment. You can specify segments by either pathname or segment number.

ARGUMENTS

paths
: are the segment numbers and pathnames of segments known in your process. They can be "-name STR" ("-nm STR") to specify a pathname that begins with a minus sign or looks like a segment number. If you supply no paths, information for all segments known in your process is printed, excluding those known in ring 0.

CONTROL ARGUMENTS

-all, -a
: prints information for all known segments, including ring 0 segments. It is equivalent to -from 0.

-brief, -bf
: suppresses printing of the reference names for the entire execution of the command.

-from N, -fm N
: allows you to specify a range of segment numbers. You can use it with -to; information for the segment numbers in this range is printed. If you don't select -to, the highest used segment number is assumed.

-to
: allows you to specify a range of segment numbers. If you supply no -from, the segment number of the first segment not in ring 0 is assumed.

NOTES

You can mixed all the above arguments (segment specifiers and control arguments). For example, in the command line

```
! lrn 156 -from 230 path_one
```

information is printed for segment 156, all segments from 230 on, and the segment whose pathname is path_one. In the default condition, when called with no arguments, list_ref_names prints information on all segments that are not in ring 0.

Name: list_resource_types, lrt

SYNTAX AS A COMMAND

```
lrt {type1...typeN} {-control_args}
```

FUNCTION

prints a list of all resource types described in a resource type description table (RTDT).

ARGUMENTS

type1
is the resource type defined in the RTDT for which information is to be listed. If you give no type, all known resource types are listed.

CONTROL ARGUMENTS

-long, -lg
lists the defined attributes for each resource type.

-no_header, -nhe
omits the column headers.

-pathname path, -pn path
lists resource types defined in the RTDT specified by path. If you give no -pathname, the RTDT residing in >system_control_1 is used.

NOTES

See also the list_resources command, which lists groups of resources according to specified criteria. For example, list_resources can list all resources known by the Resource Control Package (RCP) or those owned by a given user and/or project.

Name: list_resources, lr

SYNTAX AS A COMMAND

```
lr {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lr {-control_args}]
```

FUNCTION

lists groups of resources managed by the Resource Control Package (RCP), selected according to the criteria specified by you.

CONTROL ARGUMENTS

-acquisitions, -acq
lists resources acquired by you specified by -user. If you supply -acquisitions, you must also give -type.

-assignments, -asm
lists resource assignments.

-awaiting_clear
lists those resources awaiting manual clearing.

-device STR, -dv STR
lists only device resources with the name STR.

-logical_volume, -lv
lists logical volumes that are currently attached.

-long, -lg
prints all the information known about each resource listed. If you don't supply it, only the name for each resource listed is printed. It has no effect if you select -acquisitions.

-mounts, -mts
lists resources currently mounted by the process.

-reservations, -resv
lists only device and volume reservations.

-type STR, -tp STR
lists resources of the type STR. See list_resource_types for information on obtaining the names of resource types.

-user User_id
selects a particular user or group of users for whom resource information is to be

printed. If you select no -user, your User_id is assumed. You can use -user only in conjunction with -acquisitions. User_id can be any of the following:

Person.Project
    specifies a particular Person_id and Project_id combination.
*.Project
    specifies all users on a given project project.
.
    specifies all users (i.e., all acquired resources are listed).
free
    specifies all resources in the free pool.
system
    specifies all resources in the system pool.
**
    specifies all users plus the free and system pools (i.e., all registered resources are listed).

NOTES ON ACCESS RESTRICTIONS

You require access to rcp_admin_ to obtain information on other users and read access to the Project Definition Table (PDT) of a given project to obtain information for that project.

NOTES

If you invoke list_resources without any arguments, all resources assigned and devices attached to the calling process are listed.

You cannot use the -assignments, -device, -logical_volume, -long, -mounts, and -reservations control arguments with the active function.

Name: list_retrieval_requests, lrr

SYNTAX AS A COMMAND

```
lrr {path} {-control_args}
```

FUNCTION

lists retrieval requests in the retrieval daemon queues. The request identifier and entryname of each request are printed.

ARGUMENTS

path
is the pathname of a request to be listed. The star convention is allowed. Only requests matching this pathname are selected. If you give no path, all pathnames are selected. This argument is incompatible with -entry.

CONTROL ARGUMENTS

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

-admin {User_id}, -am {User_id}
selects the requests of all users, or of the user specified by User_id. If you don't choose -admin, only your own requests are selected. This control argument is incompatible with -user. (See "Access Required" below.)

-all, -a
searches all queues and prints the totals for each nonempty queue whether or not any requests are selected from it. This control argument is incompatible with -queue.

-brief, -bf
does not print the state and comment of each request. This control argument is incompatible with -long and -total.

-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.

-id ID
selects only requests whose identifiers match the specified ID.

-long, -lg
prints all the information pertaining to a retrieval request. If you omit -long, only the full pathname of the object or subtree to be retrieved is printed.

-long_id, -lgid
prints the long the request identifier.

-position, -psn
prints the position within its queue of each selected request. When used with -total, it prints a list of all the positions of the selected requests. (See "Access Required.")

-queue N, -q N
searches only queue N. If you don't select -queue, all queues are searched but nothing is printed for queues from which no requests are selected.

-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 you choose -psn. If the queue is empty, it is not listed.

-user User_id
selects only requests entered by the specified user (see "Access Required").

ACCESS REQUIRED

You must have o access to the queue(s). You must have r extended access to the queue(s) to use -am, -psn, or -user, since it is necessary to read all requests in the queue(s) to select those entered by a specified user.

NOTES

The default is to list only pathnames for the default queue.

The User_id arguments specified after -am 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.
```

If you select no arguments, only your own requests are selected for listing (see enter_retrieval_request).

Name: list_sub_tree, lst

SYNTAX AS A COMMAND

```
lst {pathnames} {-control_args}
```

FUNCTION

lists the segments in a specified subtree of the hierarchy. The complete subtree is listed unless you supply -dh.

ARGUMENTS

pathname
is the relative pathname of the subtree to be searched. If you specify more than one pathname, only the last one is listed. If you specify no pathname, your working directory is assumed.

CONTROL ARGUMENTS

-all, -a
prints all the names of a segment. (Default: to print only the primary names)

-depth NNN, -dh NNN
scans the hierarchy to the depth indicated by NNN. The depth is relative to the base of the given subtree and must be specified by a decimal integer.

NOTES

For each level listed in the hierarchy, the names are indented three more spaces to indicate the depth of the segments.

Each segment printed includes the number of records used by the segment.

Name: list_tape_contents, ltc

SYNTAX AS A COMMAND

```
ltc vol1 {-comment comment_string}...volN {-comment comment_string}
    {-attach_args} {-control_args}
```

FUNCTION

prints information about files recorded on 9-track magnetic tape in either ANSI standard labeled or IBM standard labeled format.

ARGUMENTS

vol(s) {-comment comment_string}
specifies the name of the tape volume set to be listed (see "Notes on Volume Selection" below).

CONTROL ARGUMENTS

-attach_args
makes mtape_ attach control arguments (see "Notes on Attachment" below).

-brief, -bf
prints the identifier and sequence number of each file selected from the volume set.

-comment comment_string, -com comment_string
displays comment_string on the operator's console when the volume name immediately preceding -comment is mounted.

-from N
starts output of information with file number N, where 0 < N < 10000.

-long, -lg
prints detailed information about each file selected from the volume set.

-to N
stops processing the volume set after file number N, where 0 < N < 10000.

-volume_type type, -vt type
specifies the format type of the volume set being processed, where "type" can be "ibm" or "ansi." (Default: ansi, if you omit -vt)

NOTES ON VOLUME SELECTION

When you specify the volume identifier, use -vt before any volume identifier beginning with a hyphen.

If the volume set to be listed was created on Multics, give only the first volume identifier of the set; the command retrieves the remainder of the identifiers. If it was not created on Multics, give each volume identifer. Up to 64 volumes can be selected.

NOTES ON VOLUME SET INFORMATION

The information ltc prints is extracted from the tape labels and printed in various amounts according to the control arguments you supplied. The information available for each level of control is shown below. Where information is not obtainable from the label, the value "****" is printed as the item entry.

Information printed by list_tape_contents

Id:	<file identifier>	-brief	
Number:	<file sequence number>	-bf	
Format:	<record format>		
Blksize:	<physical block size in characters>	(default)	
Lrecl:	<logical record length in characters>		
Mode:	<encoding mode>		-long
Created:	<file creation date>		-lg
Expires:	<file expiration date>		
Section:	<file set section number>		
Version:	<file generation version number>		
Generation:	<file generation number>		

NOTES ON ATTACHMENT

A complete attach description is created for processing the volume set. It is composed of the string

```
"mtape_ -volume_type ansi -no_display -density 1600 -track 9
        -error -device 1 -label -no_system -no_wait "
```

or

```
"mtape_ -volume_type ibm  -no_display -density 1600 -track 9
        -error -device 1 -label -no_system -no_wait "
```

Any mtape_ attach control arguments you give to ltc are added to the end of the attach description and passed to mtape_ (see the mtape_ I/O module in the Subroutines manual).

NOTES ON mtape_ ARGUMENT DEFAULTS

To avoid unexpected results, ltc supplies complete open, close, and detach descriptions to mtape_. These arguments override any default values that may have been established by the mtape_set_defaults command.

EXAMPLES

The command ! ltc m9999

produces

```
Mounting volume "m9999" with no write ring
Mounted ANSI volume "m9999" (recorded at 1600 BPI), on device tapa_07

ID        Number   Format   Blksize   Lrecl    Mode    Created   Expires

FILE0001    1        SB       8192   1044480   BINARY  09/30/85  12/31/99
FILE0002    2        SB       2048   1044480   BINARY  09/30/85  12/31/99
```

list_tape_contents

The command

```
ltc m9999 -bf -from 2 -to 4 -vt ansi
```

produces

```
Mounting volume "m9999" with no write ring
Mounted ANSI volume "m9999" (recorded at 1600 BPI), on device tapa_07

ID                                    Number

FILE0002                                 2
FILE0003                                 3
FILE0004                                 4
```

The command

```
ltc m9999 -lg
```

produces

```
Mounting volume "m9999" with no write ring
Mounted ANSI volume "m9999" (recorded at 1600 BPI), on device tapa_03

ID: ATTRIBUTEFILE0001                    Number:        1  Section:      1
Created: 10/21/85  Expires: 12/31/99     Generation:    1  Version:      0
Format:  SB        Mode:    BINARY       Blksize:    8192  Lrecl:  1044480

ID: ATTRIBUTEFILE0002                    Number:        2

ID: ATTRIBUTEFILE0003                    Number:        3

Displayed characteristics for the last 3 files are identical.
```

The command

```
ltc m9999 -long -comment "message to console" -vt ibm
```

This page intentionally left blank.

Name: list__temp__segments

SYNTAX AS A COMMAND

```
list_temp_segments {names} {-control_arg}
```

FUNCTION

lists the segments currently in the temporary segment pool associated with your process. This pool is managed by the get_temp_segments_ and release_temp_segments_ subroutines.

ARGUMENTS

names
is a list of names identifying the programs whose temporary segments are to be listed. It cannot be used with -all.

CONTROL ARGUMENTS

-all, -a
lists all temporary segments, including free ones. If the command is issued with no arguments (the default invocation), it lists only those temporary segments currently assigned to programs (i.e., free temporary segments are not listed). This control argument is incompatible with the names argument.

EXAMPLES

To list all the segments currently in the pool, type

```
!  list_temp_segments -all

     5 Segments, 2 Free

!BBBCdfghgffkkkl.temp.0246   work
!BBBCdffddfdffkl.temp.0247   work
!BBBCddffdfffhhh.temp.0253   (free)
!BBBCdgdgfhfgfsf.temp.0254   (free)
!BBBCvdvfgvdgvvv.temp.0321   editor
```

To list the segments currently in use, type

```
!  list_temp_segments

3 Segments

!BBBCdfghgffkkkl.temp.0246   work
!BBBCdffddfdffkl.temp.0247   work
!BBBCvdvfgvdgvvv.temp.0321   editor
```

To list segments used by the program named "editor", type

```
list_temp_segments editor

1 segment

 BBBCvdvfgvdgvvv.temp.0321   editor
```

Name: login_args

SYNTAX AS A COMMAND

```
login_args {argument_number} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[login_args {argument_number} {-control_args}]
```

FUNCTION

prints or returns information about selected login arguments. You can supply login arguments on the command line that creates the process. For interactive processes, this is the login and enter or enterp preaccess requests; for absentee processes, enter_abs_request.

ARGUMENTS

argument_number
selects a single login argument.

CONTROL ARGUMENTS

-count, -ct
prints or returns the number of login arguments you gave when you entered the process creation command. You can't use -count in combination with either argument_number or the other control arguments.

-from argument_number, -fm argument_number
selects all login arguments from argument_number through the last login argument.

-no_requote
does not requote arguments in the string that is returned or printed.

-quote
doubles embedded quotes of each selected argument before it is returned or printed.

-requote
requotes each selected argument before it is returned or printed. (Default)

NOTES

If you supply no argument_number and no -count, the default is -from 1.

If no login arguments exist for the process, login_args prints "There are no login arguments." The active function returns the empty string.

If argument_number exceeds the number of login arguments for the process, login_args prints "Argument number N exceeds the number of login arguments (NN)." The active function returns the empty string.

If -from is in force, explicitly or by default, login_args prints each argument on a separate line, prefixed by its number, a right parenthesis, and a space; for example, the fourth argument is preceded by "4) ". The active function separates multiple arguments by a single space in the return string, which is not itself embedded in quotation marks, and inserts no argument number in the return string.

The -from, -no_requote, -quote and -requote control arguments allow you to obtain return strings that are equivalent to exec_com argument substitution forms.

login_args -control_arg	exec_com form
no control arguments	&rf1
argument_number	&r(argument_number)
-from argument_number	&rf(argument_number)
-no_requote	&f1
argument_number -no_requote	&(argument_number)
-from argument_number -no_requote	&f(argument_number)
-quote	&qf1
argument_number -quote	&q(argument_number)
-from argument_number -quote	&qf(argument_number)
-count	&n

The active function cannot duplicate some exec_com argument forms. The exec_com interpreter has information about surrounding context and produces different results for &q1 and "&q1".

Name: logout

SYNTAX AS A COMMAND

```
logout {-control_args}
```

FUNCTION

terminates your 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, closes all open files and destroys the process.

CONTROL ARGUMENTS

-brief, -bf
prints neither the logout message nor, if you give -hold, the login message.

-hold, -hd
terminates your session but not communication with the system: you can immediately log in without redialing.

NOTES

If your site is security conscious, it may have disabled "logout -hold"; in this case if you wish to change authorization, do this:

1. log out
2. verify, using terminal/modem indications, that the terminal has dropped DTR and that the system acknowledged by dropping DSR
3. log in at the new authorization.

This procedure is the only way to guarantee that you are communicating with the answering service and not with a Trojan horse.

DTR and DSR are EIA RS232 control signals that are part of the interface between your terminal and the system.

Name: long_date

SYNTAX AS A COMMAND

```
long_date {time_string} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[long_date {time_string} {-control_args}]
```

FUNCTION

returns a month name, a day number, and a year as a single string in the form "month day, year" (e.g., November 2, 1985). The format string to produce this is "^mn ^dm, ^9999yc".

ARGUMENTS

time_string
: indicates the date about which information is desired. If you supply no time_string, the current date is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-language STR, -lang STR
: STR specifies the language in which month name, day names, and zone names are to be expressed. (Default: the process default)

-zone STR
: STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default language and zone. Use the display_time_info command to display a list of all acceptable language and zone values.

Name: long_year

SYNTAX AS A COMMAND

```
long_year {time_string} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[long_year {time_string} {-control_arg}]
```

FUNCTION

returns the four-digit number of a year in the clock from 0001 through 9999. The format string to produce this is "^9999yc".

ARGUMENTS

time_string
: indicates the date about which information is desired. If you supply no time_string, the current date is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-zone STR
: STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default zone. Use the display_time_info command to display a list of all acceptable zone values.

Name: low

SYNTAX AS A COMMAND

```
low N
```

SYNTAX AS AN ACTIVE FUNCTION

```
[low N]
```

FUNCTION

returns a specified number N of copies of the first (lowest) character in the ASCII character set (NUL or 000 octal).

NOTES

See the description of the high command.

Name: lower_case, lowercase

SYNTAX AS A COMMAND

```
lowercase strings
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lowercase strings]
```

FUNCTION

returns strings with all uppercase alphabetic characters translated to lowercase.

ARGUMENTS

strings
is one or more character strings.

NOTES

Returned strings are separated from each other by a space. See the description of the upper_case command.

EXAMPLES

The following interactions illustrate use of lower_case as an active function:

```
ioa_ [lower_case "The time zone is EST."]
the time zone is est.
```

The following interactions illustrate use of lower_case as a command:

```
lower_case It is winter in Boston. The time zone is EST.
it is winter in boston. the time zone is est.
```

Name: ltrim

SYNTAX AS A COMMAND

```
ltrim STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ltrim STRA STRB]
```

FUNCTION

returns a character string trimmed of specified characters on the left.

NOTES

The ltrim command, or active function, finds the first character of strA not in strB, trims the characters from strA preceding this character, and returns the trimmed result. Space characters are trimmed if strB is omitted.

EXAMPLES

```
!  string [ltrim 000305.000 0]
   305.000
!  string [ltrim "   This is it.   "]
   This is it.
```

Name: lv_attached

SYNTAX AS A COMMAND

```
lv_attached lv_name
```

SYNTAX AS AN ACTIVE FUNCTION

```
[lv_attached lv_name]
```

FUNCTION

returns true if the volume specified (i.e., the lv_name argument) is attached to the user's process or if the volume is a public logical volume; otherwise it returns false.

ARGUMENTS

lv_name
    is the name of the logical volume.

NOTES

See the related commands attach_lv and detach_lv.

Name: mail, ml

SYNTAX AS A COMMAND

```
ml path User1...{UserN} {-control_args}

ml {destination} {-control_args}
```

FUNCTION

sends a message to another user or prints messages in any mailbox to which you have sufficient access.

ARGUMENTS

path
: is the pathname of a segment to be sent or is an asterisk (*) to indicate that you wish to type a message to be sent (see "Notes on Composing Mail" below).

User_1
: is the User_id of a person to whom mail is to be sent. Mail is sent to the mailbox >udd>Project_id>Person_id>Person_id.mbx for each Person_id.Project_id (User_id) argument in the command line.

destination
: can be User_id to specify a mailbox. If destination contains a < or >, it is the pathname of a mailbox. The mbx suffix is assumed in this case. You cannot use destination with -pathname. (Default: your default mailbox)

CONTROL ARGUMENTS

-acknowledge, -ack
: requests acknowledgement of the pieces of mail. The acknowledgement consists of the string:

```
"Acknowledge message of <date-time sent>"
```

and is sent as an interactive message when you invoke this command to print mail.

-brief, -bf
: prints the total number of messages in the mailbox. If the mailbox is empty, nothing is printed.

-exclude STR, -ex STR
ignores messages sent by users whose User_id matches the User_id specified in STR. The star convention is allowed. If you supplied -match, exclusion is performed before matching.

-header, -he
prints only the header line for each message. No messages are deleted.

-match STR
prints messages sent by users whose User_id matches the User_id specified in STR. The star convention is allowed. If you gave -exclude, exclusion is performed before matching.

-no_notify, -nnt
suppresses the sending of an interactive "You have mail" notification.

-pathname path, -pn path
specifies a mailbox by pathname. The mbx suffix is assumed.

NOTES

The extended access used on mailboxes (which are ring 1 segments) permits you to control other users' access to it. Adding, reading, and deleting messages are independent privileges under extended access; for example, you can give a user access to only add messages, to other user access to add messages and to read and delete only the messages he or she has added. Mail and interactive messages sent to a user are placed in the mailbox >udd>Project_id>Person_id>Person_id.mbx.

If you are accepting interactive messages, you receive an immediate notification of the form:

```
You have mail from Person_id.Project_id.
```

Segments to be mailed have a maximum length of one record (4096 ASCII characters).

See print_mail, read_mail, and send_mail.

NOTES ON COMPOSING MAIL

If path is *, mail responds with "Input" and accepts lines from the terminal until you type a period on a line by itself. The typed lines are then sent to the specified user(s).

NOTES ON PRINTING MAIL

When the contents of the mailbox named by path are printed, they are preceded by a line of the form:

```
N messages.
```

Each message is preceded by a line of the form:

```
i) From: Person_id.Project_id (sent_from) date time (N lines)
```

where:

i
: is the incremental number of the message. The messages are printed in ascending numerical order; the oldest one is numbered 1.

Person_id
: is your registered person identifier.

Project_id
: is the name of the project on which you were logged in when you sent the message.

sent_from
: is an optional field that further identifies you, e.g., your anonymous log-in name.

date
: is the date you sent the message, of the form mm/dd/yy to indicate the month, day, and year.

time
: is the time you sent the message, of the form hhmm.m zzz www to indicate the hours, minutes, and tenths of minutes in 24-hour time followed by the time zone and day of the week.

N lines
: is the number of lines in the message.

After printing all messages, this command asks whether you want them deleted. If yes, all messages are deleted; if no, no messages are deleted. In either case, your return to command level.

If you quit while the messages are being printed and then issue program_interrupt, the command stops printing and asks whether to delete all messages, including those that were not printed.

NOTES ON CREATING A MAILBOX

A default mailbox is created the first time print_mail, read_mail, or accept_messages is used. The default mailbox is

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

NOTES ON EXTENDED ACCESS

Access on a newly created mailbox is set to adrosw for you, aow for *.SysDaemon.*, and aow for *.*.*. The types of extended access for mailboxes are:

add, a
adds a message.

delete, d
deletes any message.

read, r
reads any message.

own, o
reads or delete only your own messages, i.e., those sent by you.

status, s
finds out how many messages are in the mailbox.

wakeup, w
sends a wakeup when adding a message (used by send_message).

The modes n, null, and "" indicate null access.

The mode u indicates that the user has access to send "urgent" messages to the user accepting messages on the mailbox. However, urgent messages are not currently implemented.

Name: manage_volume_pool, mvp

SYNTAX AS A COMMAND

```
mvp key {args} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[mvp key {args} {-control_args}]
```

FUNCTION

allows a user or a group of users to regulate the use of a predefined set of volumes (tape reels, etc.). The concept of volume sets as used with tape_ansi_/tape_ibm_ is also supported. The default volume pool for each user pool is named Person_id.volumes and exists in your home directory. This default can be reset via the use key described below.

ARGUMENTS

key
can be any of the tokens (see "List of Keys"):

add (a)	pv_expire (pvexp)
allocate (alloc)	remove_volume_set (rmvs)
append_volume_set (appvs)	reserve (rsv)
delete (dl)	reuse
free	set
list (ls)	test
print (pr, p)	use

args
are optional arguments associated with the various keywords. They can be volume names of volumes to be acted upon according to the key given.

CONTROL ARGUMENTS

are optional control arguments associated with the various keywords.

LIST OF KEYS

add, a
`Usage: mvp a vol_names {-control_args}`

adds the selected volumes to your volume pool. Each volume added is considered a volume set of size one.

Control Arguments:

-force, -fc
adds the volumes it can without aborting the entire request or querying you (see "Notes on Querying" below).

-pv_expire DATE, -pvexp DATE
sets the expiration date of all given physical volumes to DATE. DATE must be acceptable to convert_date_to_binary_. (See "Notes on Expiration Dates" below.)

Failure can occur when the volume to be added already exists in the volume pool.

As an active function, "true" is returned if all volumes were successfully added (see "Notes on Active Function" below).

allocate, alloc

```
Usage: mvp alloc {vol_names} {-control_args}
```

allocates the supplied free or reserved volume sets. The vol_names argument can be primary or secondary volumes of a set.

Control Arguments:

-comment STR, -com STR
: designates STR as the comment to be associated with the vol_names chosen.

-expire DATE, -exp DATE
: sets the expiration date associated with the data on the allocated vol_names. DATE must be acceptable to convert_date_to_binary_. (See "Notes on Expiration Dates" below.)

-first {N}, -ft {N}
: allocates the first N free volume sets in the pool; "first" is defined as the volume sets that have been most recently freed (i.e., volume sets that have the most recent state change date associated with them). (Default: 1, if you give no N)

-force, -fc
: assigns the volume sets it can without aborting the entire request or querying you (see "Notes on Querying").

-last {N}, -lt {N}
: allocates the last N free volume sets in the pool; "last" is defined as the volume sets that have been in the free state the longest (i.e., volume sets that have the oldest state change date associated with them). (Default: 1, if you supply no N)

-volume_size {N}, -vs {N}
: allocates a volume set of size N. If more than one volume set of size N exists and neither -first nor -last is used, then the last free (least recently freed) volume set of size N is assigned. If there are no volume sets of size N, an error message occurs. The test key can be used to avoid such errors. (Default: 1, if you supply no N)

The use of vol_names and -ft, -lt, or -vs are mutually exclusive. If -vs is not selected, the default action is to consider all volume sets of any size in the pool.

Reserved volume sets to be allocated must be specified by their vol_name. Failure can occur if the volume sets to be allocated do not exist in the volume pool, if they are not in the free or reserve state, or if N free volume sets are requested and only N-1 free volume sets exist in the pool.

As an active function, the allocated vol_names are returned.

append_volume_set, appvs

```
Usage: mvp appvs primary_vol_name {vol_names} {-control_args}
```

appends the vol_names to the volume set specified by the primary_vol_name. The vol_names cannot be secondary volumes of another existing volume set. The pv_expire date of each vol_name is checked before appending; if this date is passed (i.e, less than the current date), the state of the volume becomes expired, a message to that effect is printed, and the volume is not appended.

Control Arguments:

If no vol_names are given, acceptable control arguments are

-first {N}, -ft {N}
: appends the first N free volume sets to the new set; "first" is defined as the volume sets that have been most recently freed. (Default: 1, if you give no N)

-force, -fc
: appends the volumes it can without aborting the entire request or querying you (see "Notes on Querying").

-last {N}, -lt {N}
: appends the last N free volume sets to the new set; "last" is defined as the volume sets that have been in the free state the longest. (Default: 1, if you supply no N)

-volume_size {N}, -vs {N}
: appends a volume set of size N. If more than one volume set of size N exists, and neither -first nor -last is used, then the last free volume set of size N is appended to the new set. If no volume set of size N exists, an error message occurs. The test key can be used to avoid such errors. (Default: 1, if you supply no N)

The vol_names argument and -ft, -lt, or -vs are mutually exclusive. If -vs is not chosen, the default action is to consider volume sets of any size in the pool.

Failure can occur when a given volume set does not exist or when a volume to be appended is not free.

As an active function, the volume names that were appended to the volume set are returned. If a multiple volume set is appended, all the volumes in the set are returned (see "Notes on Active Function").

delete, dl

```
Usage: mvp dl vol_names {-control_arg}
```

deletes the specified physical volume sets from your volume pool. Volume sets

must be in the free, reserve, or pv_expire state to be deleted. If vol_name is a multiple volume set, all volumes in the set are deleted from the pool; if it is a secondary_vol_name, the volume set to which the secondary volume belongs to is deleted.

Control Argument:

-force, -fc
deletes the volume sets it can without aborting the entire request or querying you (see "Notes on Querying").

As an active function, a successful delete returns "true," "false" otherwise (see "Notes on Active Function").

free

```
Usage: mvp free {vol_names} {-control_args}
```

frees the selected volume sets in your volume pool by changing the state to "free." Upon freeing a volume set, the pv_expire date of each volume in the set is checked; if one of these dates is passed, the state of the volume set becomes pv_expired and a message to that effect is printed. In the case of an allocated volume to be freed, the -expire date is checked first; if this date has not passed yet, the volume set is not freed and a message to that effect is printed. If vol_name is a secondary_vol_name, the state of the volume set to which the secondary volume belongs to becomes free.

Control Arguments:

-brief, -bf
suppresses the pv_expire message or the allocate expire message when appropriate.

-expire, -exp
frees all allocated volume sets for which the respective expiration date has been passed.

-force, -fc
frees the volume sets it can without aborting the entire request or querying you (see "Notes on Querying").

-force_expire, -fexp
overrides the checking of the -expire date, freeing allocated volume sets with an unexpired expiration date.

-match STR
frees only those volume sets whose comment contains STR as a substring. The volume sets can be in the allocate, reserve, or pv_expire state. Both expirations dates are first checked before freeing the volume sets.

The vol_names argument and -exp, -fc, or -match are mutually exclusive.

As an active function, a successful free returns "true," otherwise "false" is returned (see "Notes on Active Function").

list, ls

```
Usage: mvp ls {vol_names} {-control_args}
```

lists information about the specified volume sets or about all volume sets in the pool if no arguments are supplied. The list is printed in state change date order with the volumes whose states changed most recently listed first.

Control Arguments:

-header, -he
prints the header information of the list display. (Default)

-no_header, -nhe
suppresses printing of the header.

-total, -tt
prints the total number of volume sets in the pool.

Control Arguments for Field Selection:

These control arguments determine which fields are displayed by list.

-comment, -com
lists only the comment field of the designated volumes.

-default_format, -dfmt
lists the name, state date, state, and comments fields, as illustrated below. This is the default list format if no other field control arguments are chosen.

```
Name            State Change       State  Comment
1234567890  06/05/83 1316.1 mst    ALOC   This is a comment
1000        01/23/83 0645.0 mst    FREE   Multiple volume set
 1001
 1002
```

-expire_date, -edt
lists the expiration date field of the given allocated volume sets.

-name, -nm
lists the volume name field of the selected volume sets.

-pv_expire_date, -pvedt
lists the physical expiration date field of the supplied physical volume sets.

-state
lists only the state field of the indicated volume sets.

-state_date, -sdt
lists only the state change date field of the designated volume sets.

If you want to list the expiration dates, you have to use -edt and/or -pvedt.

Control Arguments for Volume State Selection:

Volume sets can be listed according to what state they are in: allocate, free, pv_expire, or reserve. The default format is displayed unless field control arguments are also selected. The specified vol_names and the control arguments listed below are mutually exclusive.

-all_states, -ast
lists all volume sets in the pool. (Default, when you invoke list with no arguments)

-allocate, -alloc
lists only those volume sets that are allocated.

-free
lists only those volume sets that are free.

-pv_expire, -pvexp
lists only those volume sets that are in the pv_expire state.

-reserve, -rsv
lists only those volume sets that are reserved.

Additional Selection Control Arguments:

These control arguments allow one to list volume sets by criteria other than the volume state or in combination with the volume state control arguments. The default format display is used unless field arguments are given. The indicated vol_names are incompatible with these control arguments:

-expire, -exp
lists all allocated volume sets for which the respective expiration date has been passed.

-first {N}, -ft {N}
lists the first N volume sets in state change date order; "first" is defined as the volume sets that have the most recently changed state. (Default: 1, if you give no N)

-last {N}, -lt {N}
lists the last N volume sets; "last" is defined as the volume sets that have the least recently changed state. (Default: 1, if you supply no N)

-match STR
lists only those volume sets whose comment contains STR as a substring.

-volume_size {N}, -vs {N}
lists only the volume sets in the data base of size N. If -vs is not given, all volume sets of any size are listed. (Default: 1, if you supply no N). The volume sets have the following format:

```
Volume          State Change        State     Comment
4123        12/11/83 1022.9 mst     RESV      backup
  1000
  3400
```

where vol_name 4123 is the primary volume of the volume set and 1000 and 3400 are the secondary volumes.

As an active function, list with no control arguments returns a list of the primary volume names of all volume sets; otherwise, the control arguments chosen determine what is returned for each selected volume. Examples:

Given the pool:

```
Volume          State Change        State     Comment
500         10/23/83 1042.9 mst     FREE
1000        10/23/83 0853.1 mst     ALOC      acct
5000        09/11/83 1022.9 mst     RESV      backup
 200
 457
6500        06/28/83 1134.1 mst     ALOC      test1
4100        06/28/83 1132.8 mst     ALOC      test1
374         03/27/83 0740.0 mst     RESV

! mvp ls -alloc -lt 2 -dmft -edt

Volume          State Change        State          Expires          Comment
6500        06/28/83 1134.1 mst     ALOC     01/31/84 2100.0 mst     test1
4100        06/28/83 1132.8 mst     ALOC     01/31/84 2100.0 mst     test1

! mvp ls -alloc -match ac -state_date -name

Volume          State Change
1000        10/23/83 0853.1 mst

! mvp ls -lt 2 -name -sdt -pv_expire_date

Volume          State Change          Volume Expires
4100        06/28/83 1132.8 mst   06/30/85 2100.0 mst
374         03/27/83 0740.0 mst
```

where volume 374 has no physical volume expiration date.

print, pr, p
prints the pathname of the current volume pool segment. As an active function, the pathname is returned.

pv_expire, pvexp
`Usage: mvp pvexp vol_names {-control_arg}`

designates the specified volume sets and their secondary volumes as expired by changing the state to pv_expire. The set key can be used to reset the physical volume expiration date. Volume sets cannot be in the allocate state. If vol_name is a secondary_vol_name, the state of the volume set to which the secondary volume belongs to becomes pv_expire. As an active function, "true" is returned when all indicated vol_names are successfully expired, otherwise "false" is returned (see "Notes on Active Function").

Control Argument:

-force, -fc
expires the volume sets it can without aborting the entire request or querying you (see "Notes on Querying").

remove_volume_set, rmvs
`Usage: mvp rmvs primary_vol_name {secondary_vol_names} {-control_args}`

removes the secondary_vol_names from the volume set as specified by primary_vol_name. The volumes removed are placed in the pool as volume sets of size 1 in the free state. The primary volume of a volume set cannot be removed unless -all is used. If the volume set supplied is in the allocate state, the -expire date is checked first. If this date has not passed yet, the request is aborted and a message is printed. Upon freeing each volume, its pv_expire date is checked; if this date is passed, the state of the volume becomes expired and a message to that effect is printed.

Control Arguments:

-all, -a
breaks the volume set into individual volume sets of size 1, each with a state of free (or pv_expire, as explained above); this includes freeing the primary volume.

-brief, -bf
suppresses the pv_expire message or the volume-not-removed message when appropriate.

-force, -fc
removes the volumes it can without aborting the entire request or querying you (see "Notes on Querying").

-force_expire, -fexp
overrides the checking of the -expire date, freeing allocated volume sets with an unexpired expiration date.

-pv_expire, -pvexp
removes all volumes of the designated volume set whose pv_exire date has been passed and puts them in the pool with the pv_expire state.

The specified secondary_vol_names and -all and -pvexp are mutually exclusive.

Failure can occur when the volumes to be removed do not exist in the volume set.

As an active function, the vol_names removed from the volume set are returned (see "Notes on Active Function").

reserve, rsv
```
Usage: mvp rsv {vol_names} {-control_args}
```

reserves the indicated free volume sets. If vol_name is a secondary volume name of a volume set, this volume set is reserved. Only a free volume set can be reserved and used by a person, so long as her or his process is active. When the reserved volume is referenced, a check is made to see if one's process is still active; if not, the volume can be used as requested. When reserving, the pv_expire date is checked; if this date is passed, the state of the volume set becomes pv_expire and a message is printed.

Control Arguments:

-comment STR, -com STR
states that STR be the comment associated with the given volume sets reserved.

-first {N}, -ft {N}
reserves the first N free volume sets; "first" is defined as the volume sets that have been most recently freed. (Default: 1, if you give no N)

-force, -fc
reserves the volumes it can without aborting the entire request or querying you (see "Notes on Querying").

-last {N}, -lt {N}
reserves the last N free volume sets; "last" is defined as the volume sets that have been in the free state the longest. (Default: 1, if you supply no N)

-volume_size {N}, -vs {N}
reserves a volume set of size N. If more than one volume set of size N

exists and neither -first nor -last is given, then the last free volume set of size N is reserved. If no volume set of size N exists, an error message occurs. The test key can be used to avoid such errors. (Default: 1, if you supply no N)

The vol_names argument and -ft, -lt, or -vs are mutually exclusive. If -vs is not given, the default action is to consider volume sets of any size in the pool.

Failure can occur if the specified vol_names to be reserved do not exist, if they are not in the free state, if no volume set of size N exists, or if there are only N free volumes in the pool and N+1 volumes are selected to be reserved.

As an active function, a list of the volume set names reserved is returned (see "Notes on Active Function").

reuse

```
Usage: mvp reuse {vol_names} {-control_args}
```

allows one to free and reallocate a designated number of allocated volume sets, without needing to know the volume names. Before this operation is performed, the -expire date is checked; if this date has not passed yet, the request is not performed and a message with that purport is printed.

Control Arguments:

-brief, -bf
: suppresses the reuse request message not performed.

-first {N}, -ft {N}
: reallocates the first N allocated volume sets; "first" is defined as the volume sets that have been most recently allocated. Failure can occur when N volume sets are requested and only N-1 volume sets are in the pool. (Default: 1, if you give no N)

-force, -fc
: reuses the volume sets it can without aborting the entire request or querying you (see "Notes on Querying").

-force_expire, -fexp
: overrides the checking of the -expire date and reallocates the specified volume sets with an unexpired expiration date.

-last {N}, -lt {N}
: reallocates the last N allocated volume sets; "last" is defined as the volume sets that have been in the allocated state the longest. Failure can occur when N volume sets are requested and only N-1 volume sets are in the pool. (Default: 1, if you supply no N)

-match STR
: reallocates all volume sets whose comment contains the substring STR.

-volume_size {N}, -vs {N}
reallocates a volume set of size N. If more than one volume set of size N exists and neither -first nor -last is given, then the last allocated volume set of size N is reallocated. If no volume set of size N exists, an error message occurs. The test key can be used to avoid such errors. (Default: 1, if you supply no N)

The vol_names argument and -ft, -fc, -lt, -match, or -vs are mutually exclusive. If -vs is not supplied, the default action is to consider all allocated volume sets of any size in the pool.

As an active function, the primary names of the volume sets that were reused are returned (see "Notes on Active Function").

set

```
Usage: mvp set vol_names -control_args
```

sets the comment or expiration date fields of the specified volume sets.

Control Arguments:

-comment STR, -com STR
sets the comment field of the designated vol_names to STR. If the volume name is secondary, the comment field of the volume set to which it belongs is changed.

-expire DATE, -exp DATE
sets the expiration date associated with allocated vol_names selected. The vol_names argument must be in the allocated state, otherwise a message is printed to that effect. If the volume name is secondary, the expire date of the allocated volume set to which the secondary volume belongs is changed. DATE must be acceptable to convert_date_to_binary_. (See "Notes on Expiration Dates.")

-pv_expire DATE, -pvexp DATE
sets the physical volume expiration date of the specified vol_names to DATE. The volume set state does not become pv_expire until the next time the volume set state is changed (via the alloc, free, and rsv keys), after the date is reached. If the volume name is primary, the expiration date is reset to DATE for only the primary volumes unless followed by -secondary_volumes. Secondary volumes of a volume set can be selected individually to set their pv_expire date. DATE must be acceptable to convert_date_to_binary_. (See "Notes on Expiration Dates.")

-secondary_volumes, -svol
is used in conjunction with -pvexp to indicate that the secondary volumes of the primary volume name preceeding -svol should also be set to the given pv_expire date.

As an active function, "true" is returned when the date and/or comment has been successfuly changed for all designated vol_names (see "Notes on Active Function").

test

```
Usage: mvp test {vol_names} {-control_args}
```

tests what state the specified volume sets are in. If the vol_name supplied is a secondary volume, the attributes of the set to which it belongs to are tested.

Control Arguments:

-allocate, -alloc
tests whether any volume sets or vol_names selected are in the allocate state.

-first {N}, -ft {N}
tests the first N volume sets; "first" is defined as the volume sets that have been most recently allocated. (Default: 1, if you give no N)

-free
tests whether any volume sets or vol_names given are in the free state. (Default)

-last {N}, -lt {N}
tests the last N volume sets; "last" is defined as the volume sets that have been in the allocated state the longest. Failure can occur when N volume sets are requested and only N-1 volume sets are in the pool. (Default: 1, if you supply no N)

-match STR
tests all volume sets whose comment contains the substring STR.

-pv_expire, -pvexp
tests whether any volume sets or vol_names specified are in the expire state.

-reserve, -rsv
tests whether any volume sets or vol_names selected are in the reserve state.

-volume_size {N}, -vs {N}
tests whether volume sets of size N are in one of the specified states. If more than one volume set of size N exists and neither -first nor -last is given, then the last free, reserved, and so on (as indicated by the state control arguments) volume set of size N is tested. If no volume set of size N exists, an error message is returned. (Default: 1, if you supply no N)

The vol_names argument and -ft, -lt, -match, or -vs are mutually exclusive. If -vs is not given, the default action is to consider all volume sets of any size in the pool.

As an active function, "true" is returned if a volume set with the state specified is found in the pool.

use, u

```
Usage: mvp u {path}
```

specifies the pathname of the mvp segment to be used by future invocations of mvp in this process. The volumes suffix is assumed. If you omit path, your default volume segment is used. If the segment specified by path does not exist, mvp creates it. As an active function, the pathname is returned.

NOTES

Normally a tape reel or disk pack is a volume, but any other set of objects, such as library books or portable terminals, could just as easily be regulated. The default volume pool for each user pool is named Person_id.volumes and exists in your home directory; you can reset this default using this key. You can add objects to, or delete objects from, the pool. An object can have one of four states: allocate, free, pv_expire, or reserve. Associated with each object in the pool is a state, a state change date, a comment, and two optional expiration dates. The comment field can be any ASCII string of up to 64 characters. The comment is intended to describe the contents of the volume, but can easily describe the attach description that creates the volume, etc.

NOTES ON ACTIVE FUNCTION

As an active function, mvp returns "true," a list of volume names, or pathnames, depending on the actual key request. In the case of a partial success when an attempt to query you is made, active_fnc_err_ is called; however, its action is overridden when you give -force or -fexp. This results in returning "true" or the partial list of volume names successfully acted upon.

NOTES ON EXPIRATION DATES

There are two kinds of expiration dates. One is associated with the physical volume and is referenced with -pv_expire. The state pv_expire is associated with the physical volume also. The pv_expire date is checked whenever the volume state is changed to free, allocate, or reserve or when a volume is removed from, or appended to, another volume set. When a secondary volume within a volume set expires, the next time that volume set state is changed the state of the volume set changes to pv_expire and you are notified. The date is not checked when you use the reuse key. When the physical expiration date of a volume set is reached, the state changes to pv_expire and a message is printed. You can only delete or free expired volume sets from the volume pool; if you free them, you can reset the expiration date using the set key. Physical volume expiration dates are useful for keeping track of old and bad tapes.

The other optional expiration date applies only to allocated volume sets. It is referenced by -expire and refers to the time when the information on the volume set is considered no longer relevant. You can set this date at allocation time. When the volume set is freed, the expire date field is cleared. This expire date is useful for keeping track of recycling volume sets. There is no state associated with this expire date.

Both kinds of expiration dates have to be explicitly set by you using control arguments or the set key. If not set, the default is for volume sets never to expire.

NOTES ON QUERYING

You are queried on whether to continue a requested action (allocate, reserve, etc.) when the action can only be performed on some but not all volume names specified. You can use -force to override the query. If you answer "no," the entire request is aborted and no action is taken. If you answer "yes" to the query or use -force, the request is performed on the volume sets eligible and a message is printed listing the volume sets on which no action was taken.

NOTES ON VOLUME SETS

The volume pool is made up of volume sets. A volume set consists of a primary volume and, optionally, a group of secondary volumes. The set is referenced by the primary volume name. The size of a volume set can range from 1 to N volumes. Therefore a single volume A is a volume set of size one; it is a primary volume with no associated secondary volumes. A volume set grows and shrinks by the appvs and rmvs keywords. For example, three volume sets, A, B, C, each of size one, can be "bound" together into one volume set thus: "mvp appvs A B C". The primary volume name to reference this set is A with two secondary volumes, B and C, associated with it. The state of a volume set is changed by the allocate, free, reserve, and pv_expire keys, given the primary volume name or the -volume_size and any optional control arguments.

Name: master_directories, mdirs

SYNTAX AS A COMMAND

```
mdirs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[mdirs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of master directories that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

EXAMPLES

The following interaction illustrates the use of the mdirs active function.

```
string [mdirs >udd>**]
Multics SysMaint
```

Name: max

SYNTAX AS A COMMAND

```
max num_args
```

SYNTAX AS AN ACTIVE FUNCTION

```
[max num_args]
```

FUNCTION

returns the maximum of the numeric arguments passed to it.

EXAMPLES

```
string [max 3.6 1e-3]
3.6
```

Name: mbx_create, mbcr

SYNTAX AS A COMMAND

```
mbcr paths
```

FUNCTION

creates a mailbox with a given name in a specified directory.

ARGUMENTS

paths
are pathnames of mailboxes to be created. You need not give the mbx suffix.

ACCESS REQUIRED

Modify and append permissions are required on the parent directory.

NOTES

Name duplication is handled this way: If the old segment with that name has other names, the conflicting name is removed and you are notified; otherwise you are asked whether to delete the old segment. The extended access placed on a new mailbox is

```
adrosw  Person.*.*
aow     *.SysDaemon.*
aow     *.*.*
```

For more information on extended access, see the mail command. *

EXAMPLES

The command line

```
! mbcr JBentham RAmundsen.home >udd>Multics>LAriosto>LAriosto
```

creates the mailboxes JBentham.mbx and RAmundsen.home.mbx in the working directory and creates the mailbox LAriosto.mbx in the directory >udd>Multics>LAriosto.

Name: memo

SYNTAX AS A COMMAND

```
memo {-memo_options} memo_text

memo {-action_args} {-memo_options} {-selection_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[memo memo_text]

[memo -list {-totals}]
```

FUNCTION

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

ARGUMENTS

memo_text
is the text of the memo being set. It cannot be longer than 132 characters. You can specify it in one of two forms:

STR
is the string without an initial hyphen that begins 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

Use these control arguments 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. An alarm memo is printed, or executed if set with -call, when its timer goes off, if timers are enabled, rather than being explicitly processed. It is deleted immediately after it reaches maturity, unless you supplied -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 for the memo to mature. DT is truncated to the midnight preceding the date in which DT falls. (See "Notes.")

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

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

-no_retain, -nret
processes the memo only once and then deletes it. (Default for alarm memos)

-repeat DT, -rpt DT
identifies the repeat interval of the memo, where DT must be greater than or equal to one minute. When the memo is reset, the new maturity time is the next successive interval that matures in the future. When used as a selection argument, all repeating memos are selected, regardless of the repeat intervals given. (See "Notes" and "Notes on Repeating Memos.")

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

-retain, -ret
keeps an alarm memo as an ordinary printing (or executing if set with -call) memo after it matures, rather than being deleted. (Default for nonalarm memos)

-time DT, -tm DT
identifies a time for the memo to mature (see "Notes").

LIST OF ACTION ARGUMENTS

These control arguments control various memo options. The -delete, -list, -postpone, -print, and -process actions are mutually exclusive.

-brief, -bf
does not print message "No memos." if no memos are found.

This page intentionally left blank.

-delete {-force}, -dl {-fc}
deletes all memos selected by the optional arguments. You must explicitly supply at least one memo. It queries you before deleting nonmature memos; however, with -force, it deletes memos—without querying you—even if they are not yet mature.

-list, -ls
prints text and control information of selected memos; no memos are executed. If you don't explicitly select memos, all memos are listed. If you also give -totals, only the total number of selected memos is printed.

-off
suppresses all memo alarms until the next memo command with no explicitly specified action. You can combine -on and -off with other actions.

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

-pathname -default, -pn -dft
resets the default memo segment to Person.ID.memo in your home directory.

-pathname path, -pn path
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 you supply -pathname along with -on or -off, the default memo segment is changed and alarms are turned on or off, as appropriate, for the new segment. You need not give the suffix .memo.

-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. You must explicitly provide at least one memo. (See "Notes.")

-print, -pr
prints text of all selected memos. No memos are executed. If you don't explicitly select memos, 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 explicitly specifying no action.

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

-totals, -tt
you can only use it together with -list. When you give it, the total number of memos selected is printed, rather than listing each of the memos.

LIST OF SELECTION_ARGS

These arguments are used to select memos to be listed, printed, deleated, or postponed. You can also use some memo_options to specify types of memos to be selected (see "Notes"). When you supply more than one selection_args, only those memos that match all the selection criteria are chosen.

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

-from DT, -fm DT
selects all memos that mature on, or after, DT. You can combine it with -to, but specify each only once. It is incompatible with -date and -time. (See "Notes.")

-match STR
specifies a string against which memo texts are matched to select memos. STR cannot be longer than 32 characters. You can supply up to 40 STRs; all memos that match at least one are selected.

-to DT
selects all memos that mature on, or before, DT. This control argument is incompatible with -date and -time. (See "Notes.")

NOTES

See Section 1 for a description of valid DT values.

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

You can use the memo_options to specify types of memos to be selected; those that take a date/time interval (-repeat, -expires, but not -date or -time) cause the selection of all repeating or expiring memos, as the time interval (which you must specify) 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: menu_create

SYNTAX AS A COMMAND

```
menu_create menu_name {-control_args}
```

FUNCTION

creates a menu description, assigns it a specified name, and stores it in a segment. The menu description can be used with other menu commands, active functions, and subroutines.

ARGUMENTS

menu_name
is the name assigned to the menu when it is stored.

CONTROL ARGUMENTS

-brief, -bf
creates the segment given by -pathname without querying you.

-header STR, -he STR
specifies a line of header. All header lines supplied appear in the menu in the order given. Quote STR if it contains blanks or special characters.

-option STR, -opt STR
specifies a menu option. The options appear in the menu in the order given. Supply at least one option. Quote STR if it contains blanks or special characters.

-pathname PATH, -pn PATH
is the pathname of the value segment in which the menu is stored. The value suffix is assumed. If the value segment selected does not exist, you are asked if you want to create it (unless you have used -brief). If you omit -pathname, your default value segment (>udd>Project_id>Person_id>Person_id.value) is used to store the menu.

-trailer STR, -tr STR
specifies a trailer line. All trailers appear in the menu in the order given. Quote STR if it contains blanks or special characters.

LIST OF FORMAT CONTROL ARGUMENTS
The following control arguments manipulate the format of the menu:

-center_headers, -ceh
centers all header lines.

-center_trailers, -cet
centers all trailer lines.

-columns N, -col N
sets the number of columns in the menu to N, where N is a positive decimal integer. (Default: one column)

-line_length N, -ll N
specifies the line length for the menu, where N is a positive decimal integer. If not supplied, the line length is your terminal's line length at the time you invoke menu_create.

-no_center_headers, -nceh
left-flushes header lines. (Default)

-no_center_trailers, -ncet
left-flushes trailer lines. (Default)

-option_keys STR, -okeys STR
specifies the keystrokes to be associated with each option. Each character in STR is associated with the corresponding option, so that if it is typed, the corresponding option is selected. There must be at least as many characters in STR as there are options. If you give no -option_keys, the string "123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" is used.

-pad C
specifies the padding character to be used for centering headers and trailers, where C is one character. (Default: space character)

ACCESS REQUIRED

You must have rw access on the value segment.

EXAMPLES

The following example sets up a small menu named compile:

```
! menu_create compile -pn [pd]>temp -pad = -he "SAMPLE MENU"
    -tr = -ceh -cet -columns 2 -ll 78 -opt "Compile with No Options"
    -opt "Symbol Table" -opt "Profile Info"
```

The menu looks like this:

```
==================================SAMPLE MENU===================================
 (1) Compile with No Options          (3) Profile Info
 (2) Symbol Table
================================================================================
```

Name: menu_delete

SYNTAX AS A COMMAND

```
menu_delete menu_name {-control_arg}
```

FUNCTION

deletes a menu description from a specified value segment.

ARGUMENTS

menu_name
is the name that was assigned to the menu when it was stored.

CONTROL ARGUMENTS

-pathname PATH, -pn PATH
is the pathname of the value segment in which the menu is stored. The value suffix is assumed. If you don't give it, your default value segment (>udd>Project_id>Person_id>Person_id.value) is searched for the menu.

Name: menu_describe

SYNTAX AS A COMMAND

```
menu_describe menu_name {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[menu_describe menu_name -control_args]
```

FUNCTION

prints or returns information about a menu.

ARGUMENTS

menu_name
is the name that was assigned to the menu when it was stored.

CONTROL ARGUMENTS

-count, -ct
returns the number of options defined in the menu.

-height
returns the height of the menu.

-pathname PATH, -pn PATH
is the pathname of the value segment in which the menu is stored. The value suffix is assumed. If you don't give it, your default value segment (>udd>Project_id>Person_id>Person_id.value) is searched for the menu.

-width
returns the width of the menu.

NOTES

As a command, any number of control arguments is allowed. If none are given, all attributes are displayed.

As an active function, one of -count, -height, or -width must be given.

Name: menu_display

SYNTAX AS A COMMAND

```
menu_display menu_name {-control_args}
```

FUNCTION

displays a menu in a window.

ARGUMENTS

menu_name
is the name that was assigned to the menu when it was stored.

CONTROL ARGUMENTS

-io_switch STR, -is STR
specifies the name of an I/O switch for a window. This serves to identify a window. (Default: user_i/o)

-pathname PATH, -pn PATH
is the pathname of the value segment in which the menu is stored. The value suffix is assumed. If not given, your default value segment (>udd>Project_id>Person_id>Person_id.value) is searched for the menu.

Name: menu_get_choice

SYNTAX AS A COMMAND

```
menu_get_choice menu_name {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[menu_get_choice menu_name {-control_args}]
```

FUNCTION

gets a menu choice from you and prints or returns it.

ARGUMENTS

menu_name
is the name assigned to the menu when it is stored.

CONTROL ARGUMENTS

-default_fkeys STR, -dfkeys STR
specifies the keys to be used if the terminal does not have function keys or the proper set of function keys. (See "Notes on Function Keys" below.)

-function_keys STR, -fkeys STR
specifies the keys to be used to simulate function keys. This control overrides any function key definitions already established for the terminal. (See "Notes on Function Keys.")

-io_switch STR, -is STR
specifies the name of an I/O switch for a window. (Default: user_i/o)

-pathname PATH, -pn PATH
is the pathname of the value segment in which the menu is stored. The value suffix is assumed. If not given, your default value segment (>udd>Project_id>Person_id>Person_id.value) is searched for the menu.

NOTES

Many terminals have function keys. On many of these terminals (such as the Honeywell VIP7801), they are labeled "F1", "F2", etc. If you type one of these function keys, menu_get_choice returns the string "F*", where * is a one- or two-digit number signifying which function key is pressed. It is possible to specify your own set of keystrokes to be used instead of the terminal's function keys (-function_keys), or to specify a set of keystrokes to be used if the terminal does not have enough function keys (-default_fkeys). Each character in the string simulates a function key: The first one simulates function key 0, the next, function key 1, etc. To simulate a given function key, type esc-C, where C is the character corresponding to the function key; thus, if the string is "0123456789", typing esc-2 returns F2.

Supply -function_keys to specify keystrokes to be used instead of any that might be defined for the terminal; if given, the simulation of function keys always takes place.

Give -default_fkeys if you want to use the terminal-defined function keys when possible, but wish to specify key sequences to be used to simulate function keys if necessary. Each character in the string following -default_fkeys corresponds to one function key. If the character is a space, it means it makes no difference if the terminal has a function key corresponding to that position. If the character is not a space, that character is employed to simulate a function key if necessary. If the terminal does not have a function key for every nonspace character in the string, then STR is used to simulate function keys. Thus, the string " ?p q" means that you do not care whether the terminal has a function key 0 or a function key 3, but you wish to use function keys 1, 2, and 4. If any of these three function keys is not present on the terminal, then esc-? substitutes for F1, esc-p substitutes for F2, and esc-q substitutes for F4.

Name: menu_list

SYNTAX AS A COMMAND

```
menu_list {menu_starname} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[menu_list {menu_starname} {-control_arg}]
```

FUNCTION

lists the names of the menu descriptions stored in a value segment.

ARGUMENTS

menu_starname
is a starname used to search for menu descriptions. If omitted, the default is **.

CONTROL ARGUMENTS

-pathname PATH, -pn PATH
is the pathname of the value segment in which the menu is stored. The value suffix is assumed. If not given, your default value segment (>udd>Project_id>Person_id>Person_id.value) is searched for the menu.

Name: merge_ascii, ma

SYNTAX AS A COMMAND

```
ma paths {-control_args}
```

FUNCTION

merges two or more related ASCII text segments.

ARGUMENTS

paths
: are pathnames of segments to be merged as automatically as possible. The equal and archive component pathname conventions are allowed. You can merge up to six segments, including those preceded by -edit.

CONTROL ARGUMENTS

-edit path
: merges the segment named path in a nonautomatic manner. Edit mode is entered each time a modification is found in the specified segment.

-minchars N
: specifies the minimum number of characters that must be identical for merge_ascii to assume blocks of text in different segments are identical. (Default: 25)

-minlines N
: specifies the minimum number of lines that must be identical for merge_ascii to assume blocks of text in different segments are identical. (Default: 2)

-old_original path, -old_orig path
: identifies path as the pathname of a segment antecedent to the most recent common ancestor of the texts being merged and allows the automatic picking up of identical changes present in all the texts being merged.

-original path, -orig path
: identifies path as the pathname of a segment containing the original version of the text. The proper original is the most recent common ancestor of the texts being merged. Overlapping changes, even if identical, enter edit mode.

-output_file path, -of path
: puts the merged output text in the segment named path. The archive component pathname convention is not allowed.

NOTES

This command is typically used to merge texts that have been independently modified by several users. If an original version of the text is available, and if you desire, merge_ascii performs the merge automatically, requiring your intervention only when it detects overlapping modifications. When your intervention is required, merge_ascii displays line-numbered blocks of text and then enters edit mode allowing you to choose lines from any text or insert new ones.

When blocks of text are displayed, each line is preceded by a text identifier and a line number. The text identifier A is reserved for the original, whether supplied or not. The identifiers B–G are assigned to the texts being merged in the order in which their pathnames are encountered on the command line. The identifier M is used for the merged output if printed while in edit mode.

You can use either -original or -old_original to enable automatic merging. If you supply neither, edit mode is entered each time differences are found in the segments being merged. Use -old_original judiciously, only if appropriate and when you fully understand the relationships between the texts being merged.

LIST OF EDIT REQUESTS

In the syntax of the edit requests, <text_id> is the lowercase letter corresponding to the text identifier used by merge_ascii and <line_no> is a line number in the text segment. You can specify line numbers as "<" to address the first line or as ">" to specify the last line of a current block.

<text_id>k
copy current block from specified text (e.g., bk copies current block from text B).

<text_id><line_no>k
copy specified line from specified text (e.g., b5k copies line 5 from text B).

<text_id><line_no>,<line_no>k
copy specified lines from specified text (e.g., b4,7k copies lines 4 through 7 from text B).

<text_id>p
print current block from specified text (e.g., bp prints current block from text B).

<text_id><line_no>p
print specified line from specified text (e.g., b6p prints line 6 from text B).

<text_id><line_no>,<line_no>p
print specified lines from specified text (e.g., b12,16p prints lines 12 through 16 from text B).

`<text_id>d`
delete the current block in specified text (e.g., md deletes the current block in text M).

`input`
enter input mode.

return from input mode to edit mode.

`go`
exit editor and continue comparison.

`quit`
abort merge and return to command level. If you give this request during a merging procedure, all work is lost; work is not saved unless merging is done from the beginning to the end of the segments.

`e`
execute rest of line as a Multics command line.

`x`
display identifiers, current line numbers, and pathnames of each text.

`help`
print a list of the edit requests and a brief explanation of each one.

NOTES ON EDIT REQUESTS

In any invocation of edit mode the current block in each text is just the block of lines previously displayed. The current block in text M is initially empty and grows as you select or input lines (see "Examples" below).

The print (p) and copy (k) requests can address any lines in any text (A to M) known to merge_ascii. The delete (d) request can only be applied to the current block in text M and has the effect of undoing all edit requests made since changes were last displayed.

You can give multiple edit requests, delimited by blanks, on a single request line; however, quit, go, input, and e must not be followed by other requests.

EXAMPLES

The command line

```
! ma -orig pathA pathB pathC -of pathM
```

automatically merges the contents of pathB and pathC into pathM. Because you supplied an original version (pathA), all nonoverlapping changes in pathB and pathC are placed in pathM.

The command line

```
! ma pathB pathC -of pathM
```

performs a nonautomatic merge on the contents of pathB and pathC, displays all differences, and enters edit mode. This type of merging is typically used when there is no original segment.

The command line

```
! ma -orig pathA -edit pathB -edit pathC -of pathM
```

performs a nonautomatic merge, but gives you information about the contents of the original text. In this case, although an original segment exists, you want complete control over what goes into the output segment.

The command line

```
! ma -original pathA pathB -edit pathC -of pathM
```

performs a merge in which changes found in pathC enter edit mode. Nonoverlapping changes in pathB are picked up and automatically placed in the output segment. This combination of control arguments is useful when you are familiar with the changes present in pathB but wishes to review changes present in pathC before picking them up.

The command line

```
! ma -old_orig pathA pathB pathC -of pathM
```

merges pathB and pathC automatically assuming that pathA is an earlier version of the text than the most recent common ancestor of pathB and pathC. Changes present in both pathB and pathC are picked up automatically. You can use -old_original to obtain an automatic merge if pathA is a true original but some changes have been applied to both pathB and pathC. If you give -old_original and pathA contains changes not present in both pathB and pathC, then the resulting output segment is nearly always useless to you.

Name: message_status, msgst

SYNTAX AS A COMMAND

```
msgst {mbx_specification} {-control_arg}}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[msgst {mbx_specification}]
```

FUNCTION

prints information about mailboxes on which messages are being accepted.

ARGUMENTS

mbx_specification
specifies the mailbox on which messages are being accepted. If not given, the user's default mailbox (>udd>Project>Person>Person.mbx) is used.

CONTROL ARGUMENTS

-all, -a
prints information for all mailboxes on which the user is accepting or deferring messages.

LIST OF MBX SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table.

STR
is any noncontrol argument and is first interpreted as -mailbox STR; if no mailbox is found, STR is then interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

NOTES

As an active function, msgst returns the command string that you can use to set the message acceptance state on the specified mailbox to the current state. Thus you can push and pop multiple acceptance states using the value segment:

```
value_set old_state ||[msgst] -push
dm
value_set old_state ||[msgst] -push
am -call "ec message_handler"
[value_get old_state -pop]
[value_get old_state -pop].
```

("||" means the returned string is not to be rescanned for command language special characters; the returned value is a single token.)

This page intentionally left blank.

Name: micro_transfer, mt

SYNTAX AS A COMMAND

```
mt path {-control_args}
```

FUNCTION

transfers files between a Multics system and a remote microcomputer (personal computer) using either 1) the xmodem protocol or 2) the IBM PC-to-Host protocol. Your terminal must be connected to the system through the tty_ I/O module. (This is the usual method of connecting terminal to the Multics system.)

ARGUMENTS

path
can be one of the following:

1) is the pathname of the source segment on Multics when transferring files from Multics to a microcomputer.

2) is the pathname of the target segment on Multics when transferring files to Multics from a microcomputer.

CONTROL ARGUMENTS

-attach_description STR, -atd STR
specifies the I/O module to be used to implement the file transfer, where STR specifies the I/O module to be used and the I/O switch to which the module is to be attached. Enclose STR in quotation marks if it contains spaces or other command language characters. STR can be one of the following:

xmodem_io_ user_i/o
specifies the XMODEM protocol is to be used. (Default)

ibm_pc_io_ user_i/o
specifies the IBM PC-to_Host protocol is to be used.

-eof STR
specifies the end-of-file sequence for the microcomputer, where STR is the end-of-file character. STR can be a printable ASCII character or a control character; you must express the latter type in the octal equivalent and surround it by quotation marks. When transmitting a file to a microcomputer the end-of-file character is transmitted as STR; when receiving a file from a microcomputer the occurrence of STR indicates the end-of-file to Multics.

-eol STR
specifies the end-of-line sequence for the microcomputer, where STR is the end-of-line character. STR can be a printable ASCII character or a control character; you must express the latter type in the octal equivalent and surround it by quotation marks. When transmitting files to a microcomputer, each linefeed character is translated to STR; when receiving files from a microcomputer, each occurrence of STR is translated to a linefeed character.

-modes STR
sets the modes for file transfer according to STR, which is a string of mode names separated by commas. You can optionally precede many modes by ^ to turn the specified mode off. Modes not specified in STR are left unchanged. Modes are restored to their original value after the file transfer is complete. (See set_tty for a list of valid modes; see "Notes on Data Transfer I/O Modules" for the default modes.)

-receive
receives data from the microcomputer. Give either -receive or -send.

-send
sends data to the microcomputer.

NOTES ON DATA TRANSFER I/O MODULES

The micro_transfer command provides an interface between the Multics file system and a data transfer protocol. The data transfer protocol is implemented as an I/O module. Such I/O modules must specify a target I/O switch, and they must support the stream_input and stream_output opening modes. The switch identified by the switch argument must be open for stream_input_output. The following I/O modules are currently available for use with micro_transfer:

xmodem_io_
uses the XMODEM data transfer protocol. The default mode string used by micro_transfer for file transfer is: "no_outp,8bit,breakall,^echoplex,rawi,^crecho,^lfecho,^tabecho,rawo"

ibm_pc_io_
uses the IBM PC-to-Host data transfer protocol. This protocol does not transfer binary data or check for errors. The default mode string used by micro_transfer for file transfer is: "^8bit,breakall,^echoplex,rawi,^crecho,^lfecho,^tabecho,rawo"

Users writing their own data transfer protocol I/O modules with micro_transfer may do so. Its descriptions would be:

XXX
uses the user-specified data transfer protocol for the file transfer. The default mode string used by micro_transfer for file transfer is: "no_outp,8bit,breakall,^echoplex,rawi,^crecho,^lfecho,^tabecho,rawo"

NOTES ON FILE TRANSFER SPEED

There is no guarantee of any particular line speed when transferring files between Multics and a microcomputer. Line speed is dependent on the microcomputer and the load of the FNP and communication system for Multics. Due to the nature of the XMODEM and IBM PC-to-Host protocols, files may not be successfully transferred to Multics over high-speed lines. The actual limit depends on the site configuration and current load. The Video System shold not be used when using micro_transfer.

PROCEDURE FOR USING MICRO_TRANSFER

Use the following procedure to transfer files between Multics and a microcomputer with micro_transfer:

1. Invoke the control program on the microcomputer. This program is a terminal emulator and file transfer program.

2. Connect to Multics by issuing the appropriate command to the microcomputer. To find out which command to use, refer to the manual that documents the microcomputer's file transfer protocol. Once connected, the standard Multics banner is displayed.

3. Login to Multics.

4. Issue micro_transfer on Multics specifying the pathname on Multics and the applicable control arguments.

5. Escape now back to the microcomputer. The escape sequence used depends on the microcomputer. To find out which escape sequence to use, refer to the manual that documents the microcomputer's file transfer protocol. Upon return to the microcomputer, enter the type and direction of the file transfer and the microcomputer file name. This must correspond to the type and direction specified on Multics. For example, if you used micro_transfer -send, the command used for receiving a file transfer must be executed on the microcomputer.

6. The file transfer begins. A display indicating the status of the transfer may or may not occur, depending on the communications package residing on the microcomputer.

7. At the end of the transfer, the microcomputer returns to the communications command level.

EXAMPLES

```
micro_transfer foobar.foo -send -atd "ibm_pc_io_ user_i/o" -eol "\015"
```

Name: min

SYNTAX AS A COMMAND

```
min num_args
```

SYNTAX AS AN ACTIVE FUNCTION

```
[min num_args]
```

FUNCTION

returns the minimum of the numeric arguments passed to it.

EXAMPLES

```
!  string [min 3 -4]
   -4
```

Name: minus

SYNTAX AS A COMMAND

```
minus {numA {numB}}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[minus {numA {numB}}]
```

FUNCTION

returns the result of numA minus numB. If numB is not specified, the negative of numB is returned. If you give no arguments, returns 0.

EXAMPLES

```
!  string [minus 3.5 3]
   0.5
!  minus 5
   -5
!  minus -6
   6
!  minus
   0
```

Name: minute

SYNTAX AS A COMMAND

```
minute {time_string} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[minute {time_string} {-control_arg}]
```

FUNCTION

returns the one- or two-digit number of a minute of the hour, from 0 to 59. The format string to produce this is "^Z9MH".

ARGUMENTS

time_string
: indicates the minute about which information is desired. If you supply no time_string, the current minute is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-zone STR
: STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default zone. Use the display_time_info command to display a list of all acceptable zone values.

Name: mod

SYNTAX AS A COMMAND

```
mod numA numB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[mod numA numB]
```

FUNCTION

returns the remainder of numA divided by numB (numA modulo numB).

EXAMPLES

```
! string [mod 4. 3]
   1
! string [mod 4.5 3.5]
   1
```

Name: monitor_quota

SYNTAX AS A COMMAND

```
monitor_quota {-control_arguments}
```

FUNCTION

calculates storage of a directory and sends a warning message when approaching a record quota overflow condition.

CONTROL ARGUMENTS

-call STR {N}
: passes STR to the command processor as a command when a directory's segment quota used is found to be greater than 90 percent (default) of the quota assigned. If you give N, the default is overridden. (See "Notes" below.)

-console {N}
: sends a warning of an approaching record quota overflow condition to the system console. You require access to the phcs_ gate to issue warnings on the system console. If you specify N, the default percent value at which the warning is to be issued (as given in the functional description) is overridden.

-off
: turns off all monitoring in the current process. You cannot use -off with any other control arguments.

-pathname, -pn
: is the pathname of the directory to be monitored. You can give only one path. (Default: your working directory)

-repeat DT, -rpt DT
: identifies the interval for setting the monitor time. It overrides the default time calculation. DT is a relative time >= 1 minute and acceptable to convert_date_to_binary_ (e.g., 10min, 1hr).

-warn Person_id.Project_id {Person_id.Project_id...} {N}
sends the warning message to Person_id.Project_id. You can list up to 10 users. You get a message by default if you omit -warn and -console. If you give N, the default percent value at which the warning is to be issued is overridden.

NOTES

You can use monitor_quota several times in a process to monitor different directories.

The number of records given with -call, -console, and -warn must be less than the quota assigned to the directory. The default interval when you invoke monitor_quota without -repeat is set with a time interval dependent on storage availability: if the directory is 50 percent full, an alarm is set to trigger in 30 minutes to check again; if 80 percent full, a message is sent and an alarm time of two minutes is set; if 90 percent, a warning is sent every minute, and if you provide -call the specified string is passed to the command processor.

Name: month

SYNTAX AS A COMMAND

```
month {time_string} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[month {time_string} {-control_arg}]
```

FUNCTION

returns the one- or two-digit number of a month of the year, from 1 to 12. The format string to produce this is "^Z9my".

ARGUMENTS

time_string
indicates the month about which information is desired. If you supply no time_string, the current month is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-zone STR
STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default zone. Use the display_time_info command to display a list of all acceptable zone values.

Name: month_name

SYNTAX AS A COMMAND

```
month_name {time_string} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[month_name {time_string} {-control_args}]
```

FUNCTION

returns the full name of a month of the year (e.g., "June"). The format string to produce this is "^mn".

ARGUMENTS

time_string
: indicates the month about which information is desired. If you supply no time_string, the current month is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-language STR, -lang STR
: STR specifies the language in which month name, day names, and zone names are to be expressed. (Default: the process default)

-zone STR
: STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default language and zone. Use the display_time_info command to display a list of all acceptable language and zone values.

Name: move, mv

SYNTAX AS A COMMAND

```
mv move path1 {path2...path1N path2N} {-control_arg}
```

FUNCTION

moves a specified segment or multisegment file (along with its access control list and all names) 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 you don't give the last path2 segment, path1 is moved to your 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

You need read access for path1, status and modify permission for the directory containing path1, and status, modify, and append permission for the directory containing path2.

NOTES

The default for chasing links depends on path1: if it is a star name, links are not chased by default; if it is not, links are chased.

If the primary name of path1 is the only one, it is added as a secondary name to path2.

If an entry with the entryname path1 already exists in the target directory, you are asked whether the already-existing entry should be deleted. If you answer "no," the move does not take place.

If path1 is protected by the safety switch, you are asked whether you want to delete path1 after it has been moved.

EXAMPLES

The command line

```
! move alpha >Verdi>= >Verdi>beta b
```

moves alpha from the current working directory to the directory >Verdi, keeping the name alpha, and moves beta from the directory >Verdi to the current working directory with the names b and beta.

Name: move_abs_request, mar

SYNTAX AS A COMMAND

```
mar request_identifiers {-control_args}
```

FUNCTION

moves a request from one absentee queue to another. The request is always placed at the end of the target queue.

ARGUMENTS

request_identifiers
you can specify them in one of the following forms:

path
is the full or relative pathname for the absentee input segment of requests to be moved. The star convention is allowed.

-entry STR, -et STR
identifies requests to be moved 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 moved by request_identifier. You can use this identifier to further define any path or -entry identifier (see "Notes").

CONTROL ARGUMENTS

-all, -a
searches all queues for the requests to be moved. The target queue is not searched by -all if the source and target request types are identical. This control argument is incompatible with -queue.

-brief, -bf
suppresses messages telling that a particular request_id was not found or which requests were moved when using star names or -all.

-foreground, -fg
moves the requests contained in the foreground queue.

-queue N, -q N
specifies that queue N for the given request type contains the request to be moved, where N is an integer specifying the number for the queue. If you omit -queue, all the queues are searched.

-sender STR
moves only requests from sender STR. You must give one or more request identifiers.

-to_queue N, -tq N
specifies which queue to move the request to. (Required)

-user User_id
is a character string giving the name of the submitter of the request if not equal to the group ID of the process. This control argument is primarily for operators and administrators. User_id can be Person_id.Project_id, Person_id, or .Project_id. You need both r and d extended access to the queue. This control argument causes the command to use privileged message segment primitives that preserve the original identity of the submitter. You need the AIM ring_1 privilege to preserve the original AIM attributes. If ring_1 privilege is not present, your AIM attributes are used. (Default: only requests entered by you are moved)

ACCESS REQUIRED

You 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. You must have r and d extended access to move a request owned by another.

NOTES

If you give any path or -entry, only one -id is accepted and it must match any requests selected by path or -entry.

You can supply multiple -id identifiers in a single invocation only if you give no path or -entry.

When you use no star names 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 there are.

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

Name: move_daemon_request, mdr

SYNTAX AS A COMMAND

```
mdr request_identifiers {-control_args}
```

FUNCTION

moves a request from one I/O daemon queue to another. The move can be within the same request type or from one request type to another. The request is always placed at the end of the target queue.

ARGUMENTS

request_identifiers
Can be specified in one of the following forms:

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. You can use this identifier to further define any path or -entry identifier (see "Notes").

CONTROL ARGUMENTS

-all, -a
searches all queues for the requests to be moved. The target queue is not searched by -all if the source and target request types are identical. This control argument is incompatible with -queue.

-brief, -bf
suppresses messages telling you 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 given request type contains the request to be moved, where N is an integer specifying the number for the queue. If you omit -queue, all the queues are searched.

-request_type STR, -rqt STR
specifies that the request moved is found in the queue(s) for the request type identified by STR. If this control argument is not specified, the default request type is "printer". Request types can be listed by the print_request_types command.

-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

You 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. You must have r and d extended access to move a request owned by another user (see -user).

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.

EXAMPLES

To move from every queue, to queue 1, in the default request_type all requests where the last component of the pathname matches "list", type

```
! mdr -et *.list -tq 1 -all

   Daemon request mydir.list moved from queue 2 to queue 1.
   Daemon request myseg.list moved from queue 3 to queue 1.
```

Name: move_dir, mvd

SYNTAX AS A COMMAND

```
mvd source_dir {target_dir} {entry_type_keys} {-control_args}
```

FUNCTION

moves a directory and its subtree, including all of the associated attributes, to another point in the hierarchy.

ARGUMENTS

source_dir
: is the pathname of the directory to be moved. The star convention is allowed to match directory names. Matching names associated with other storage types are ignored. The source_dir cannot be contained in target_dir.

target_dir
: is the new pathname for source_dir. If the entryname is different from one already on source_dir, it is added to the existing names. If target_dir is not specified, source_dir is moved to the working directory and given the same entryname. The equal convention is allowed.

CONTROL ARGUMENTS

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

-force
: continues execution when target_dir already exists, without asking you. If you don't supply -force, you are queried.

-replace, -rp
: deletes the contents of target_dir existing before the copying begins. If target_dir is non-existent or empty, this control argument has no effect. The default is to append the contents of the source directory to the target directory if it already exists.

LIST OF ENTRY_TYPE_KEYS

These keys control what type of storage system entry is moved. The default is to move all entries. The keys are

-branch, -br
-directory, -dr
-file, -f
-link, -lk
-multisegment_file, -msf

-non_null_link, -nnlk
-segment, -sm

If one or more entry_type_keys are specified, but not the -directory key, the subtree of source_dir will not be followed.

ACCESS REQUIRED

Status and modify permission are required for source_dir and all of the directories in its tree, and its containing directory. If target_dir does not exist, append permission is required for its containing directory. If it does exist, modify and append permission for target_dir are required. This command does not force access.

The access control list associated with source_dir is moved to target_dir.

NOTES

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

Links are translated; that is, if there are references to a source directory in a link pathname, the link pathname is changed to refer to the target directory.

If part of the tree is not moved, problems with link translation may occur. If the target of the link in the source_dir tree was in the part of the tree not moved, there may be no corresponding entry in the target_dir tree. Hence, translation of the link (presumably originally nonnull) will cause the link to become null.

See also the copy, move, and copy_dir commands.

EXAMPLES

If the working directory is >udd>Project>Smith, the command line

```
! mvd source_dir new>target_dir -rp
```

moves the directory named >udd>Project>Smith>source_dir and its subtree to >udd>Project>Smith>new>target_dir replacing its contents with the contents of source_dir.

Name: move_names

SYNTAX AS A COMMAND

```
move_names from_path1 {to_path1...from_pathN to_pathN}
```

FUNCTION

moves all the names but the one used to designate the entry from one entry (directory, segment, multisegment file, data management file, extended entry, or link) to another. To copy the alternate names, see copy_names.

ARGUMENTS

from_pathi
: is the pathname of the entry whose names are to be moved. The star convention is not allowed.

to_pathi
: is the pathname of the entry to which names on from_pathi are to be moved. The equal convention is allowed. If you omit the last to_path, the entry with the same name as from_path in your working directory is assumed.

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. The target queue is not searched by -all if the source and target request types are identical. This control argument is incompatible with -queue.

-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 given request type contains the request to be moved, where N is an integer specifying the number for the queue. If you omit -queue, all the queues are searched.

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

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

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

-request_type STR, -rqt STR
specifies that the request moved is found in the queue(s) for the request type identified by STR. Use the print_request_types command to list request types. (See "Notes.")

-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 you don't give -to_request_type, 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 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 1 privilege, the AIM attributes of the original submitter are preserved; otherwise the AIM attributes of the current process are used. (Default: to move only requests entered by the user executing the command)

ACCESS REQUIRED

You 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. You must have r and d extended access to move a request owned by another user (see -user).

NOTES

The control arguments -print, -punch, -plot, and -request_type are mutually exclusive. If you use none, the default request type for enter_output_request -print (as displayed by print_request_types) is assumed.

If you supply any path or -entry STR request identifiers, only one -id ID request identifier are accepted and it must match any requests selected by path or entryname. You can specify multiple -id ID identifiers in a single command invocation only if you give no path or entry request identifiers.

When you use no star names 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.

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

Name: move_quota, mq

SYNTAX AS A COMMAND

```
mq path1 quota_change1...{pathN quota_changeN}
```

FUNCTION

allows you to move records of quota between two directories, one immediately inferior to (contained in) the other.

ARGUMENTS

pathi
: is the pathname of a directory. The quota change occurs between this branch and its containing directory. A pathi of -working_directory (-wd) specifies your working directory. You can't use the star convention.

quota_changei
: is the numbers of records to be moved between the immediately superior (containing) directory quota and the pathi quota. This argument can be either positive or negative. If it is positive, the quota is moved from the containing directory to pathi; if negative, the move is from pathi to the containing directory.

ACCESS REQUIRED

You must have modify permission on both directories.

NOTES

After the change, the quota on pathi must be greater than or equal to the number of records used in pathi unless the change makes the quota zero. If the change makes the quota on pathi zero, there must be no immediately inferior directory with nonzero quota and the records used and the record-time product for pathi are reflected up to the superior directory.

If pathi is an upgraded directory (its access class is greater than the one of its containing directory), quota_changei must be positive. You can move quota back to the containing directory of an upgraded directory only by deleting the latter.

You can't move quota between a master directory and its containing directory.

EXAMPLES

The command line

```
! mq >udd>m>WShakespeare>sub1_dir 1000
```

adds 1000 records to the quota on >udd>m>WShakespeare>sub1_dir and subtracts 1000 records from the quota on >udd>m>WShakespeare.

The command line

```
! mq >udd>m>JJoyce>sub1_dir>sub2_dir -50
```

subtracts 50 records from the quota on >udd>m>JJoyce>sub1_dir>sub2_dir and adds 50 records to the quota on >udd>m>JJoyce>sub1_dir.

Name: msfs

SYNTAX AS A COMMAND

```
msfs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[msfs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of multisegment files (MSFs) that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: mtape_delete_defaults

SYNTAX AS A COMMAND

```
mtape_delete_defaults OPN {-control_args}
```

FUNCTION

deletes default arguments set by the mtape_set_defaults command. It deletes the default arguments, from a specified value segment, that are associated with a given volume type and tape-processing operation.

ARGUMENTS

OPN
is the type of tape operation (attach, open, close, or detach) that uses the default arguments.

CONTROL ARGUMENTS

-pathname path, -pn path
specifies the pathname of a value segment to be searched for default arguments. If you omit it, the value segment >udd>[user project]>[user name]>[user name].value is used.

-volume_type VT, -vt VT
specifies the volume type (ansi or ibm) used by mtape_ to select the per-format module for tape processing. Give -vt when you supply either the open or close operations; omit it when you supply either the attach or detach operations.

EXAMPLES

Suppose you want to delete all defaults in the value segment >udd>[user project>[user name]>[user name].value that apply to the close operation for ANSI tapes:

```
! mtape_delete_defaults close -vt ansi
```

Now you want to delete all the defaults in the value segment >udd>m>slk>defaults.value that pertain to the detach operation:

```
! mtape_delete_defaults detach -pn >udd>m>slk>defaults.value
```

In this case you must give the pathname. The volume type is not given because attach and detach are general operations that are applicable to all volume types.

Name: mtape_get_defaults

SYNTAX AS A COMMAND

```
mtape_get_defaults OPN {-control_args}
```

FUNCTION

prints default arguments set by the mtape_set_defaults command. It prints the default arguments, stored in a specified value segment, that are associated with a given volume type and tape-processing operation.

ARGUMENTS

OPN
is the type of tape operation (attach, open, close, or detach) that uses the default arguments.

CONTROL ARGUMENTS

-pathname path, -pn path
specifies the pathname of a value segment to be searched for default arguments. It is incompatible with -usl.

-use_search_list, -usl
specifies that each value segment in the mtape_arguments search list is to be searched for default arguments and that the final default linear form that the mtape_ argument-processing subroutine uses is to be printed. (Default)

-volume_type VT, -vt VT
specifies the volume type (ansi or ibm) used by mtape_ to select the per-format module for tape processing. Give -vt when you use either the open or close operations; omit it when you use either the attach or detach operations.

EXAMPLES

Suppose you want to print out the final form of all defaults that affect the open operation for IBM tapes, then

```
! mtape_get_defaults open -vt ibm -usl
```

prints out the default linear form associated with the open operation for IBM tapes.

If you then want to print out the defaults found in a particular value segment that affect the open operation for IBM tapes, the value segment being >udd>m>slk>defaults.value, type:

```
! mtape_get_defaults open -vt ibm -pn >udd>m>slk>defaults.value
```

Now you want to print out the defaults contained in the value segment >udd>[user project]>[user name]>[user name].value that affect all tape attachments:

```
! mtape_get_defaults attach -pn
    >udd>[user project]>[user name]>[user name].value
```

The volume type is not given because attach and detach are general operations that are applicable to all volume types. You must specify the pathname because the default operation is -usl.

Finally, if you want to scan all the value segments in your mtape_arguments search list for default arguments associated with the detach option, type:

```
! mtape_get_defaults detach -pn ([psp mtape_args])
```

Since the print_search_paths active function returns all pathnames in the search list, the default arguments associated with the detach operation in each value segment in the mtape_arguments search list are printed.

Name: mtape_set_defaults

SYNTAX AS A COMMAND

```
mtape_set_defaults OPN -control_args
```

FUNCTION

sets default arguments used by the mtape_ I/O module.

ARGUMENTS

OPN
: is the type of tape operation (attach, open, close, or detach) that uses the default arguments.

CONTROL ARGUMENTS

-arguments ARGS, -argument, -ag ARGS
: are the arguments appropriate to the specified operation and tape format. They must be syntactically correct and appropriate to the conditions under which they are applied. (Required; it must be the last one specified on the command line)

-pathname path, -pn path
: is the name of the value segment in which the requested default values are stored. If you omit it, the value segment >udd>[user project]>[user name]>[user name].value is used.

-volume_type VT, -vt VT
specifies the volume type (ansi or ibm) used by mtape_ to select the per-format module for tape processing. Give -vt when you specify either the open or close operations; omit it when you supply either the attach or detach operations. This control argument, along with -operation, defines restrictions on the specification of the default arguments (see the mtape_ I/O module in the Subroutines manual).

NOTES

The command sets the default arguments associated with a given volume type and tape-processing operation. These default arguments are eventually used to complete attach, open, close, and detach descriptions when you have not explicitly supplied all the necessary information.

The default arguments specified in the command line are processed by the mtape_ argument-processing subroutine to assure that they follow all the restrictions imposed by their intended future usage. The result of this processing is then converted to a character string and is stored in the data space of a specified value segment. The stored value is later located and used as default information for argument processing when tapes are being processed by the mtape_ I/O module (see the Programmer's Reference Manual for tape processing).

NOTES ON USING DEFAULT ARGUMENTS

When it is necessary to use default arguments in a particular application, they are located by the mtape_ argument-processing routine using the mtape_arguments search list. The default mtape_arguments (mtape_args) search list, is as follows:

```
mtape_arguments
mtape_args
   >udd>[user project]>[user name]>[user name].value
   >site>mtape_arguments.value
   >sss>mtape_arguments.value
```

You can add or delete search paths as necessary using the search paths commands.

In locating default arguments, the mtape_ argument-processing routine looks in every value segment in the search list and takes the appropriate default arguments from each (if it finds them). Whether a group of default arguments is determined to be appropriate for an application depends on the volume type and tape-processing operation for which is intended.

Default arguments from value segments at the top of the search list take precedence over those from value segments at the bottom. Equivalently arguments on the right side of an argument list take precedence over arguments on the left. This means that after all default arguments for a particular application have been gathered from the search list, if an argument occurs more than once, the argument with the highest precedence is retained and the others are excluded. The result is called the default linear form.

EXAMPLES

Suppose you want to set some defaults (using mtape_) for attaching tapes:

```
! mtape_set_defaults attach -ag -track 9 -density 6250 -ring -display
```

Assuming that you have retained the default mtape_arguments search paths (as described above), these default arguments have the highest level of precedence for the attach operation. Now suppose that the >site and >sss mtape_arguments.value segments contain defaults for the attach operation as follows:

```
>site contains "-density 800 -track 7 -no_ring"
>sss contains "-density 1600 -track 9 -no_system -no_ring"
```

Using the precedence rules, the default linear form that results is

```
-no_system -track 9 -density 6250 -ring -display
```

The control argument -no_ring does not appear in the linear form because it has been excluded by -ring.

If you specify the following mtape_ attach description:

```
mtape_ volume_name -vt ansi
```

the default linear form is used intact (as previously specified). You can, however, override anything in the default linear form by respecifying it in the attach description. For example, if you want to override the default density of 6250 with a density of 1600, you specify it in the attach description this way:

```
mtape_ volume_name -vt ansi -density 1600
```

Arguments actually specified by you in an attach, open, close, or detach description always override those in the default linear form.

Name: nequal

SYNTAX AS A COMMAND

```
nequal numA numB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nequal numA numB]
```

FUNCTION

returns true if numA is numerically equal to numB, false otherwise.

EXAMPLES

```
string [nequal 5 5.0]
true
string [nequal 001 1]
true
string [nequal one 1]
Error
```

Name: network_request, nr

SYNTAX AS A COMMAND

```
nr 16_ftf source_path destination_path -net channel_name {-control_args}
```

FUNCTION

allows you to interactively transfer files to or from a DPS 6 X.25 Satellite.

ARGUMENTS

source_path
specifies the source file to be used for the transfer:
`{-name} file_name {-at net_address}`

* Precede file_name by -name (-nm) if it begins with a hyphen, enclose it in quotes if it contains spaces or special characters, and follow it by "-at net_address" if the file does not reside on the local host. Supply the file name in a syntax acceptable to the host on which the file resides. The net_address consists of the address of the DPS 6 on the X.25 connection (as specified in the DPS 6 CLM_USER file) followed by the end-task-ID of the DPS 6 Listener (as specified in the DPS 6 CLM file) in the X.25 directive. If the file resides on Multics, you can use an arbitrary star name; if it resides on a DPS 6, you can give the name "**" to transfer all the files in a directory on the Level 6.

destination_path
specifies the destination file to be used for the transfer:
`{-name} file_name {-at net_address}`

It has the same syntax and restrictions as source_path. You can use the equal convention.

CONTROL ARGUMENTS

-attended, -att
specifies that the DPS 6 already has a server running and no login dialogue is needed.

-brief, -bf
does not print messages as the command executes.

-data_type ascii
-data_type binary
-data_type bcd
specifies the data type of the Multics file. If binary, then the Multics file must be sequential or blocked; it can not be unstructured. (Default: ascii)

-long, -lg
prints a message when the transfer starts and when it is finished, giving the pathnames, records transfered, etc. (Default)

-network_name channel_name, -net channel_name
specifies the channel name of the X.25 channel (i.e., the network "name") to be used for the transfer. (Required)

-not_attended, -natt
specifies that a login dialogue is needed with the DPS 6 to initiate the transfer. (Default)

-password STR, -pw STR
specifies the password used by the remote host to authenticate the file transfer. If the remote host requires a password and none is given, you are prompted for one with a mask. (Default: none)

-user STR
STR specifies the user wishing the transfer. This can be used by the remote host for authentication of the file transfer. (Default: the Multics User_id of the user who submitted the request)

ACCESS REQUIRED

You must have the "dialok" attribute and have rw access to the X.25 channel specified by -net.

NOTES

Either the source file or the destination file must be on the local host (i.e., both must not use the -at argument); thus third-party transfers are not allowed.

EXAMPLES

```
! nr 16_ftf mult_file 16_file -at 99911 -net g.h102.*
```

where mult_file is the source on Multics, 16_file is the destination file on the DPS 6, and the connection between the DPS 6 and Multics is on channel g.h102. This example transfers mult_file to 16_file.

Name: new_proc

SYNTAX AS A COMMAND

```
new_proc {-control_arg}
```

FUNCTION

destroys your current process and creates a new one, using the control arguments given initially with login and -authorization.

CONTROL ARGUMENTS

-authorization STR, -auth STR
creates 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. (Default: to create the new process at the same authorization)

NOTES

Just before the old process is destroyed, the "finish" condition is signaled. 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 you had just logged in.

If your initial working directory contains the segment start_up.ec and you did not log in with -no_start_up, new_proc automatically issues the command line "exec_com start_up new_proc interactive" in the new process.

If your site is security conscious, it may have disabled "new_proc -auth"; in this case if you wish to change authorization, do this:

1. log out
2. verify, using terminal/modem indications, that the terminal has dropped DTR and that the system acknowledged by dropping DSR
3. log in at the new authorization.

This procedure is the only way to guarantee that you are communicating with the answering service and not with a Trojan horse.

DTR and DSR are EIA RS232 control signals that are part of the interface between your terminal and the system.

Name: ngreater

SYNTAX AS A COMMAND

```
ngreater numA numB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ngreater numA numB]
```

FUNCTION

returns true if numA is numerically greater than numB, false otherwise.

EXAMPLES

```
string [ngreater 5 8]
false
string [ngreater 9 4]
true
```

Name: nless

SYNTAX AS A COMMAND

```
nless numA numB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nless numA numB]
```

FUNCTION

returns true if numA is numerically less than numB; otherwise it returns false.

EXAMPLES

```
string [nless 8 4]
false
string [nless 4 8]
true
string [nless -5 -3]
true
```

Name: no_save_on_disconnect

SYNTAX AS A COMMAND

```
no_save_on_disconnect
```

FUNCTION

disables process preservation across hangups in your process, causing the process to log itself out automatically if its terminal channel hangs up.

NOTES

This command is meaningful only if process preservation was in effect for the process at login time, either by default or because you gave -save_on_disconnect on the login command line.

Name: nonbranches

SYNTAX AS A COMMAND

```
nonbranches star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nonbranches star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of nonbranches that match one or more star names.

ARGUMENTS

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

CONTROL ARGUMENTS

-absolute_pathname, -absp
    returns absolute pathnames rather than entrynames.

-chase
    processes the targets of links when you specify a starname.

-inhibit_error, -ihe
    returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
    does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
    signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: nondirectories, nondirs

SYNTAX AS A COMMAND

```
nondirs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nondirs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of segments, multisegment files (MSFs), and links that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

Only one name per entry is returned; i.e., if a segment, MSF, or link has more than one name that matches star_name, only the first match found is returned.

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

Name: nonfiles

SYNTAX AS A COMMAND

```
nonfiles star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nonfiles star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of directories and links that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

Only one name per entry is returned; i.e., if a directory or link has more than one name that matches star_name, only the first match found is returned.

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

Name: nonlinks

SYNTAX AS A COMMAND

```
nonlinks star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nonlinks star_names {-control_args}]
```

FUNCTION

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

ARGUMENTS

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

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames. (Default: to return entrynames)

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: nonmaster_directories, nmdirs

SYNTAX AS A COMMAND

```
nmdirs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nmdirs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of directories that are not master directories that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: nonmsfs

SYNTAX AS A COMMAND

```
nonmsfs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nonmsfs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of segments, directories, and links that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

Only one name per entry is returned; i.e., if a segment, directory, or link has more than one name that matches star_name, only the first match found is returned.

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

Name: nonnull_links, nnlinks

SYNTAX AS A COMMAND

```
nnlinks star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nnlinks star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of links for which the target entry exists that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: nonobject_files, nobfiles

SYNTAX AS A COMMAND

```
nobfiles star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nobfiles star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of files that are not executable object files and that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Segments and MSFs that you do not have at least r access to are ignored, since r access is needed to determine if the file is an object file.

Name: nonobject_msfs, nobmsfs

SYNTAX AS A COMMAND

```
nobmsfs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nobmsfs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of multisegment files (MSFs) that are not object MSFs and that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

Only one name per msf is returned; i.e., if an MSF has more than one name that matches star_name, only the first match found is returned.

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

MSFs that you do not have at least r access to are ignored, since r access is needed to determine if the file is an object MSF.

Name: nonobject_segments, nobsegs

SYNTAX AS A COMMAND

```
nobsegs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nobsegs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of segments that are not executable object segments and that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

ACCESS REQUIRED

You need at least r access to the segments.

NOTES

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

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

Name: nonsegments, nonsegs

SYNTAX AS A COMMAND

```
nonsegs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nonsegs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of directories, multisegment files (MSFs), or links that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

Only one name per entry is returned; i.e., if a directory, MSF, or link has more than one name that matches star_name, only the first match found is returned.

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

Name: nonzero_files, nzfiles

SYNTAX AS A COMMAND

```
nzfiles star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nzfiles star_names {-control_arg}s]
```

FUNCTION

returns the entrynames or absolute pathnames of files--segments and multisegment files (MSFs)--with a nonzero-bit count that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: nonzero_msfs, nzmsfs

SYNTAX AS A COMMAND

```
nzmsfs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nzmsfs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of multisegment files (MSFs) with a nonzero-bit count that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: nonzero_segments, nzsegs

SYNTAX AS A COMMAND

```
nzsegs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nzsegs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of segments with a nonzero-bit count that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: not

SYNTAX AS A COMMAND

```
not STR
```

SYNTAX AS AN ACTIVE FUNCTION

```
[not STR]
```

FUNCTION

returns false if STR is equal to true; true if STR is equal to false; otherwise prints an error message.

Name: nothing, nt

SYNTAX AS A COMMAND

```
nt {optional_args}
```

FUNCTION

performs a return to its caller and does nothing.

ARGUMENTS

optional_args
are optional arguments, which can have any value and are ignored.

NOTES

This command uses a special feature in the Multics linking mechanism that allows it to be executed by any reference name; thus you can use it as a "stub" procedure for testing the development of programs. To do this, initiate it with the reference name of the program it is supposed to replace. You can't use it in this fashion if the entrypoint name is different from the reference name.

Name: null_links, nlinks

SYNTAX AS A COMMAND

```
nlinks star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[nlinks star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of links for which the target does not exist that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Name: object_files, obfiles

SYNTAX AS A COMMAND

```
obfiles star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[obfiles star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of files that are executable object files and that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

Files that you do not have at least r access to are ignored, since r access is needed to determine if the file is an object file.

Name: object_msfs, obmsfs

SYNTAX AS A COMMAND

```
obmsfs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[obmsfs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of multisegment files (MSFs) that are executable object MSFs and that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

MSFs that you do not have at least r access to are ignored, since r access is needed to determine if the file is an object MSF.

Name: object_segments, osegs

SYNTAX AS A COMMAND

```
osegs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[osegs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of segments that are executable object segments and that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

ACCESS REQUIRED

You need at least r access to the object segments.

NOTES

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

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

Name: octal, oct

SYNTAX AS A COMMAND

```
oct values
```

SYNTAX AS AN ACTIVE FUNCTION

```
[oct values]
```

FUNCTION

returns one or more values in octal.

ARGUMENTS

value
is a value to be processed.

NOTES

The last character of the value indicates its type. Acceptable types are binary (b), quaternary (q), octal (o), hexadecimal (x), or unspec (u). Any valid PL/I real value is allowed. The absence of any specifier means decimal. The unspec value is limited to eight characters.

EXAMPLES

```
string [octal 1024]
2000

string [octal abcu]
141142143
```

Name: on

SYNTAX AS A COMMAND

```
on conditions handler_com_line {-control_args} subject_com_line
```

SYNTAX AS AN ACTIVE FUNCTION

```
[on conditions handler_com_line {-control_args} subject_com_line]
```

on

FUNCTION

establishes a handler for a specified set of conditions, executes an embedded command line with this handler in effect, and then reverts the handler. The handler is another embedded command line to be executed if the condition is signaled.

ARGUMENTS

conditions
is a list of condition names separated by commas to be trapped by the command.

handler_com_line
is the command line to be executed when one of the conditions contained in the list of condition names is raised. If handler_com_line contains spaces or other command language characters, enclose it in quotes. If no command is to be executed when a condition is raised, give handler_com_line as "".

subject_com_line
is the command line to be executed under the control of on; it consists of the remaining arguments. Quote it if it contains parentheses, brackets, quotes, or semicolons.

CONTROL ARGUMENTS

-brief, -bf
suppresses the comment printed when a condition occurs.

This page intentionally left blank.

-cl
establishes a new command level after the execution of handler_com_line. You cannot use it in the active function. The state of subject_com_line is preserved. If you issue the start command, the same action is taken as would have been had you not specified -cl.

-exclude STR, -ex STR
prevents on from trapping the conditions given in STR. If you list more than one condition, separate condition names by commas. This control argument is useful when handling the any_other condition.

-long, -lg
prints a detailed message describing the condition raised if one is available. This message is the same as the one printed by the reprint_error command.

-restart, -rt
continues execution of the subject_com_line after execution of handler_com_line or, if you also selected -cl, after execution of start. It is incompatible with -retry_command_line.

-retry_command_line, -rcl
aborts and executes over again subject_com_line after executing handler_com_line.

NOTES

The default action after executing handler_com_line is to abort the execution of subject_com_line.

If a condition is raised and trapped by on while executing the handler_com_line, it is considered a recursive signal and the entire invocation is aborted.

See the Programmer's Reference Manual for a list of standard system conditions.

NOTES ON ACTIVE FUNCTION

The active function returns "true" if any of the specified conditions are signaled during the execution of subject_com_line, "false" otherwise.

EXAMPLES

The command line

```
! on command_error "pwd;ls" -bf ws node la
```

does a walk_subtree starting at the node directory, listing the access of the working directory. If the list_acl (la) command fails because of insufficient access, for example, the pathname and contents of the working directory are printed and you are returned to command level since you didn't select -restart.

The command line

```
! on any_other -ex quit,program_interrupt,mme2 "ec dump" -lg myprog
```

executes the myprog command. If any condition except quit, program_interrupt, and mme2 is raised, on executes "ec dump", after printing a detailed explanation of the condition raised.

The command line

```
! on quit,mme2 db -bf -rt testcom
```

executes the testcom command, but responds to quits and breaks set in testcom by invoking debug; -rt continues execution of testcom after you quit out of debug; and -bf suppresses a warning message when one of the specified conditions is signaled.

In an exec_com, the command line

```
! on linkage_error "ec linkerr" ec recurse
```

calls a recursive entry point in the exec_com to continue execution, but with a linkage_error handler in effect. When linkage_error is signaled during the course of running recurse.ec, that exec_com is aborted and linkerr.ec is run.

The exec_com &if control line

```
&if &[on command_error "" -bf -rt command_name]
&then &quit
```

executes the command command_name. If the command_error condition is raised, the exec_com being executed is terminated after completing the execution of the command. The on command does not print any message in this example; restarting the command_error condition prints a message.

Name: or

SYNTAX AS A COMMAND

```
or {tf_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[or {tf_args}]
```

FUNCTION

returns true if any tf_arg is equal to true; otherwise it returns false. If there are no tf_args, it returns the or-identity "false". If any tf_arg has a value other than true or false, an error message is printed.

ARGUMENTS

tf_args
are any active functions that return either "true" or "false".

EXAMPLES

The command line

```
! or [equal ([st -mode ([segs **])]) rew]
```

returns true if there is at least one segment in the working directory to which you have rew access. It returns false if there are no segments in the working directory.

The active function

```
[or [equal (&r1 &r2 &r3) tape1]]
```

inside an exec_com returns true if either the first, second, or third argument to ec is "tape1".

Name: overlay, ov

SYNTAX AS A COMMAND

```
ov paths {-control_args}
```

FUNCTION

reads several ASCII segments and writes the result of superimposing print positions from each segment on the user_output I/O switch output.

ARGUMENTS

paths
are the pathnames of input segments. The archive convention is allowed.

CONTROL ARGUMENTS

-indent N, -ind N
indents the print positions of an input segment N columns on output. It only affects the path immediately preceding it. If you don't give it, an indent of 0 is used.

-page_length N, -pl N
sets the page length of the output. If you don't supply it, a page length of 60 is used.

NOTES

Because the overlay command uses the printer conversion programs, control characters are removed from input files except for newline (NL), backspace (BS), vertical tab (VT), and formfeed (FF).

If identical print positions containing the same characters are superimposed, a boldface type results. By following input segments with -indent, you create output containing columns of text.

Name: page_trace, pgt

SYNTAX AS A COMMAND

```
pgt {N} {-control_args}
```

FUNCTION

prints a recent history of page faults and other system events within the calling process.

ARGUMENTS

N
prints the last N system events (mostly page faults) recorded for the calling process. If you give no N, all the entries in the system trace list for the calling process are printed. Currently, there is room for approximately 300 entries in the system trace array.

CONTROL ARGUMENTS

-from STR, -fm STR
searches the trace array for a user marker matching STR. If one is found, printing begins with it; otherwise, printing begins with the first element in the array.

-long, -lg
prints full pathnames where appropriate. (Default: to print only entrynames)

-no_header, -nhe
suppresses the header that names each column. (Default: to print the header)

-output_switch swname, -osw swname
writes all output on the I/O switch named swname, which must already be attached and open for stream_output. (Default: to write all output on the user_output I/O switch)

-to STR
stops printing if a user marker matching STR is found. If you specify both -from and -to, the "from" marker is assumed to occur before the "to" marker. (Default: to print until the end of the array)

NOTES

You can't give a count value (N) and either -from or -to in the same invocation.

If you give -long, page_trace prints two lines of information for each page fault: the first indicates the page on which the fault occurred; the second, the location of the instruction that caused the fault (i.e., the instruction that referenced the page in the first line). This second line is printed only if the system administrator has enabled the collection of this data.

Since it is possible for segment numbers to be reused within a process and since only segment numbers (not entrynames or pathnames) are kept in the trace array, the entrynames and pathnames associated with a trace entry may be for previous uses of the segment numbers, not the latest ones. In fact, the entry and pathnames printed are the current ones appropriate for the given segment number.

For completeness, events occurring while inside the supervisor are also listed in the trace. The interpretation of these events sometimes requires detailed knowledge of the system structure; in particular they may depend on activities of other users. For many purposes you will find it appropriate to identify the points at which the supervisor was entered and exited and ignore the events in between.

Typically any single invocation of a program does not induce a page fault on every page touched by the program, since some pages may still be in primary memory from previous uses or used by another process. It may be necessary to obtain several traces to identify fully the extent of pages used.

NOTES ON OUTPUT FORMAT

The first column of output describes the type of trace entry; an empty column indicates that the entry is for a page fault. The second column is the real time, in milliseconds, since the previous entry's event occurred. The third (printed for page faults only) is the ring number in which the page fault occurred. The fourth contains the page number for entries, where appropriate. The fifth gives the segment number for entries, where appropriate. The last is the entryname (or pathname) of the segment for entries, where appropriate.

Name: pascal, pas

SYNTAX AS A COMMAND

```
pas path {-control_args}
```

FUNCTION

invokes the Pascal compiler, which compiles a source program written in Pascal and produces a Multics executable object segment.

ARGUMENTS

path
is the pathname of the source segment. The suffix .pascal is assumed.

CONTROL ARGUMENTS

-add_exportable_names, -aen
adds names of exported variables and procedures to the object segment.

-brief_map, -bfm
produces a compilation listing containing source, error messages, and a statement map.

-brief_table, -bftb
generates a partial symbol table consisting of only a statement table that gives the correspondence between source line numbers and object locations for use by symbolic debuggers. The table appears in the symbol section of the object segment. This control argument does not significantly increase the size of the object segment.

-compilation_warnings, -cw
prints compilation warnings for minor errors. (Default)

-conditional_execution VAR_NAME true/false, -cond VAR_NAME true/false
forces the value of the conditional compilation variable VAR_NAME to either true or false. It overrides any assignments of VAR_NAME in the text of the program.

-debug, -db
generates code to check for references outside of array bounds, invalid assignments, values that are out of range, and a variety of other potential errors. Also initializes program storage to blanks (\040) so that a reference through an uninitialized pointer causes a fault_tag_1 condition. (Default)

-english
assumes that Pascal reserved words are in English. (Default)

-error_messages, -em
prints error messages on user_output and includes them in the listing segment. (Default) cbn A

-extended_character_code, -ecc
extends internal code allowed for characters to 255 (decimal).

-french
accepts Pascal reserved words in French. Type "help pascal_french_keywords.gi" for the correspondence between French and English reserved words.

-full_extensions, -full
uses all nonstandard extensions defined for Multics Pascal. (Default)

-interactive, -int
allows text files to operate in interactive mode. On reset or readln, "get" of next character is deferred until the next reference to the file or to one of the variables attached to the file, such as eof, eoln, and file^. (Default)

-io_warnings, -iow
allows warnings to be printed by I/O procedures called by the compiled program. (Default)

-list
produces a compilation listing including source, error messages, map and cross-reference of symbols, statement map, and generated code in symbolic ALM.

-long_profile, -lpf
generates additional code that records the virtual CPU time and number of page faults for each source statement. It is incompatible with -pf. The profile command can handle both regular and long profiles. This feature adds considerable CPU overhead to heavily executed code. The extra CPU time is subtracted out so that it does not appear in the report printed by profile.

-map
produces a compilation listing including source, error messages, map and cross-reference of symbols, and statement map.

-no_compilation_warnings, -ncw
does not print compilation warnings.

-no_debug
does not generate code to test for references outside of array bounds, values out of range, or other errors, and does not initialize storage to blanks.

-no_error_messages, -nem
does not print error messages on user_output. They are still included in the listing segment.

-no_extended_character_code, -necc
allows internal code range of 0..127 for characters, as required by the standard. (Default)

-no_interactive, -nint
does not allow text files to operate in interactive mode.

-no_io_warnings, -niow
does not print I/O warnings if a nonfatal error occurs in I/O procedures called by this program.

-no_list
does not produce a compilation listing. (Default)

-no_long_profile, -nlpf
does not generate additional code to record the virtual CPU time and number of page faults for each source segment. (Default)

-no_private_storage, -nps
dynamically allocates exported variables in external static. (Default)

-no_profile, -npf
does not generate code to meter the execution of source statements. (Default)

-nonrelocatable, -nrlc
generates an object segment that cannot be bound, thus saving from 10 to 20 percent of compilation time.

-no_table, -ntb
does not generate a symbol table in the object segment.

-page_length N, -pl N
specifies a page length for the listing segment. (Default: 59 lines)

-private_storage, -ps
allocates all exported variables in a segment in the process directory named progname.defs, where progname is the entryname of the path argument, without the .pascal suffix. This segment is created if it does not exist.

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

-reference_table -rftb
generates a full symbol table (see -table) and adds for each variable a table of statements where this variable is referenced or modified. This feature, used by pascal_cross_reference, is experimental.

-relocatable, -rlc
generates an object segment that can be bound. (Default)

-sol_extensions, -sol
allows only French SOL extensions to be used (type "help pascal_extensions.gi" for their list).

-standard
allows only standard (ISO) Pascal to be used. (Default: -full)

-table, -tb
generates a full symbol table for use by symbolic debuggers. The symbol table is part of the symbol section of the object segment and consists of two parts: a statement table that gives the correspondence between source line numbers and object locations, and an identifier table containing information about every identifier actually referenced by the source program. This control argument usually lengthens the object segment significantly. (Default)

NOTES

If compilation errors are encountered, error messages are printed on user_output.

If you supply incompatible control arguments, the rightmost one is used.

Multics Pascal is case insensitive. All identifier names are mapped to lowercase in the program and its symbol table. As a result, the Pascal program header

```
program: Foo;
```

produces a segment entry point with the name "foo."

For information on Pascal see the *Multics Pascal User's Guide* (GB62).

NOTES ON LISTING

The Pascal compilation listing contains the following sections in this order:

1. Header: gives the full pathname of the source segment, the Multics site indentification, the date and time of compilation, and the compiler indentification.

2. Source: with lines numbered sequentially. In include files, file number precedes the line number.

3. Any error messages.

4. Storage requirements for the object segment.

5. List of source files used.

6. Complete map and cross-reference for symbols declared and used, symbols declared and never used, and symbols declared by default.

7. Displacement for fields given in octal (bytes), locations for variables given in octal (words), and sizes given in octal (bytes).

8. "DEF:" followed by the number of the line where the symbol is defined. "REF:" followed by the number of the line(s) where the symbol is referenced. An asterisk is printed for each reference where the variable or field is set or passed by reference ("var" parameter) to a subroutine.

9. Complete map and cross-reference of labels. "DEF:" is followed by the number of the line where the label is defined. "DCL:" is followed by the number of the line where the label is declared. "REF:" is followed by the number of the lines where the label is referenced in a GOTO statement. An asterisk is printed where the GOTO statement exits the current procedure.

10. Statement map: gives the octal location of the first instruction of each statement of the source program.

Name: pascal_area_status

SYNTAX AS A COMMAND

```
pascal_area_status {names} {-control_args}
```

FUNCTION

displays and sets attributes of specified Pascal areas.

ARGUMENTS

names
are relative pathnames of Pascal object segments that have their own private areas (see pascal_create_area).

CONTROL ARGUMENTS

-all, -a
operates on all private Pascal areas as well as on the default Pascal area.

-brief, -bf
does not print a dump of each allocated block. (Default)

-default
specifies the default area used by Pascal to allocate storage.

-dump
prints a comprehensive, unformatted dump of the area(s). To be used by the maintainers of the Pascal compiler and related software.

-long, -lg
prints a dump of each allocated block.

-no_dump
does not print a comprehensive dump of the area(s). (Default)

-no_status, -nst
does not print status information.

-no_trace
does not print the address and length of each block. (Default)

-status, -st
prints the maximum size, the size occupied by allocated blocks, and the maximum possible size for a new allocation.

-trace
prints the address and length of each block and, if you give -lg, an octal dump of each block.

NOTES

The Pascal areas are temporary segments. Allocation is performed by the Pascal "new" statement, deallocation by the the "dispose" and "reset" statements.

You can give names and control arguments in any order.

If you specify no areas, -default is assumed. If you specify no actions, -st is assumed. If you specify more than one action, the operations are performed in this order:

```
-st -dump -trace
```

For information on Pascal see the *Multics Pascal User's Guide* (GB62).

Name: pascal_create_area

SYNTAX AS A COMMAND

```
pascal_create_area names {-control_args}
```

FUNCTION

creates temporary, private areas in the process directory for the specified Pascal object segments.

ARGUMENTS

names
are relative pathnames of Pascal object segments that are to have their own private areas. An error occurs for each object segment for which a private area has already been created.

CONTROL ARGUMENTS

-brief, -bf
suppresses the error message that is printed when the private area for a specified program already exists.

-long, -lg
allows the error message that is printed when the private area for a specified program already exists. (Default)

-size N
sets the maximum size of each area to N pages. (Default: 225 records)

NOTES

All Pascal "new" operations executed by the object segments use the associated private areas.

By default, the new operation uses the default Pascal area in the process directory. You can examine this area, and any that are created, using pascal_area_status.

For information on Pascal see the *Multics Pascal User's Guide* (GB62).

Name: pascal_cross_reference, pascal_cref

SYNTAX AS A COMMAND

```
pascal_cref pathnames {-control_args}
```

FUNCTION

examines a set of Pascal object segments that import variables and procedures.

ARGUMENTS

pathnames
are absolute or relative pathnames of Pascal object segments. They must have been compiled with -table.

CONTROL ARGUMENTS

-external_references, -ext_refs
includes line numbers of statements where external (i.e., imported or exported) variables and procedures are referenced or set. The modules must have been compiled with -reference_table to include this information.

-internal_references, -int_refs
includes a list of internal variables or procedures and line numbers of statements where they are referenced or modified. The modules must have been compiled with -reference_table to include this information.

-no_external_references, -no_ext_refs
excludes line numbers of statements where external variables and procedures are referenced. (Default)

-no_internal_references, -no_int_refs
excludes a list of internal variables and procedures. (Default)

-output_file path, -of path
produces a listing named path.x_map. If you use no .x_map suffix as part of the pathname, it is assumed. You must specify -of to get a cross-reference list including include files used and error reporting; otherwise the command writes any error or warning messages on error_output.

NOTES

The modules must have been compiled with -table. The cross-referencer checks declarations of shared variables and procedures, producing an error list of differences between modules. It also notes if the difference may have undesirable results, for example, destruction of data outside the program, such as may occur when sizes of shared objects do not match. Warnings are printed if types do not match but object sizes do.

Unlike the cross_reference command, pascal_cref includes no declarations of variables and procedures in the listing, nor does it compare types. It accepts no modules generated by other translators.

Name: pascal_delete_area

SYNTAX AS A COMMAND

```
pascal_delete_area names {-control_args}
```

FUNCTION

deletes the private areas associated with the specified Pascal object segments. For information on Pascal see the *Multics Pascal User's Guide* (GB62).

ARGUMENTS

names
are relative pathnames of Pascal object segments whose private areas are to be deleted.

CONTROL ARGUMENTS

-brief, -bf
suppresses the message that is printed when a specified program is active on the Multics stack.

-long, -lg
allows the message that is printed when a specified program is active on the stack. (Default)

Name: pascal_display

SYNTAX AS A COMMAND

```
pascal_display {entry_names}
```

FUNCTION

traces the Multics stack and displays on user_output contents of variables declared in all procedures active in the stack.

ARGUMENTS

{entry_names}
are Pascal entry names. If you give entry_names, only the variables of named procedures that are currently active are displayed; if you give no entry_names, variables of all active Pascal procedures are displayed.

NOTES

If you compile programs with -table, the contents of variables are symbolic and are displayed as they would be using the value request under probe. Without symbol tables, octal and ASCII dumps of the variables are provided. Dump location counters have the values of location counters available on the compilation listing.

This command is particularly useful with absentee executions. You can use it in an on condition, as follows:

```
on pascal_error pascal_display program_name
```

EXAMPLES

```
    PROGRAM test_display (input, output) ;

       TYPE
          charac8 = string (8) ;
          ptbox = ^box ;
          box = RECORD
             name : charac8 ;
             value : real ;
             next : ptbox ;
          END ;
       VAR
          first : ptbox ;
          vfl : real ;
str : charac8;

       PROCEDURE build (name : charac8 ; val : real) ;

          VAR
             newbox : ptbox ;
          BEGIN
             new (newbox) ;
             newbox^.name := name ;
             newbox^.value := val ;
             newbox^.next := first ;
             first := newbox ;
          END ;

       BEGIN
          first := NIL ;
          WHILE true DO
             BEGIN
                str := '  ' ;
                write ('name : ') ;
                readln (str) ;
                write ('value : ') ;
                readln (vfl) ;
                build (str, vfl) ;
             END ;
       END.
```

```
   pascal test_display
Pascal 8.03
   on pascal_error pascal_display -long test_display
name : ?Blaise
value : ?134
name : ?Deryl
value : ?123.56
name : ?Amy
value : ?xx
on:  Condition "pascal_error" raised.
pascal_io_$READ_text: Error during READ at line 6 of Pascal file input
pascal_error condition by
 >user_dir_dir>PASCAL>JMAthane>v803>info>test_display|133 (line 36)
  (actually by support procedure pascal_io_$READ_text|17422 (line 2907)
(>user_dir_dir>PASCAL>JMAthane>v803>e>bound_pascal_runtime_|43506))

input chain has a bad real format
pascal_io_$READ_text:  Error during READ at line 6 of Pascal file input

Active procedures in the Multics stack are
234|46600 command_processor_$command_processor_|245
                              (bound_multics_bce_|245) (PL/I)
234|46120 abbrev$abbrev_processor|1307 (bound_command_loop_|10111)
                              (PL/I)
234|43540 on$handler|1505 (bound_command_env_|16331) (PL/I)
234|43200 signal_$signal_|52 (bound_library_1_|7512) (PL/I)
234|40320 pascal_io_$READ_text|6614 (bound_pascal_runtime_|32700)
                              (PL/I) (line 2907)
234|40140 test_display$test_display|46 (PASCAL) (line 36)
   - main -

234|37300 command_processor_$command_processor_|245
                              (bound_multics_bce_|245) (PL/I)
234|36620 abbrev$abbrev_processor|1307 (bound_command_loop_|10111)
                              (PL/I)
234|36160 on$on|220 (bound_command_env_|15044) (PL/I)
234|35320 command_processor_$command_processor_|245
                              (bound_multics_bce_|245) (PL/I)
234|34640 abbrev$abbrev_processor|1307 (bound_command_loop_|10111)
                              (PL/I)
234|34340 listen_$release_stack|72 (bound_command_loop_|23444) (PL/I)
234|33740 get_to_cl_$unclaimed_signal|77 (bound_command_loop_|25025)
                              (PL/I)
234|31200 default_error_handler_$wall|377 (bound_error_handlers_|377)
                              (PL/I)

234|31100 initialize_process_$any_other.2|431 (bound_process_init_|431)
                              (PL/I)
234|30540 signal_$signal_|52 (bound_library_1_|7512) (PL/I)
234|27660 ipc_fast_$ipc_fast_$block|12 (bound_ipc_|54) (PL/I)
```

```
234|26220 tty_io_$tty_io_$get_line|3202 (bound_command_loop_|3202)
                                (PL/I)
234|20060 audit_$audit_get_line|5702 (bound_audit_|5702) (PL/I)
234|16500 tedutil_$tedread_ptr_|2041 (bound_ted_|67331) (PL/I)
234|7400  ted_$ted__|3177 (bound_ted_|6425) (PL/I)
234|5660  ted_command_$ted|543 (bound_ted_|1033) (PL/I)
234|5220  command_processor_$read_list|5116 (bound_multics_bce_|5116)
                                (PL/I)
234|4240  command_processor_$complex_command_processor|1741
                                (bound_multics_bce_|1741) (PL/I)

234|3400  command_processor_$command_processor_|245
                                (bound_multics_bce_|245) (PL/I)
234|2700  abbrev$abbrev_processor|1307 (bound_command_loop_|10111)
                                (PL/I)
234|2400  listen_$listen_|50 (bound_command_loop_|23422) (PL/I)
234|2000  initialize_process_$initialize_process_|241
                                (bound_process_init_|241) (PL/I)

globals for >user_dir_dir>PASCAL>JMAthane>v803>info>test_display are
  str = "Amy"
  vfl = 123.56
  first = 522|116 [pd]> BBBJQWPnghPmKg.temp.0522
  input =
          - Multics io switch :
             syn_ user_input
             stream_input_output
          - Pascal file status :
             text file        input interactive
  input^ = 'x'
  output =
          - Multics io switch :
             syn_ user_output
             stream_input_output
          - Pascal file status :
             text file        output interactive eof
  output^ = ''

                    -------------------------

 item at 522|116 [pd]> BBBJQWPnghPmKg.temp.0522
  (box) =
     name = "Deryl"
     value = 123.56
     next = 522|102 [pd]> BBBJQWPnghPmKg.temp.0522

                    -------------------------
```

```
item at 522|102 [pd]> BBBJQWPnghPmKg.temp.0522
 (box) =
    name = "Blaise"
    value = 134.
    next = null
```

Name: pascal_file_status

SYNTAX AS A COMMAND

```
pascal_file_status
```

FUNCTION

displays information on the status of all standard Pascal files currently in use and all files of active Pascal procedures in the Multics stack. For information on Pascal see the *Multics Pascal User's Guide* (GB62).

Name: pascal_indent

SYNTAX AS A COMMAND

```
pascal_indent old_path {new_path} {-control_args}
```

FUNCTION

indents a Pascal source program according to a standard set of conventions described below. For information on Pascal see the *Multics Pascal User's Guide* (GB62).

ARGUMENTS

old_path
: is the pathname of the source segment to be indented. The .pascal suffix is assumed.

new_path
: is the optional pathname of the indented result. The .pascal suffix is assumed. If you omit new_path, the indented copy replaces the original segment. If errors are detected in the source, however, a temporary indented copy is created instead and its pathname is printed in an error message.

CONTROL ARGUMENTS

-brief, -bf
suppresses warning messages for invalid or non-Pascal characters found outside a string or comment. Errors corresponding to suppressed messages do not prevent the original source segment from being replaced.

-comment N, -com N
indents comments at column number N. Comments are lined up at this column unless they occur at the beginning of a line and are preceded by a blank line. (Default: column 61)

-english
assumes that the source program is written in English. (Default)

-french
assumes that the source program is written in French.

-highlight, -hl
translates reserved symbols of the Pascal language to lowercase if you provide -uc; to uppercase otherwise so that they stand out from the rest of the text.

-indent N, -in N
indents each level an additional N spaces. (Default: 5 spaces)

-lmargin N, -lm N
sets the left margin for top-level program statements after the Nth column. (Default: 10)

-long, -lg
allows warning messages for invalid or non-Pascal characters. (Default)

-lower_case, -lc
translates all uppercase letters outside of strings and comments to lowercase.

no_case_translation, nct
does not translate letters outside strings and comments to uppercase or lowercase. (Default)

-no_highlight, -nhl
does not translate Pascal reserved symbols to lowercase or uppercase. (Default)

-upper_case, -uc
translates all lowercase letters outside of strings and comments to uppercase.

NOTES ON INDENTING STYLE

Multiple spaces are replaced by single spaces, except inside strings and for nonleading spaces and tabs in comments. Trailing spaces and tabs are removed from all lines before indenting. Spaces are inserted before left parentheses, brackets, and braces, and removed after them. Spaces are inserted after right parentheses, brackets, and braces, and removed before them. Spaces are inserted around the constructs =, ^=, <>, <=, >=, :=, ;, and : and operators in expressions.

Parentheses, brackets, and braces must balance. The keywords "begin," "case," and "repeat" must balance with their corresponding "end" statements; likewise for "repeat" and "until" constructs.

Name: pascal_reset_area

SYNTAX AS A COMMAND

```
pascal_reset_area {names} {-control_arg}
```

FUNCTION

frees all blocks in the specified areas. For information on Pascal see the *Multics Pascal User's Guide* (GB62).

ARGUMENTS

names
are relative pathnames of Pascal object segments that have their own private areas. If you supply no names, the default Pascal area is reset. (See pascal_create_area.)

CONTROL ARGUMENTS

-size N
sets the maximum size of each specified area to N records after resetting the area. (Default: 255 records)

Name: pascal_set_prompt

SYNTAX AS A COMMAND

```
pascal_set_prompt {string} {-control_arg}
```

FUNCTION

sets the prompt string used by Pascal programs in interactive mode. Type "help pascal_terminal_io" for a description of the interactive mode.

ARGUMENTS

string
specifies the prompt string.

CONTROL ARGUMENTS

-no_prompt, -npmt
prints nothing for a prompt.

NOTES

If you provide no arguments, the default prompt "?" is restored.

For information on Pascal see the *Multics Pascal User's Guide* (GB62).

Name: path

SYNTAX AS A COMMAND

```
path path {entry {component}}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[path path {entry {component}}]
```

FUNCTION

returns the absolute pathname represented by the argument if you supply one argument; returns the absolute pathname of the archive component (if you give component) of the entry in the directory specified by path if you specify two or three arguments.

ARGUMENTS

path
If you don't give entry, this is the pathname to be expanded and returned; otherwise it is the pathname of the directory to be used in the returned pathname.

entry
is the optional entryname to be used in the returned pathname.

component
is the optional archive component name to be used in the returned pathname.

NOTES

Since the pathname is returned in quotes, the command processor treats it as a single argument regardless of special characters in the name.

EXAMPLES

Assume that the user's working directory is >udd>Proj>Myname.

```
! path start_up.ec
  >udd>Proj>Myname>start_up.ec

! path >udd>Multics>Library>Source>bound_command_demos_.s::program.pl1
  >udd>Multics>Library>Source>bound_command_demos_.s::program.pl1

! path <s>bound_expand_path_.s.archive
  >udd>Proj>s>bound_expand_path_.s.archive

! path [hd] my_exec_coms.archive start_up.ec
  >udd>Proj>Myname>my_exec_coms::start_up.ec
```

Name: pause

SYNTAX AS A COMMAND

```
pause {time}
```

FUNCTION

interfaces the timer_manager_$sleep entry point, which allows the caller to "sleep" for a given number of seconds.

ARGUMENTS

time
is the number of seconds (decimal integer) to sleep. If you specify no time, 10 seconds is used.

Name: peruse_crossref, pcref

SYNTAX AS A COMMAND

```
pcref {cref_path} search_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[pcref {cref_path} search_names]
```

FUNCTION

prints or returns information extracted from the output file generated by the cross_reference command.

ARGUMENTS

cref_path
is the pathname of the crossref output file to search. It can be a multisegment file (MSF). It must contain a > or < character to distinguish it from a search_name. If you supply no cref_path, the total system cross-reference (>ldd>crossref>total.crossref) is used. To specify a cross-reference in your working directory, use -pathname.

search_names
are one or more names to search for references to in the crossref. They can be either symbolic linker references or include file names and can have any of the following forms:

```
segname
segname$entryname
XXX.incl.YYY
```

Any component of a search name can be a star name, except that neither a segname nor an include file name can begin with a star name character and the string ".incl" must appear in toto. If you specify no entryname with the segname, all references to any entry points in the segment are listed. XXX.incl is accepted as an abbreviation for XXX.incl.*. Don't use > and < in a search name.

CONTROL ARGUMENTS

-brief, -bf
does not print any information for selected cross-reference items that have no entries (callers). (Default)

-brief_errors, -bfe
suppreses any error messages due to entrypoints or include files that are not found in the crossref.

-long, -lg
print selected cross-reference items that have no entries.

-long_errors, -lge
prints an error message if one or more entrypoints or include files given on the command line are not found in the crossref. (Default)

-pathname crossref path, -pn crossref path
specifies crossref path as the crossref to search.

NOTES

This command uses a binary search to locate the desired information and thus is quite inexpensive, even when searching the total system crossref. Average cost for a single search of the system crossref is about 45 page faults and 0.5 CPU seconds, or roughly 30 times cheaper and far more convenient than using an editor.

No attempt is made to combine the results of the search names--if you ask for something twice, it gets printed twice.

This command does not perform any significant validation on the input filen and is likely to either take faults or signal the logic_error condition if asked to search something other than a crossref output file.

There is no support for synonyms: a search name must be the primary name of a segment and not a synonym established in a bindfile or the hardcore header.

There is no way to select specific types of things, such as all the unresolvable references in the crossref.

EXAMPLES

```
! pcref phcs_
! pcref hphcs_$*acl
! pcref >ldd>crossref>total.crossref stack_frame.incl
```

OUTPUT EXAMPLE

References to objects matching search names are displayed as follows:

References to phcs_$ring_0_peek: (STAND-ALONE in HARDCORE)

```
as_meter_, copy_salvager_output, display_branch, namef_,
ring_zero_peek_, sweep_pv, vpn_cv_uid_path_
```

If a matching object is not referenced by anything, it is identified as such. If a search name does not match anything found in the crossref, a diagnostic is displayed. The listing is a maximum of 72 characters wide.

Name: picture, pic

SYNTAX AS A COMMAND

```
pic pic_string values {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[pic pic_string values {-control_arg}]
```

FUNCTION

returns one or more values processed through a specified PL/I picture.

ARGUMENTS

pic_string
: is a valid PL/I picture as defined in the PL/I Reference Manual and the PL/I Language Specification.

values
: are strings having data appropriate for editing into the picture. Each value must be convertible to the type implied by the picture specified. If multiple values are presented, the results are separated by single spaces. Any resulting value that contains a space is quoted.

CONTROL ARGUMENTS

-strip
: removes leading spaces from edited picture values; removes trailing zeros following a decimal point; removes a decimal point if it would have been the last character of a returned value.

NOTES

For more information on PL/I picture and picture strings, see the *PL/I Reference Manual* (AM83) or the *PL/I Language Specification* (AG94).

EXAMPLES

```
create file_([pic 999 [index_set 8 14]])
list file_*

Segments = 7, Lengths = 0

r w  0  file_014
r w  0  file_013
r w  0  file_012
r w  0  file_011
r w  0  file_010
r w  0  file_009
r w  0  file_008

string [pic zzzzz9v.9999 000305.000]
305.0000
string [pic zzzzz9v.9999 000305.000 -strip]
305
```

Name: pl1

SYNTAX AS A COMMAND

```
pl1 path {-control_args}
```

FUNCTION

invokes the PL/I compiler to translate a segment containing the text of a PL/I source program into a Multics object segment. You cannot call it recursively.

ARGUMENTS

path
: is the pathname of a PL/I source segment that is to be translated by the PL/I compiler. If path does not have a suffix of pl1, one is assumed; it must be the last component of the source segment's name.

CONTROL ARGUMENTS

-brief, -bf
: produces error messages written onto the user_output I/O switch containing only an error number, a statement identification, and, when appropriate, the identifier or constant in error.

-brief_table, -bftb
generates a partial symbol table consisting of only a statement table that gives the correspondence between source line numbers and object locations for use by symbolic debuggers. The table appears in the symbol section of the object segment. This control argument does not significantly increase the size of the object program.

-check, -ck
does a syntactic and semantic checking of a PL/I program. Only the first three phases of the compiler are executed. Code generation and the manipulation of the working segments used by the code generator are skipped.

-check_ansi
generates a severity 1 error message for each construct the compiler detects that is allowed by Multics PL/I but not by the ANSI standard X3.53-1976.

-list, -ls
produces a source program listing with symbols, followed by an assembly-like listing of the compiled object program. It significantly increases compilation time; avoid it whenever possible by using -map.

-long, -lg
produces error messages written onto the user_output I/O switch containing an error number, statement identification, when appropriate, the identifier or constant in error, an explanatory message of one or more sentences, and, in most cases, the text of the erroneous statement. Once a given error message is printed in the long form, all further instances of it are printed in the brief form. (Default)

-long_profile, -lpf
generates additional code that records the virtual CPU time and number of page faults for each source statement. It is incompatible with -profile. The profile command can handle both regular and long profiles. Use of this feature adds considerable CPU overhead to heavily executed code. The extra CPU time is subtracted out, so that it does not appear in the report generated by profile.

-map
produces a source program listing with symbols, followed by a map of the object code generated by the compilation. It gives sufficient information to allow you to debug most problems online.

-no_check, -nck
generates an object segment. (Default)

-no_check_ansi
does not generate an error message for Multics-dependent PL/I constructs. (Default)

-no_list, -nls
does not produce a listing segment. (Default)

-no_optimize, -not
    does not invoke the extra compiler phase. (Default)

-no_profile, -npf
    does not generate code to meter the execution of source statements. (Default)

-no_separate_static, -nss
    places internal static variables in the linkage section of the object segment. (Default)

-no_table, -ntb
    does not generate a symbol table in the object segment. (Default, if you supply -optimize)

-optimize, -ot
    invokes an extra compiler phase just before code generation to perform certain optimizations, such as the removal of common subexpressions, which reduces the size and execution time of the object segment. Use of this control argument adds 10 to 20 percent to the compilation time.

-prefix STR
    if STR is not null, compiles the program as if it were preceded by the condition prefix "(STR):". STR is a list of one or more PL/I enabled or disabled computational condition names separated by commas and optional horizontal white space. Since the program is compiled as if "(STR):" was inserted before the first line of the source segment, -prefix does not override condition prefixes given in the source segment. If STR is null, no additional condition prefix is used. The enabled computational condition names are: conversion (conv), fixedoverflow (fofl), overflow (ofl), size, stringrange (strg), stringsize (strz), subscriptrange (subrg), underflow (ufl), and zerodivide (zdiv). The disabled computational condition names are: noconversion (noconv), nofixedoverflow (nofofl), nooverflow (noofl), nosize, nostringrange (nostrg), nostringsize (nostrz), nosubscriptrange (nosubrg), nounderflow (noufl), and nozerodivide (nozdiv). STR cannot contain an enabled condition name and a disabled condition name that identify the same condition.

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

-separate_static, -ss
    generates separate sections in the object segment created for the linkage information and the internal static variables. The default is to place internal static variables in the linkage section since both types of data are per process and writable. This control argument is useful primarily for programs that are prelinked and can therefore share the linkage section with other users.

-severityN, -svN
writes error messages whose severity is less than N (where N is 1, 2, 3, or 4) into the user_output switch although all errors are written into the listing. If you don't select this control argument, a severity level of 1 is assumed (see "Notes on Error Diagnostics" below).

-single_symbol_list, -ssl
produces a source program listing with symbols. The symbols are listed in one, single, alphabetized list. If you don't give -single_symbol_list but specify -list, -map, or -symbols, the symbols are separated into four lists, arranged by declaration type.

-source, -sc
produces a source program listing.

-symbols, -sb
produces a source program listing with symbols.

-table, -tb
generates a full symbol table for use by symbolic debuggers. The symbol table is part of the symbol section of the object program and consists of two parts: a statement table that gives the correspondence between source line numbers and object locations and an identifier table containing information about every identifier actually referenced by the source program. This control argument usually lengthens the object segment significantly. (Default, unless you supplied -optimize)

LIST OF ADDITIONAL CONTROL ARGUMENTS

The following control arguments, while available, are probably not of interest to you.

-debug, -db
leaves the list-structured internal representation of the source programs intact after a compilation. This control argument is used for debugging the compiler. You can use the command pl1$clean_up to discard the list structure.

-no_debug, -ndb
deletes the internal representation after compiling a program. (Default)

-no_time, -ntm
does not print a table of the time used by each phase of the compiler after the compilation. (Default)

-time, -tm
prints a table after compilation, a table giving the time (in seconds), the number of page faults, and the amount of free storage used by each of the phases of the compiler. This information is also available from the command pl1$times invoked immediately after a compilation.

NOTES

If you invoke this command without control arguments, it generates an object segment.

If you give both a control argument and an incompatible alternative on the same command line, the rightmost one is used. For example:

```
pl1 prog -brief_table -map -no_list -table
```

is equivalent to

```
pl1 prog -table
```

A successful compilation produces an object segment and leaves it in your working directory. If an entry with that name already exists in the directory, its access control list (ACL) is saved and given to the new copy; otherwise, you are given re access to the segment with ring brackets v,v,v, where v is your process's validation level.

If you specify -map, -list, -source, -symbols, or -single_symbol_list, the command creates a listing segment in your working directory and gives it a name consisting of the entryname portion of the source segment with a suffix of list rather than pl1 (e.g., a source segment named valid.pl1 has a listing segment named valid.list). The ACL is as described for the object segment except that you are given rw access to the newly created segment. Previous copies of the object segment and the listing segment are replaced by the new segments created by the compilation.

See the *Multics PL/I Language Specification Manual* (AG94) and the *Multics PL/I Reference Manual* (AM83).

NOTES ON SEARCH LIST

The PL/I compiler uses the translator search list, which has the synonym trans. For more information on search lists, see the search facility commands--add_search_paths, in particular.

NOTES ON ERROR DIAGNOSTICS

The PL/I compiler can diagnose and issue messages for about 350 errors. These messages are graded in severity as follows:

1 Warning only. Compilation continues without ill effect.

2 Correctable error. The compiler remedies the situation and continues, probably without ill effect. For example, a missing end statement can be corrected by appending the string ";end;" to the source. This action does not, however, guarantee the correct results.

3 An uncorrectable but recoverable error. That is, the program is definitely in error and cannot be corrected, but the compiler can and does continue executing up to just before code is generated. Thus, any further errors are diagnosed. If the error is detected during code generation, code generation is completed although the code generated is not correct. After the compilation, a message is printed to the error_output I/O switch to inform you that a severity 3 error has occurred.

4 An unrecoverable error. The compiler cannot continue beyond this error. The message is printed and then control is returned to the pl1 command unwinding the compiler. The command writes an abort message into the error_output I/O switch and returns to its caller.

Error messages are written into the user_output I/O switch as they occur; thus, you can quit the compilation immediately when an error message is printed. You can specify -brief so that the messages are shorter. Here is an example of a long error message:

```
ERROR 158, SEVERITY 2 ON LINE 30
A constant immediately follows the identifier "zilch".
SOURCE:  a = zilch 4;
```

If you choose -brief, you see instead:

```
ERROR 158, SEVERITY 2 ON LINE 30
"zilch"
```

Once a given error message is printed on your terminal in the long form, all further instances of it are printed in the short form.

If a listing is being produced, the error messages are also written into the listing segment. They appear, sorted by line number, after the listing of the source program. No more than 100 messages are printed.

NOTES ON SEVERITY VALUES

This command associates the following severity values to be used by the severity active function:

Value	*Meaning*
0	No compilation yet or no error
1	Warning
2	Correctable error
3	Fatal error
4	Unrecoverable error
5	Could not find source.

NOTES ON LISTING

The listing created by the pl1 command begins with a line-numbered image of the source segment. If you specify -symbols, -single_symbol_list, -map, or -list, this is followed by a table of all the names declared within the program. The names are categorized by declaration type as follows:

1. declared by declare statement
2. declared by declare statement and never referenced
3. declared by explicit context (labels and entries)
4. declared by implicit context or implication.

Within these categories, the symbols are sorted alphabetically and then listed with their location; storage class; data type; size or precision; level; attributes such as initial, array, internal, external, aligned, and unaligned; and a cross-reference list. If you give -single_symbol_list, these four categories are combined into one alphabetized list. Next is a table of the program's storage requirements and the reasons why a block is nonquick. Next is a listing of any internal static variables, sorted by offset, and a listing of any automatic variables, sorted by block and offset. Next is a listing of external operators used, external entries called, and external variables referenced by the program. The symbol listing is followed by any error messages.

If you select -map, the object code map follows the list of error messages. This table gives the starting location in the text segment of the instructions generated for statements starting on a given line. The table is sorted by ascending storage locations.

Finally, the listing contains the assembly-like listing of the object segment produced (if you specify -list). The executable instructions are grouped under an identifying header that contains the source statement that produced the instruction. Operation code, base register, and modifier mnemonics are printed beside the octal instruction. If the address field of the instruction uses the IC (self-relative) modifier, the absolute text location corresponding to the relative address is printed on the remarks field of the line. If the reference is to a constant, the octal value of the constant's first word is also printed. If the address field of the instruction references a symbol declared by you, its name appears in the remarks field of the line.

EXAMPLES

The following command line compiles the segment prog.pl1 with options that facilitate debugging:

```
pl1 prog -table -prefix size,strz,strg,subrg
```

The default action for the stringsize condition returns to the point where the condition is signaled without printing a message. All computational conditions except size, stringsize, stringrange, and subscriptrange are enabled by default.

Name: pl1_abs, pa

SYNTAX AS A COMMAND

```
pa paths {-pl1_args} {-dp_args} {-control_args}
```

FUNCTION

submits an absentee request to perform PL/I compilations.

ARGUMENTS

paths
are the pathnames of segments to be compiled.

pl1_args
are one or more control arguments accepted by the pl1 command.

dp_args
are one or more control arguments (except -delete) accepted by the dprint command.

CONTROL ARGUMENTS

-queue N, -q N
is the priority queue of the request. (See "Notes.") (Default: defined by your site)

-hold, -hd
specifies that pl1_abs should not dprint or delete the listing segment.

-limit N, -li N
specifies a time limit in seconds for the absentee job. (Default: defined by your site)

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

NOTES

The absentee process for which pl1_abs submits a request compiles the segments named and dprints and deletes the listing segments. If you don't supply -output_file, an output segment (path.absout) is created in your working directory; if you specify more than one path, only the first is used. If none of the segments to be compiled can be found, no absentee request is submitted.

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

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

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

If you give no -queue, the request is submitted into the default absentee priority queue defined by your site and, if requested, the output files are dprinted in the default queue of the request type specified on the command line. If you don't specify request type, the "printer" request type is used.

If you supply no -queue, the output files is dprinted in the same number queue as the absentee request. If the request type specified for dprinting does not have that queue, the highest numbered queue available for the request type is used and a warning is issued.

Name: pl1_macro, pmac

SYNTAX AS A COMMAND

```
pmac in_path {out_path} {-control_args}
```

FUNCTION

invokes the stand-alone pl1 macro processor to translate a segment in accordance with the defined pl1 macro language.

ARGUMENTS

in_path
: is the pathname of the source segment. The source segment name must have at least three components, a suffix of "pmac", and a penultimate component of "pl1" or "cds" or "rd". If you don't supply the suffix, it is assumed. The star convention is not supported.

out_path
: is the pathname of the macro processed output segment. If you haven't used -print or -process_dir, then the name of the inpath less the suffix is assumed if not given. The outpath cannot be the same segment as the inpath. The equal convention is not supported. This argument is incompatible with -print and -process_dir.

CONTROL ARGUMENTS

-arguments STRs, -ag STRs
passes the strings as command line arguments. It must be the last control argument, and at least one STR must follow. These arguments are used in conjuntion with the %isarg macro construct (see "Command Line Argument Testing" below).

-call STR
calls STR as a command after the translation is complete if the macro processor does not discover an error.

-no_version, -nver
does not print the version of pl1_macro.

-parameter IDENTIFIER VALUE, -pm IDENTIFIER VALUE
sets the value of macro replacement identifiers on the command line. IDENTIFIER must be a pl1 identifier; VALUE, a decimal integer, a bit string constant, a character constant, or an identifier. (See "Command Line Constants and Their Defaults.")

-print, -pr
prints the macro processed output on user_output rather than place it in a segment.

-process_dir, -pd
places the macro processed output in the process directory rather than in your working directory.

-target STR, -tgt STR
makes the macro processor interpret STR as a target machine and sets the %target builtin (see "Code for Different Target Machines").

-version, -ver
prints the version of pl1_macro. (Default)

BASIC REPLACEMENT CONSTRUCT

To facilitate the use of named constants in places where internal static options (constant) don't work (e.g., label arrays and functions of named constant), there are three ways of defining replacement identifiers--pl1_identifiers that are transformed at lex level to their defined values--by the %replace statement, by the %set statement, and, on the command line, by the %default statement.

Macro Time Constants

```
%replace <identifier> by <constant-expression>
```

where <constant-expression> is an expression whose operands are either constants or identifiers previously defined in other statements or on the command line. The valid operators are the arithmetic ones (+, -, *, and /) the concatenation operator (||), the logical operators (^, &, and |), and the relational operators (=, ^=, <, <=, >, >=, ^>, and ^<). Arithmetic values are represented internally as fixed binary (71). Parentheses can be in all expressions. The usual semantics of expression evaluation apply, and pl1-like conversions are not done implicitly.

Semantic Rules

1. All replacement identifiers must be lexically declared by a replace statement prior to use.

2. A replacement identifier may not appear lexically prior to its declaration in a replace statement. This is to insure that its meaning remains constant throught the compilation unit.

3. Once declared, a replacement identifier may not be redefined to have a different value by a replace statement, nor is there any way to "undefine" a replacement identifier; however, to facilitate replacement identifiers being used in a variety of include files, redeclaration to the same value is permitted.

4. After its declaration, the replacement identifier is replaced where it appears as a token in the lexed source by the value defined in the replace statement.

5. Replacement identifiers have four data types: arithmetic, bit, character, and identifier. For all data types, operands must agree with their operators. The identifier data type is associated with no operator except the = and ^= comparison.

Macro Time Variables

```
%set <identifier> to <constant-expression>
```

The %set statement is like the %replace statement in the way it deals with replacement activity and constant expression evaluation, conflicts with parameters, etc. The only difference is that the placement identifier declared in an %set statement may appear in another %set statement with a different value. Its replacement rule is that it uses the value set in the last %set statement (in the lexical sense). The use of variables in any two of %default, %replace, and %set statements is not allowed.

Command Line Constants and Their Defaults

The -parameter control argument allows the virtual equivalent of a replacement identifier declaration on the command line.

The macro processor uses the replacement identifiers as though they had been declared in %replace statements, with two important differences: (1) these parameters must not be declared in the source in a %replace statement, even to the same value; (2) they must be declared in a %default statement. If the same identifier appears in more than one instance of a "-pm IDENTIFIER VALUE" triplet, the last such triplet takes effect.

```
%default <identifier> to <constant-expression>;
```

If the identifier has been used in a parameter statement, this statement is ignored except to check that the data type of the constant given in the command line and the data type of the expression in the statement agree; otherwise, this statement causes the same substitution behavior as a %replace statement.

Macro Variable Declaration Builtin

```
%isdef (<identifier>)
```

is a macro builtin function that returns a value of data type bit (1). Its value is true only if the argument is a macro replacement identifier that you have lexically declared either in a %default, %set, or %replace construct prior to using %isdef or as a command line parameter.

Command Line Argument Testing

```
%isarg (<pl1-token>)
%isarg (<char_string>)
```

is a macro builtin function that returns a value of data type bit (1). Its value is true only if the argument is one of the character strings following -arguments on the command line.

The character string form of the argument is necessary if a command line argument is a string according to the command processor, but is not a pl1 token (e.g., 34xy). If the argument to %isarg is a character string, it is dequoted and the dequoted value is used in the test; for example, if the command line has -ag 34xy, the test in the source must be phrased as %isarg ("34xy"), rather than %isarg (34xy) because the macro processor works on pl1 tokens. So use only identifiers as command line arguments, to facilitate more reasonable-looking code.

Conditional Compilation

SYNTAX

```
%if <constant-expression> %then <token-string>
[%elseif <constant-expression> %then <token-string>]...
[%else <token-string>] %endif
```

where <token-string> is a possibly null string of tokens and the <constant-expression>'s must evaluate to a bit_string constant.

Semantic Rules

1. The usual semantics of if-then-elseif-then-else statements apply. If the boolean expression in the test clause equals ""b, then the condition is false, otherwise it is true. The %elseif and %else terms are optional, but the %then and %endif keywords are required.

2. The conditional compilation construct is invalid if all the constant expressions do not evaluate to proper logical values.

3. There is no restriction on what may appear as the object token-string of a then or else clause. In particular it may be standard pl1 tokens or further macro constructs such as %replace, %include, etc.

4. In order to facilitate the maintainability of code, use the conditional compilation facility to construct token strings that comprise entire pl1 statments, rather than code fragments.

Code for Different Target Machines

There is a strategy for informing translators which machine they should generate code for. The macro processor also uses this same strategy for use in conditional compilation.

```
%target (<identifier>)
```

is a replacement identifier of data type bit (1) whose value is true only if the value of the <identifier> is equal to the value of the identifier given as the argument of -target on the command line. If you use %target without -target, a default value is supplied and an error of severity 2 indicated. If you don't use %target, then you need supply no information on the command line about the target machine.

There are currently two flavors of target machines. The names l68, 6180, and dps8 are cannonically equivalent and refer to the standard Multics cpu's.

Expansion Time Include Files

```
%INCLUDE <identifier>;
%INCLUDE <quoted-string>;
```

provides an expansion time include file feature. Include files are found through the translator search rules and have the same naming conventions as compile time include files. You are permitted a maximum of 255 include files in one expansion, and you can nest them 64 deep. This differs from %include in that the macro processor merely checks to see that the %include statement is syntactically correct and outputs the statement.

User-Generated Messages

```
%print <char_string>;
%warn <char_string>;
%error <char_string>;
%abort <char_string>;
```

The macro processor sets a severity, an external fixed binary (35) variable, called pl1_macro_severity_. These four constructs allow you to send messages to user_output at macro time and set the minimum value of pl1_macro_severity_ to zero, one, three, and four respectively. The %abort construct immediately aborts the macro_processor. The char_string can be generated as a result of macro time activity.

Skip and Page Macros

These are features that the pl1 compiler accepts. The macro processor checks them for syntactic correctness and passes the statement through.

EXAMPLES

```
!  pmac pc.pl1.macro -call "pl1 pc -ot -map" -target L68
     -pm VERSION 1 -ag PTE
```

The example

```
%set FOO to 1;
first = FOO;
%set FOO to 2;
second = FOO;
%set FOO to (FOO + 3)**2;
third = FOO;
```

assigns the values 1, 2, and 25 to first, second, and third, respectively.

With no -parameter triplet to declare FRED on the command line, the statement

```
%default FRED to 2345;
```

replaces all future references of FRED with 2345. If, however, the command line contains the triplet

```
-pm FRED 123
```

all references to FRED are replaced by 123. If the command line contains the triplet

```
-pm FRED foo
```

an error of incompatible data types occurs.

Examples of the $isdef macro builtin function are:

```
%if ^ %isdef (fred) %then
%error "The replacement identifier ""fred"" has not been defined.";

%replace paradise by %isdef (adam) & %isdef (eve) & ^ %isdef (snake);
```

An example of good style in conditional compilation can be:

```
%if %target (168)
%then call x (2);
%else call y (22); z = z + foo; %endif
```

An example of bad style can be:

```
call %if %target (6180)
%then x (2);
%else  y (22); z = z + foo; %endif
```

Examples of the %target replacement identifier can be:

```
%if %target (dps8)
%then %include sdw.t3;
%else %include sdw.t1; %endif

%if %target (168)
%then call foo$bar (carrot_juice, fruit_salad, code); %endif
```

Here are some examples of user-generated messages:

```
%default NAME to "6180";
%print, "NAME has been set to " || NAME;

%if ^ $isdef (FRED) %then
%error, "You forgot to define ""FRED"".; %endif

%if %isarg (PTE) & %isarg (NOPTE) %then
%abort, "PTE and NOPTE cannot both be arguments"; %endif.
```

Name: plus

SYNTAX AS A COMMAND

```
plus {num_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[plus {num_args}]
```

FUNCTION

returns the sum of num_args. If you provide no num_args, 0 (the additive identity) is returned.

EXAMPLES

```
! plus 3.5 3
  6.5

! plus -5
  -5
```

Name: print, pr

SYNTAX AS A COMMAND

```
pr paths {-control_args}
```

FUNCTION

prints ASCII segments and multisegment 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
: treats each archive component as a new file for heading and line numbering. If any lines are printed from an archive component and if you supplied -header, prints 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 you give -no_archive.

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

-exclude STRING, -ex STRING
does not 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/
does not print lines containing a string matching the regular expression REGEXP. (See the qedx command for the definition of regular expressions.)

-for N
prints N lines from the file, including the first line. If you also use -to, printing stops when the first control argument is satisfied. (Default: to print the whole file)

-from X, -fm X
begins printing from the Xth line. This control argument and -last are incompatible. (Default: line 1)

-from /REGEXP/, -fm /REGEXP/
begins with first line matching the regular expression REGEXP.

-from_page P
starts printing with the Pth page, counting the first page as 1. (Default: to start with page 1)

-header, -he
prints a header of the form

```
NAME     date time
```

before each segment. If you choose -archive, the header is printed before each archive component instead of before each segment. This control argument is the default if you give no other control argument or if you use the star convention or multiple pathnames.

-indent N, -ind N
indents the printed output N columns. (Default: no indentation)

-last N, -lt N
print the last N lines from the file, or the last N lines of the region selected by -to.

-left_col N, -lc N
does not print columns 1 to N-1. It 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. (Default: to print starting with column 1)

-line_length N, -ll N
formats 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 although you can give larger values.

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

-match /REGEXP/
prints only lines containing a string matching the regular expression REGEXP.

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

-no_archive, -nac
does not print headings for individual archive components (even if the file being printed is an archive) and treats the file being printed as a single segment for line numbering and heading.

-no_chase
does not include links when processing starnames. (Default)

-no_header, -nhe
suppresses the header before segments or archive components. This is the default if you give only one pathname and other control arguments.

-no_vertsp
simulates formfeed and vertical-tab characters by outputting newline characters.

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

-output_switch SWITCH_NAME, -osw SWITCH_NAME
directs the output to an attached and open (for stream output or stream input/output) I/O switch. If not supplied, the output is directed to the user_output switch.

-page_length N, -pl N
starts a new page by inserting a formfeed character after every N lines of the file are printed (see "Notes"). (Default: no pagination)

-phys_page_length N, -ppl N
determines how many newline characters should be inserted between pages when you specify -no_vertsp. N, whose default value is 66, is the number of lines on a whole page of paper. (See "Notes.")

-right_col N, -rc N
does not print columns past N. Lines extending past column N are truncated on

the right. (Default: to print all columns)

-stop, -sp
pauses before the first page and after each successive page until you type a newline.

-to N
stops printing with line number N. (Default: to print all lines)

-to /REGEXP/
stops printing with the first line matching the regular expression REGEXP. The search for REGEXP begins after the first line printed.

-to_page N
stops printing after the Nth page.

-vertsp
sends formfeed and vertical-tab characters to the terminal. (Default)

-wait, -wt
pauses before the first page until you type a newline.

NOTES

The -page_length control argument works with -phys_page_length 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 you position the printer paper so that text begins printing on the 13th line, then there are even amounts of leading and trailing space on each page.

If you select any of -line_length, -page_length, -phys_page_length, or -right_col 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 print. 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. You can use -name to indicate that a number is intended as a pathname.

You can supply more than one -match and more than one -exclude; a line is printed if any -match selects it unless one -exclude prevents it from being printed.

EXAMPLES

The command line

```
! print xyz
```

prints the segment or multisegment file "xyz" with a header.

The command line

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

scans all archive segments in your working directory for lines matching the regular expression /bit *(fixed/. Those lines 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.

Name: print_attach_table, pat

SYNTAX AS A COMMAND

```
pat {switch_names} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[pat {switch_names} {-control_args}]
```

FUNCTION

prints information on your terminal about the I/O switch name associations created by attach calls in your current ring. As an active function, returns the switch names selected by switch_names and control arguments.

ARGUMENTS

switch_names
: are the names of I/O switches. The star convention is allowed. You can use -name to inhibit star name processing or to supply a switch name that appears to be a control argument. Information about only the specified switches is printed. If a given switch is not currently attached, a message is printed on your terminal.

CONTROL ARGUMENTS

-all, -a
prints the state of all the selected switches that match the star names, whether they are attached or not. Specify only one of -all, -attached, or -open.

-attached, -att
prints the state of those switches that match the star names only if they are currently attached. (Default)

-brief, -bf
does not print information for the four standard switch_names (user_i/o, user_input, user_output, and error_output) even if they match a star name.

-name switch_name, -nm switch_name
interprets the switch_name literally, even if it looks like a star name or a control argument.

-open
prints the state of those switches that match the starnames only if they are attached and open.

NOTES

If you invoke it as a command, the attach and open descriptions associated with the indicated switch names are printed.

If you supply no arguments, the information for all switches currently attached is printed.

See the io_call command.

Name: print_auth_names, pan

SYNTAX AS A COMMAND

```
pan {-control_args}
```

FUNCTION

prints the names of the sensitivity levels and access categories defined for the installation.

CONTROL ARGUMENTS

-all, -a
lists all possible names (above system high).

-brief, -bf
suppresses the title and headings.

-category, -cat
lists only the access categories.

-level
lists only the sensitivity levels.

NOTES

Only the names that can be used to describe an access class or access authorization between system low and system high are printed, unless you give -all.

This command lists the names that are acceptable to convert_authorization_ (see the Subroutines manual) to define an access class or access authorization. (All commands and system interfaces that use a character string to describe an access class use this subroutine.) Both the long and short names are printed.

Name: print_bind_map, pbm

SYNTAX AS A COMMAND

```
pbm path {components} {-control_args}
```

FUNCTION

displays all or part of the bind map of an object segment generated by version 4 or subsequent versions of the binder.

ARGUMENTS

path
is the pathname of a bound object segment.

components
are the optional names of one or more components of this bound object and/or the bindfile name. Only the lines corresponding to these components are displayed. A component name must contain one or more nonnumeric characters. If it is purely numerical, it is assumed to be an octal offset within the bound segment, and the lines corresponding to the component residing at that offset are displayed. A numerical component name can be specified by preceding it with -name. If no component names are supplied, the entire bind map is displayed.

CONTROL ARGUMENTS

-long, -lg
prints the components' relocation values (also printed in the default brief mode), compilation times, and source languages.

-name STR, -nm STR
is used to indicate that STR is really a component name, even though it appears to be an octal offset.

-no_header, -nhe
omits all headers, printing only lines concerning the components themselves.

-page_offset, -pgofs
prints as an octal number the page number of the first word of the text section of each component, which is the format used by the cumulative_page_trace command. If the component crosses at least one page boundary, a plus (+) character follows the page number.

Name: print_configuration_deck, pcd

SYNTAX AS A COMMAND

```
pcd {card_names} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[pcd {card_names} {-control_args}]
```

FUNCTION

displays the contents of the Multics configuration deck. The data is kept up-to-date by the reconfiguration commands and, hence, reflects the current configuration being used. The active function returns the selected cards in quotes, separated by a single space.

ARGUMENTS

card_names
are the names of the particular configuration cards to be displayed. You can give up to 32 card names (see the *Multics System Maintenance Procedures Manual*, AM81, for the names of the configuration cards).

CONTROL ARGUMENTS

-exclude FIELD_SPECIFIERS, -ex FIELD_SPECIFIERS
excludes particular cards or card types from being displayed. You can supply one to 14 field specifiers with each -exclude and up to 16 -exclude control arguments. To be eligible for exclusion a card must contain fields that match all field specifiers selected with any -exclude.

-label, -lbl
displays cards with mnemonic labels for each field.

* -match FIELD_SPECIFIERS
selects particular cards or card types to be displayed. You can give one to 14 field specifiers with each -match and up to 16 -match control arguments. To be eligible for selection a card must contain fields that match all field specifiers supplied with any -match.

-no_label, -nlbl
does not display field labels. (Default)

-pathname PATH, -pn PATH
displays the configuration deck in the segment specified by PATH, rather than the configuration deck of the live system.

NOTES

Field specifiers can consist of a complete card field or a partial field and an asterisk. An asterisk matches any part of any field; for example, the field specifier "dsk*" matches any card containing a field beginning with the characters "dsk". You can give specifiers for numeric fields in octal or decimal, but if decimal they must contain a decimal point. You can't use asterisks in numeric field specifiers. All numeric field specifiers are converted to decimal and matched against numeric card fields, which are also converted to decimal. Hence, the field specifier "1024." matches a card containing the octal field 2000, and the field specifier "1000" matches a card containing the decimal field 512.

Selection is performed as follows. If you give no card names, all cards are selected; if you supply any card names, only the cards matching those names are selected; and if more than one card exists with a specified name, all such cards are displayed. If
* you request a nonexistent card, an error message is displayed. If you give any -match arguments, those eligible cards are matched against all field specifiers of each -match argument group; however, at least one -match group must have all its field specifiers match some field on the card to make that card eligible. A similar algorithm is used for any -exclude argument groups. So, if a card is eligible and you supply -exclude arguments, then at least one -exclude group must have all its field specifiers match some field on the card to make that card ineligible. If no match for a given card name or -match group is found in the configuration deck, nothing is displayed for that name or group and no error is displayed. If you give no arguments, the complete configuration deck is displayed.

Specify all card names before the first -match or -exclude argument. Field specifiers following a -match or -exclude argument include all arguments until the next -match or -exclude argument.

No action is taken for misspelled arguments or valid arguments for which there are no corresponding configuration cards.

EXAMPLES

```
!  pcd cpu
   cpu  a  7  168  80.  on
   cpu  b  6  168  80.  on
   cpu  c  5  168  80.  off
```

(For the configuration deck displayed above.)

```
!  pcd cpu -match on
   cpu  a  7  168  80.  on
   cpu  b  6  168  80.  on

!  pcd -match 16 -ex off -ex b
   cpu  a  7  168  80.  on
```

Name: print_default_wdir, pdwd

SYNTAX AS A COMMAND

pdwd

FUNCTION

prints out the pathname of the current default working directory on the user's terminal.

NOTES

See also change_wdir, change_default_wdir, and default_wdir.

Name: print_error_message, pem

SYNTAX AS A COMMAND

```
pem code
```

SYNTAX AS AN ACTIVE FUNCTION

```
[pem code]
```

FUNCTION

prints out the standard Multics (error_table_) interpretation of a specified error code. The various entries given below allow you to specify the error code in either decimal or octal and have the output come out in either the short or long error_table_ form. The active function returns, as a single quoted string, what the command prints.

ARGUMENTS

code
: is the decimal integer to be interpreted. The short form of the error message is printed.

Entry: pel

This entry is the same as pem except that the long form of the error message is printed or returned as a single quoted string.

SYNTAX AS A COMMAND

```
pel code
```

SYNTAX AS AN ACTIVE FUNCTION

```
[pel code]
```

Entry: peo

This entry is the same as pem except that the input code is assumed to be octal.

SYNTAX AS A COMMAND

```
peo octal_code
```

SYNTAX AS AN ACTIVE FUNCTION

```
[peo octal_code]
```

Entry: peol

This entry is the same as pel except that the input code is assumed to be octal.

SYNTAX AS A COMMAND

```
peol octal_code
```

SYNTAX AS AN ACTIVE FUNCTION

```
[peol octal_code]
```

Name: print_link_info, pli

SYNTAX AS A COMMAND

```
pli paths {-control_args}
```

FUNCTION

prints selected items of information for the specified object segments.

ARGUMENTS

paths
: are the pathnames of object files. You can use the archive component pathname convention (::).

CONTROL ARGUMENTS

-entry, -et
: prints a listing of pathi external definitions, giving their symbolic names and their relative addresses within the segment. If pathi is an object multisegment file (MSF), the external definitions in each of the executable components are listed.

-header, -he
: prints the header, which is not printed by default if you select -entry, -length, or -link.

-length, -ln
: prints the lengths of the sections in pathi. If pathi is an object MSF, the lengths of the sections for each executable component are printed.

-link, -lk
prints an alphabetically sorted listing of all external symbols referenced by pathi. If pathi is an object MSF, the list of external links in each executable component is printed. If you use a link in more than one component, the link is listed more than once.

-long
prints additional information when the header is printed. This information includes a listing of source programs used to generate the object segment, the contents of the "comment" field of the symbol header (often containing compiler options), and any unusual values in the symbol header.

-no_header
suppresses printing of the header.

NOTES

If you select no control arguments, -et, -he, -ln, and -lk are assumed. If a path given is an object MSF, the information for each of the executable components is printed.

EXAMPLES

```
! pli program -long -length
                    copy    05/27/83  1126.7 edt Fri

Object Segment >user_dir_dir>MPM>user_documents>commands>copy
Created on 12/23/82  1205.4 edt Thu
by Moore.SysLib.m
using Multics PL/I Compiler, Release 27d, of September 28, 1982

Translator:          PL/I
Comment:             optimize map single_symbol_list
Source:
   12/23/82  1205.5 edt Thu    >ldd>oi>2873dir>copy.pl1
   11/16/82  1554.7 edt Tue    >ldd>include>status_structures.incl.pl1
   05/25/82  1715.1 edt Tue    >ldd>include>star_structures.incl.pl1
Attributes:          relocatable, procedure, standard

        Object    Text    Defs    Link    Symb  Static
Start        0       0    5510    6306    6460    6316
Length    7036    5510     576     152     342       0


<ready>
```

Also printed is

```
Severity, if it is nonzero.
Entrybound, if it is nonzero.
Text Boundary, if it is not 2.
Static Boundary, if it is not 2.
```

This page intentionally left blank.

Name: print_linkage_usage, plu

SYNTAX AS A COMMAND

```
plu
```

FUNCTION

lists the locations and size of linkage and static sections allocated for the current ring. This information is useful for debugging purposes or for analysis of how a process uses its linkage segments.

NOTES

A linkage section is associated with every procedure segment and every data segment that has definitions.

For standard-procedure segments, the information printed includes the name of the segment, its segment number, the offset of its linkage section, and the size (in words) of both its linkage section and its internal static storage.

Name: print_mail, prm

SYNTAX AS A COMMAND

```
prm {mbx_specification} {-control_args}
```

FUNCTION

prints the messages in a mailbox, querying you whether to delete each one after it is printed.

ARGUMENTS

mbx_specification
: specifies the mailbox from which messages are to be printed. If not given, the user's default mailbox (>udd>Project_id>Person_id>Person_id.mbx) is used.

CONTROL ARGUMENTS

-accessible, -acc
: selects only those messages in the mailbox that you are permitted to read. If you have read (r) extended access on the mailbox, print_mail selects all messages in the mailbox; if you have own (o) extended access on the mailbox, it selects only those messages that you sent to the mailbox. (Default)

-acknowledge, -ack
acknowledges messages that request acknowledgement. (Default)

-all, -a
selects all messages in the mailbox regardless of who sent them. It requires read (r) extended access on the mailbox.

-brief, -bf
shortens the greeting message. This message indicates the number of messages in the mailbox.

-brief_header, -bfhe
displays the minimal amount of information from the message header. The date and authors are always displayed; the subject is displayed if it is not blank; the number of recipients is displayed either if there is more than one recipient or if you are not the sole recipient of the message; if the message is forwarded with comments, they are also displayed.

-count, -ct
displays the number of messages read from the mailbox before printing the first message. (Default)

-debug, -db
enables print_mail's debugging facilities. It is not recommended for normal users of print_mail.

-header, -he
displays all information from the message header, including user-defined fields and excluding the message trace and redundant information. (Default)

-interactive_messages, -im
includes interactive messages (as sent by the send_message command) along with regular mail. (Default)

-list, -ls
prints a summary of all the messages before printing the first message. This summary is identical to that produced by the read_mail list request.

-long, -lg
prints the long form of the greeting message. (Default)

-long_header, -lghe
displays all information from the message header, including network-tracing information, even if some of it is redundant (e.g., if the From, Sender, and Delivery-By fields are all equal, -long_header forces print_mail to display them when it prints the message).

-mail, -ml
prints ordinary messages in the mailbox. (Default)

-no_acknowledge, -nack
    does not acknowledge messages that request acknowledgement.

-no_count, -nct
    does not display the message count.

-no_debug, -ndb
    disables print_mail's debugging facilities. (Default)

-no_header, -nhe
    displays no information from the message header. Only the message number, message body line count, and message body are displayed.

-no_interactive_messages, -nim
    does not include interactive messages. It is incompatible with -no_mail.

-no_list, -nls
    does not print a summary of the messages. (Default)

-no_mail, -nml
    does not print ordinary messages.

-no_reverse, -nrv
    prints the messages in ascending numeric order. (Default)

-not_own
    selects only those messages in the mailbox that were not sent by you. It requires read (r) extended access on the mailbox.

-own
    selects only those messages in the mailbox that you sent to the mailbox. It requires own (o) extended access on the mailbox.

-reverse, -rv
    prints the messages in descending numeric order.

LIST OF MBX SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table (see "Notes on Mailbox Selection by User" below).

STR
is any noncontrol argument and is first interpreted as -mailbox STR; if no mailbox is found, STR is then interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

LIST OF QUERY RESPONSES

After printing each message, print_mail asks the question

```
print_mail: Delete #N?
```

The acceptable answers are

?
prints the list of acceptable answers.

abort
exits print_mail without deleting any messages.

no, n
does not delete this message.

quit, q
deletes the indicated messages and exits print_mail; the message just printed is not deleted (see "Notes").

reprint, print, pr, p
prints the message and asks the question again.

yes, y
deletes this message (see "Notes").

NOTES

Answering "yes" to the query after a message is printed does not delete the message immediately but marks it as one for deletion.

Messages are actually deleted either after you answered the query for the last message (unless you typed "abort") or after you answered any query with "quit."

NOTES ON MAILBOX SELECTION BY USER

A user's default mailbox is specified in the form Person_id.Project_id. For an entry in the mail table, STR is usually in the form of Person_id. The mail table permits you to address mail by Person_id without knowing the Project_id of the recipient. The mail table is described in the *Extended Mail System User's Guide* (CH23) and the *Multics System Administration Procedures* (AK50) manuals.

If STR contains one period and no white space, it is interpreted as a User_id that specifies a user's default mailbox; otherwise it is interpreted as the name of an entry in the mail table.

For example,

```
-user DBuxtehude.SiteSA
```

is interpreted as a User_id that identifies a default mailbox. On the other hand,

```
-user "George G. Byron"
-user L.v.Beethoven
-user Burns
```

are all interpreted as the names of entries in the mail table: the first because it contains white space; the second because it contains more than one period; the third because it contains no period.

When interpreted as a User_id, STR cannot contain any angle brackets (<>) and must have the form Person_id.Project_id, where "Person_id" cannot exceed 28 characters and "Project_id" 32 characters. In this case, "-user STR" is equivalent to the mbx_specification "-mailbox >udd>Project_id>Person_id>Person_id.mbx."

When interpreted as the name of a mail table entry, STR cannot contain any commas, colons, semicolons, backslashes (\), parentheses, angle brackets, braces ({}), quotes, commercial at-signs (@), or white space other than spaces. The query of the mail table is performed in a case-insensitive manner. Use the display_mailing_address command to determine the actual address corresponding to STR. The address in the mail table must identify a mailbox.

Name: print_messages, pm

SYNTAX AS A COMMAND

```
pm msg_specs {mbx_specification} {-control_args}
```

FUNCTION

prints any interprocess 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_messages" was in effect.

ARGUMENTS

msg_specs
are one or more numbers or ranges. Numbers are as printed next to each message when accept_messages -hold_messages is in effect. Ranges are of the form N:M, where N<=M and both N and M are valid message numbers. You can use the keywords "first" (f) and "last" (l) as message numbers and the keyword "all" (a) as a range (equivalent to "f:l").

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

CONTROL ARGUMENTS

-after DATE_TIME
prints messages sent after DATE_TIME only.

-all, -a
prints all messages, including those held by the -hold_messages mode (see accept_messages). (Default)

-before DATE_TIME
prints messages sent before DATE_TIME only.

-brief, -bf
suppresses an error message when no matching messages are found.

-call cmdline
calls the command processor with a string of the form:

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

where:

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

number
is the sequence number of the message, assigned when you use -hold; otherwise it 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 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.

-comment STR, -com STR
prints messages with comment fields containing STR only.

-exclude STR
prints messages with text not containing STR only.

-from STR, -fm STR
prints messages sent from STR only. STR can be of the form Person.Project, Person, or .Project.

-last, -lt
prints only the latest message received. You can't use it with any other message selection arguments.

-long, -lg
prints the sender and date-time of every message, even when the same for two consecutive messages. It overrides -brief.

-match STR
prints messages with text containing STR only.

-messages, -msg
prints regular messages (nonnotifications) only.

-no_messages, -nmsg
suppresses -messages.

-no_notifications, -nnt
nullifies -notifications.

-notifications, -nt
deletes notifications only.

-new
when accept_messages -hold is in effect, prints only those messages that have not been printed before. (Default: to print all held messages)

-short, -sh
precedes consecutive messages from the same sender by "=" instead of the Person_id and Project_id, and does not print the date-time string, but only if less than five minutes have passed since the previous message. It omits the date if the current message and the previous one are received on the same date. (Default)

LIST OF MBX SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table.

STR
is any noncontrol argument and is first interpreted as -mailbox STR; if no mailbox is found, STR is then interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

NOTES

A default mailbox is created the first time you issue print_mail, read_mail, or accept_messages. The default mailbox is:

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

Messages are deleted after they are printed unless accept_messages -hold_messages is in effect); however, the last message remains available for the life of the process or until replaced by a new message. (See also 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 {control_arguments}
```

FUNCTION

prints out changes to the message of the day. The default is to print changes to the system and user_project message_of_the_day segments since the last time the command was called.

CONTROL ARGUMENTS

-all_text, -all, -a
specifies that the entire contents of the system and/or project message_of_the_day segment will be displayed, regardless of whether or not any of the messages in the segment have been previously seen.

-current_project, -cpj
prints lines from the message of the day for the project on which the user is logged in. If the project administrator has not created a message for your project, nothing is printed. (default)

-new
specifies that only unseen messages in the system and/or project message_of_the_day segment will be displayed. (default)

-project projects, -pj projects
prints new or changed lines in the message of the day for the named projects. A warning is printed if there is no message for one or more of the projects.

-system, -sys
prints lines from the message of the day created by the system administrator. (default)

NOTES

If -system, -current_project and -project are not specified, print_motd prints lines from the system message and from the message for the current project. If one or more of these arguments are given, print_motd prints lines only from those messages.

For comparison purposes, copies of project motds are stored in the default value segment with the name project_motd.PROJECT._ where PROJECT is the default user project or a project specified by the -project control argument. Project motds will be created by a project administrator and placed in the project directory with the name >udd>PROJECT>PROJECT.motd with an addname of >udd>PROJECT>PROJECT.info.

The first time that print_motd is used for a specific project, it will print the entire contents of the message-of-the-day segment. Subsequent uses will default to print those lines which have been modified or added to the message-of-the-day since the last use of the command unless the user specifies the -all_text control argument. Since a copy of each motd segment is stored in the user's value segment, project administrators should keep the size of the project motd segments to a minimum by deleting older messages as they expire.

Name: print_proc_auth, ppa

SYNTAX AS A COMMAND

```
ppa {-control_args}
```

FUNCTION

prints the access authorization of the current process and any current system privileges.

CONTROL ARGUMENTS

-all, -a
: prints the maximum access authorization of this process.

-long, -lg
: prints the site-defined long names (up to 32 characters) for the sensitivity levels and categories.

NOTES

If you supply no -long, the site-defined short names (eight characters or less) for sensitivity levels and categories are printed.

The maximum authorization printed by -all is the one that this process could have been given at login and corresponds to the maximum access class of upgraded directories that can be created by this process.

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. As an active function, returns the names of the selected request types that 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").

-brief, -bf
does not print 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). This command looks for an iod_working_tables segment in this directory.

-generic_type STR, -gt STR
lists request types of generic type STR. You can use it to support site-defined generic types.

NOTES

You can select only one control argument.

The relocation bits are interpreted one word at a time. Each line contains symbolic relocation information for two half words. Printing of duplicate lines is suppressed and indicated by the string "(repeats)".

EXAMPLES

```
!  pri >udd>Project>Person>a 100  20

         100     absolute          absolute
      (repeats)
         103     link ptr 15       absolute
         104     absolute          absolute
      (repeats)
         110     int static  15    absolute
         111     absolute          absolute
         112     int static  15    absolute
         113     absolute          absolute
      (repeats)
```

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. As an active function, returns the names of the selected request types that would have been printed.

ARGUMENTS

rqt_names
are the names of request types to be printed. You can use the star convention.

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").

-brief, -bf
does not print heading line.

-directory path, -dr path
specifies the pathname of a test directory to be used in place of the I/O Daemon Directory (>ddd>idd). This command looks for an iod_working_tables segment in this directory.

-generic_type STR, -gt STR
lists request types of generic type STR. You can use it 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 enter_output_request. The printed output includes both the user-defined request type name and its target request type name. As an active function, only the user-defined request type name is returned.

NOTES

The User_id argument specified after -access_name 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.

The enter_output_request command allows you 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 print_request_types, indented under the request type to which they apply. The active function returns the names of any user-defined request types that match the selection criteria.

Name: print_sample_refs

SYNTAX AS A COMMAND

```
print_sample_refs name {-control_arg}
```

FUNCTION

interprets the three data segments produced by the sample_refs command, and produces a printable output segment that contains the following information: a detailed trace of segment references, a segment number to pathname dictionary, and histograms of the Procedure Segment Register (PSR) and Temporary Segment Register (TSR) segment reference distributions. (See the description of the sample_refs command.)

ARGUMENTS

name
specifies the names of the data segments to be interpreted, as well as the name of the output segment to be produced. This argument can be either an absolute or relative pathname. If name does not end with the suffix srf, it is assumed.

The appropriate directory is searched for three segments with entrynames as follows:

(entry portion of) name.srf1
(entry portion of) name.srf2
(entry portion of) name.srf3

The output segment is placed in the user's working directory with the entryname:

(entry portion of) name.list

CONTROL ARGUMENTS

-brief, -bf
specifies that the detailed trace of segment references is not to be generated.

NOTES

The print_sample_refs command is able to detect a reused segment number. The appearance of a parenthesized integer preceding a segment number indicates reusage.

EXAMPLES

```
     234|6542  >udd>user>bound_alpha_|6542
(1)  234|2104  >udd>user>max35|512
(2)  234|6160  >system_library_languages>assign_|6160
```

The occurrence of the above three lines in the detailed trace indicates the following:

1. A reference was made to location 6542 in bound_alpha_. The particular component of bound_alpha_ being referenced could not be determined. bound_alpha_ was assigned segment number 234.
2. A reference was made to location 512 in max35. max35 is a component of a bound segment whose name can be determined from the segment number to pathname dictionary. The segment bound_alpha_ has been terminated and, when the segment of which max35 is a component was initiated, it was assigned segment number 234.

3. A reference was made to location 6160 in assign_. The segment of which max35 is a component has been terminated and, when assign_ was initiated, it was assigned segment number 234.

The appearance of a segment number suffix (i.e., 1, 2, etc.) indicates a component of a bound segment.

```
310       >system_library_standard>bound_ti_term_
310.1     tssi_
310.2     translator_info_
```

The appearance of the above lines in the segment number to pathname dictionary indicate that tssi_ was the first component of bound_ti_term_ to be referenced, and that translator_info_ was the second component of bound_ti_term_ to be referenced.

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_lists
: is the name of a search list. If you specify no search lists, all search lists referenced in this process are printed.

CONTROL ARGUMENTS

-expanded, -exp
: prints all keyword search paths, except -referencing_dir, and all unexpanded search paths as absolute pathnames.

NOTES

All synonyms of a search list name are printed if you give no search lists.

For a complete list of the search facility commands, see add_search_paths.

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 add_search_rules, delete_search_rules, and set_search_rules commands. The standard search rules are described in the Programmer's Reference Manual (Section 4, under "Search Rules").

Name: print_terminal_types, ptt

SYNTAX AS A COMMAND

```
ptt {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: print_time_defaults, ptd

SYNTAX AS A COMMAND

```
ptd {keys} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[ptd key {-control_arg}]
```

FUNCTION

displays system or process time-related defaults.

ARGUMENTS

key
selects which default value is to be displayed.

CONTROL ARGUMENTS

-system, -sys
requests that the system defaults be displayed instead of the process defaults.

LIST OF KEYS

date
displays the default date format. A date format shows the year, month, and day in month.

date_time
displays the default date/time format. This combines both date and time.

debug, db
displays the default status of debugging in the date/time system.

language, lang
displays the default language. Any time words in output time strings are in this language.

time
displays the default time format. A time format shows the hour, minutes, and (optionally) seconds.

zone
displays the default time zone name. Unless explicitly specified, all input time strings are interpreted relative to this zone and all output time values are expressed in this zone.

NOTES

If set_time_default has pushed any values, these are also shown. The keys specify which defaults to print. When called with no keys, all time-related defaults are displayed, except for the debugging switch, which is shown only if it is on. As an active function, it returns the current value of one of the defaults. The debugging switch is returned as "true" if debugging is on, "false" otherwise.

The values displayed are in this order: date, date_time, time, language, zone, and debug.

Name: print_ttt_path

SYNTAX AS A COMMAND

```
print_ttt_path
```

FUNCTION

prints the name of the terminal type table (TTT) segment currently in use. This is the pathname last set by a set_ttt_path command or the pathname of the default system TTT.

Name: print_wdir, pwd

SYNTAX AS A COMMAND

```
pwd
```

FUNCTION

prints the pathname of your current working directory.

NOTES

A working directory is a directory in which your activity is centered. Its pathname is remembered by the system so that you need not type the absolute pathname of segments inferior to that directory.

See the change_wdir, change_default_wdir, and working_dir commands. See also "Search Rules" and "Pathnames" in the Programmer's Reference Manual.

Name: probe, pb

SYNTAX AS A COMMAND

```
pb {procedure_name} {-control_args}
```

FUNCTION

provides symbolic, interactive debugging facilities for programs compiled with PL/I, FORTRAN, Pascal, COBOL, or ALGOL-68. Its features permit you to interrupt a running program at a particular statement, examine and modify program variables in their initial state or during execution, examine the stack of block invocations, and list portions of the source program. You can invoke external subroutines and functions, with arguments as required, for execution under probe control. You can call probe recursively.

ARGUMENTS

procedure_name
is the pathname or reference name of an initiated program. If you don't give it, the procedure owning the frame in which the last condition was raised is assumed.

CONTROL ARGUMENTS

-handle_conditions
sets up a handler for any conditions signaled while in probe that prints an appropriate message and prevents faulting. (Default)

-no_handle_conditions
does not set up the handler.

OVERVIEW OF PROCESSING

When you invoke probe, it accepts requests from you. A probe request consists of a keyword (or its abbreviation) that specifies the desired function and any arguments required by the particular request. Requests are separated from each other by newlines or semicolons.

You can give a series of requests in the form of a request list. This is used in breakpoint request lists and conditional execution lists. Here, each request is separated by semicolons; for example:

```
! value a; v b; continue
```

Probe at all times has a "current language." It communicates with you in terms appropriate to the language of the procedure being examined. The syntax of an expression and the form of probe's output vary from language to language. To use probe to the fullest, you must compile a program so that the object segment produced has both a symbol table and a statement map (see "List of Probe Terms" below). A symbol table and statement map are produced for the languages supported if you give -table to the compiler. You can also compile a program with -brief_table, which produces only a statement map. In this case you can retrieve information about source statements and where the program was interrupted and can set breakpoints, but can do little else.

Don't compile with -optimize the program being probed because compiler optimization can cause program variables to appear different than they actually are and thus cause program statements to behave unpredicatably.

NOTES ON COBOL

Probe uses a COBOL-like syntax when the current language is COBOL. The current language is COBOL whenever the current source line is in a COBOL program. You can use the hyphen to form names, as it is in COBOL. You can qualify aggregate names in the COBOL manner (c of b of a) as well as in the PL/I manner (a.b.c). An aggregate is expressed by displaying the values of each of its members.

NOTES ON FORTRAN

When the current source line is a FORTRAN statement, probe is in fortran mode, where logical variables are displayed as ".true." or ".false." instead of as "1"b or "0"b. Case is ignored if you compiled the program with -card or -fold. You can use the relational operators .le., .lt., .eq., .ne., .gt., and .ge..

NOTES ON PASCAL

Invoked on a Pascal program, probe understands all the Pascal data types, including enumerated types, typed pointers, sets, records, files, and user-defined types, as well as the Pascal built-in functions chr, eof, eoln, false, nil, ord, and true.

Array indices are enclosed in brackets, e.g., "a[i,j]". Cross-section ranges are written with "..", as in "a[first..last]". References to record fields must specify all levels; implicit level names are not allowed; for example, you can't abbreviate "a.b.c.d" as "a.d" as you can sometimes do with PL/I structure elements.

Pointer values are written with a circumflex (^) as the up-arrow, for example "p^" to indicate the value that p points to. String constants are enclosed in apostrophes: 'This is a string'. The two boolean values are "true" and "false".

The notation

```
p -> type_id
```

denotes a value of type type_id located at p, where p can be any Pascal-typed. pointer, probe pointer variable, or pointer constant.

The probe statement

```
let p1 = p2
```

is allowed for p1 and p2 being any kind of pointer value, i.e., Pascal-typed pointer, pointer constant (for p2 only), or probe pointer variable.

These extensions have been implemented so that you can write, for example,

```
dcl p ptr
let p = first
while p <> nil : (v p -> data; let p = p -> data.next)
```

which can be useful if you want to display a list of records of type "data," chained by their field "next."

PROBE POINTERS

Two internal "pointers" are used by probe to keep track of the program's state: the "source" pointer, which indicates the current source program statement, and the "control" pointer, which indicates the current control point. The values of these pointers can be obtained by using the "where" request. The source pointer is set by using the "position" or "use" request. The control pointer is set when you enter probe.

The source pointer always indicates some location in a program. If the program is active, then the source pointer also indicates a stack frame associated with the program. If you compile the object segment with the "table" options, then a source line number is available; otherwise, the location is indicated by an octal offset from the base of the segment to an instruction.

This page intentionally left blank.

If you compile the program with a "table" option, probe obtains the text of source lines from the program's source segment (NAME.pl1, NAME.fortran, etc.). This is either the segment from which the program was originally compiled, if it exists, or the first segment with the appropriate name found using the "probe" search paths. If no source segment can be found, the position and source requests are unable to print source lines.

The language of the source line is the one probe uses with you. The meaning of a block depends on the language: for a PL/I program it specifies the smallest begin block or procedure that contains the source line; for a FORTRAN program it specifies the program or subprogram on which the statement occurs; for a COBOL program it indicates the program-id of the containing program. The frame specifies a stack frame associated with the block. When there are several invocations of the same block on the stack, the frame distinguishes between them. If there is no activation of the block, the frame portion of the source pointer is null; in this case, certain types of storage (i.e., PL/I automatic) are not defined.

The control pointer indicates the last location executed in the procedure first examined and whether it took a fault or was called out (unless it was inactive, in which case it is the location of the entry statement). If you invoked probe by a breakpoint, the control pointer indicates the statement where the break occurred.

1. If you invoke probe from a breakpoint, the control pointer is set to the line where the break occurred.

2. If you invoke probe from the command line and specify a procedure_name, then if the procedure is active the control pointer is set to the last line executed in the most recent invocation of that procedure.

3. If the procedure in the command line is not active, the control pointer is set to the entry statement for the procedure.

4. If you supply no procedure_name and there is a QUIT signal or condition frame on the stack, the control pointer is set to the location being executed when the condition was signaled.

5. If you give no procedure_name and there are no condition frames on the stack, the last line executed in the most recent frame is used (this is usually the command processor).

Information about programs being debugged is stored by probe in your home directory in a segment called Person_id.probe, where Person_id is your log-in name. This segment is created automatically when needed. Don't delete it because probe will be unable to reset any breaks it has set.

RESTRICTIONS ON INPUT LINES

A probe input line cannot contain unbalanced parenthesis or quotes. This means that a request or request list must fit on one line. It cannot contain a newline character. If a long line must be typed, the Multics escape convention of placing a backslash before the newline can be used. If the newline character is needed (in a character string constant, for example), the escape sequence (\012) can be entered instead.

PROBE REQUESTS

The following pages present the format and function of each probe request, giving the name of the request, any abbreviated form, and its arguments (required and optional). Optional material is enclosed in braces.

Each request that takes arguments is shown with examples of its use. Examples may be in the syntax used for PL/I, FORTRAN, or COBOL. If an example does not make sense to you, it may be in another language.

The following items are used throughout the requests section:

CROSS-SECTION
Probe provides cross-sections to allow reference to more than one element of an array at a time. To reference the ith through jth elements of an array, the cross-section reference is i:j (for Pascal, i..j). The asterisk (*) is used to reference all elements of an array.

EXPRESSION
Probe expressions are made up using the arithmetic operators + (addition), - (subtraction), * (multiplication), and / (division). Precedence of * and / over + and - is used in evaluating, and parentheses are used to alter the order of evaluation.

COBOL "abbreviated" expressions are not supported.

LINE
identifies a location in an object segment and possibly the source line associated with that location. A LINE can be a label known in the current procedure, the name of an external entry, a line number in the source file for the current procedure, $b (which denotes the current break), or $c (which denotes the current line).

Source line numbers consist of a source file number and a line number. The file number is optional; if not specified, the main file (file 0) is used. You can find file numbers by examining the compilation listing. The first include file is file number 1, and so on. The second part of a source line number is the actual line in the file. This number is found printed on the left edge of a compiler listing. It is also the number you would use with an editor.

Note: Since FORTRAN labels are numbers, which probe also uses to specify line numbers, FORTRAN labels must be preceded by a dollar sign; for example, "100" is a reference to line number 100, but "$100" is a reference to the line whose label is 100.

COBOL allows the use of labels that look like numbers. When a number is used for a LINE, it refers to a source statement line number, not a label. But if the number is preceded by a dollar sign, it is treated as the name of a paragraph label.

N, M
are positive, unsigned integers.

OBJECT
is a path name or reference name of an entry point into some object segment.

PATH
Some probe requests accept a Multics pathname as an argument. If a PATH contains characters other than the letters of the alphabet, the digits 0–9, or the characters ">", "<", ".", ",", "-", "_", "$", "/", it must be enclosed in quotes.

REGEXP
is a qedx regular expression. For details, see the qedx command.

REQUEST
is any probe request (or list of requests).

STRING
Probe recognizes character strings and bit strings. You can use the quote character and the apostrophe (for FORTRAN and Pascal) to delimit the string. The maximum length of a string is 256 characters. Strings can contain any character. To enclose the quote character in a string, double it; for example, "a""b" represents the string a"b containing a single embedded quote character.

LIST OF BASIC REQUESTS

`arguments, args`
Usage: args
args N
args OBJECT

displays the names and values of the arguments to the current procedure or a specified procedure invocation. If you supply N, stack level N is used. If you give OBJECT, the last stack frame for the procedure OBJECT is used.

`handlers`

Usage: handlers
handlers N
handlers OBJECT

displays the condition names and actions of the condition handlers established by the current procedure or a specified procedure invocation. If you give no N, stack level N is used. If you supply OBJECT, the last stack frame for the procedure OBJECT is used.

Information about a handler is printed in the following format:

```
on CONDITION_NAME {(FILENAME)} {snap} ACTION
```

where ACTION is one of the following:

call PROCEDURE_NAME
begin block at line N
system

`help {TOPIC}`

Usage: help
help *
help TOPIC

prints information about probe. If you invoke it with no argument, it prints general information about probe; if the argument is an asterisk, it prints a list of all topics for which info exists; otherwise it prints information about TOPIC.

`let, l`

Usage: l VARIABLE = EXPRESSION
l CROSS-SECTION = EXPRESSION

sets the specified variable or cross-section of array elements to the value of the expression (see "Syntax of a Variable" below). If the variable and expression are of different types, conversion is performed according to the rules of PL/I. Array cross-sections are expressed as in the value request. One array cross-section cannot be assigned to another nor can structures be assigned to as a whole. Certain data types cannot be assigned to any type other than their own (e.g., area, file).

Note that because of unpredictable compiler optimization, the change sometimes may not take effect, even though the value request shows that the variable has been altered.

`list_builtins, lb`

prints a summary of all builtins and their meanings (see "Probe Builtins" below).

`list_help, lh`
prints a summary of all probe topics for which there is an info.

`list_requests, lr`
prints a summary of all probe requests.

`list_variables {NAME}, lsv {NAMES}`
lists all variables that have been declared by the "declare" request in the current invocation of probe, along with their types and values. If you supply one or more NAMEs, only those variables are listed; otherwise, all defined variables are listed.

`quit, q`
returns the current level of probe. If there is more than one invocation of probe on the stack, you may still be in probe, although at a lower level; if there is only one, quit returns to command level.

`stack, sk`
Usage: sk {{M ,} N} {all} {long}

displays your stack, showing block names and line numbers (if known) last executed in each frame, and conditions raised. Each frame is displayed with a number, which is the "level number," and used in various requests to indicate a stack frame.

This request traces the stack backwards and displays the first N frames. If you don't supply M, the highest numbered frame is the first frame displayed, otherwise the Mth frame is the first displayed. If you don't give N, all frames are displayed and you can't specify M.

If you supply long, the following information is printed for each frame: block name, offset within the component, line number (if known), segment pathname, and offset within the segment. If you don't give long, only the block name and the line number (if known) are printed.

If you specify all, "support" frames are included in the trace; they are skipped by default. (See "Notes on Support Frames" below.)

Examples:

`sk`	displays the whole stack.
`sk 3`	displays three frames starting with the current one.

`symbol VARIABLE {long}, sb VARIABLE {long}`
displays the attributes of the variable specified and the name of the block in which it is declared. If the size or dimensions of the variable are not constant, an attempt is made to evaluate the size or extent expression; if the value cannot be determined, an asterisk is displayed instead. If "long" appears after the name of the identifier and if the identifier is a PL/I structure or COBOL record, the attributes of all members of the structure or record are displayed as well.

`value, v`

Usage: v EXPRESSION
v CROSS-SECTION

displays the value of the given EXPRESSION or the array elements specified by CROSS-SECTION. EXPRESSION can be a simple variable name or a more complex expression.

You specify a CROSS-SECTION by giving the upper and lower bounds of one or more subscripts. You can use an asterisk, which is equivalent to a cross-section from the lowest to highest subscript of an array. For example:

```
value arr (1:5, 3:7)
value p -> a.b(j)
value a of b of lrec
value ijptr(*,3)
```

You can use the value request with PL/I structures—but not areas—or COBOL records, in which case the value of every component is displayed as well.

You can call external functions with the value request. The argument list can involve arbitrary expressions, and the arguments are converted to the proper type if the called function specifies what type of arguments are expected.

Under your control, the value request can print identifier names in short or long form (see the modes request).

This request causes probe to identify itself by printing "probe" and the current version number on the terminal. It may be used, for example, to determine if a called routine has returned. If the current invocation of probe is not the first invocation of probe on the stack, the recursive depth is also printed. The version number is useful for determining whether the version of probe being used has certain features or bug-fixes. It should always be included in any trouble report about probe.

`.. COMMAND_LINE`

This request passes the COMMAND_LINE directly to the Multics command processor. It can never be used in a break request list or a conditional execution list. When used, it must be the only request on the line.

NOTES ON SUPPORT FRAMES

Support frames are frames that belong to a support procedure. A support procedure is a system utility that performs action directly related to the compiled user code (for example, allocation) and in which there is a high probability that any errors that arise are due to improper use by the user (allocating in a full area). When conditions arise in support frames, the standard Multics error handler attributes the error to the user code that called the support procedure, rather than the support procedure.

The probe "stack" request does not display support frames unless invoked as "stack all." Support frames are given incremental numbers between those of the surrounding frames (6.1, 6.2,..., 6.11,...).

LIST OF SOURCE REQUESTS

The source pointer is used to indicate a block in a program (to resolve variable name conflicts), a stack frame (to resolve separate invocations of a block), and a statement (to be printed). You can display its value with the where request, can change the value with the position or use request, and can print the source line pointed to with the source request.

position, ps
sets the source pointer to the statement specified, and prints the statement (see the use request).

source, sc
Usage: sc {N}
sc path PATH

The first form prints source lines of the current procedure, beginning with the current source line. If you omit N, then one statement is printed. The source pointer remains unchanged.

A statement can take up many lines, and there may be blank lines (or nonexecutable source lines, such as comments or declarations) between statements. Although you can set the source pointer only to a line for which code is generated, you can display these extra lines along with the statements. If N statements are displayed, any nonexecutable lines between the first and the last are also displayed.

The second form gives the pathname of the source segment for the current procedure. Multics object segments contain within them the absolute pathnames of the source segments used to compile them. Sometimes these segments have been moved by the time the object segment is being debugged, and when probe wants to locate them, it fails. When it does, it informs you that the source cannot be located; in that case, give the path of the source after the path argument.

If you give no PATH, the source segment used is either the one you originally compiled the program from or the first segment with the appropriate name (NAME.pl1, NAME.fortran, etc.) found using the probe search paths. If no source segment can be found, probe is unable to print source lines.

use
sets the source pointer to the statement specified. Unlike the position request, use does not prints the source line positioned to.

You can use this request in any of the following forms:

use
with no argument, source pointer is reset to initial value (same as control pointer).

use LINE
specifies a line in the current procedure.

use +N
specifies the statement N statements after the current one.

use -N
specifies the statement N statements before the current one.

use line +N
specifies the statement N lines after the current one.

use line -N
specifies the statement N lines before the current one.

use level N
specifies the last line executed in the block at stack level N.

use level +N
specifies the last line executed for stack level (this + N).

use level -N
specifies the last line executed for stack level (this - N).

use OBJECT
specifies the last line executed in the most recent invocation of OBJECT.

use STRING
specifies the next statement containing the literal string.

use /REGEXP/
specifies the next statement that matches the qedx regular expression REGEXP.

It is possible to set the source pointer only to an executable statement. Every language has nonexecutable statements, for example, blank lines, comments, and declarations. (See the where request and "LINE.")

In PL/I, names, including labels, are not known (available for probe) when the source pointer is outside the block in which they are declared. This means that probe cannot find a label in an internal procedure or begin block unless that block is now the "current block". The label can still be found by a search for a character string that appears in the labeled line, for example, "label". The source segment used is either the one you originally compiled the program from or the first segment with the appropriate name (NAME.pl1, NAME.fortran, etc.) found using the "probe" search paths. If no source segment can be found, probe is unable to print source lines.

`where, wh`
displays the values of the probe pointers. If invoked with no arguments, it displays the values of the source and control pointers. The following are also allowed:

wh source, wh sc
prints the value of the source pointer.

wh source path, wh sc path
prints the pathname of, the segment referenced by the source pointer (i.e., the source segment currently in use).

wh control, wh ctl
wh object
prints the value of the control pointer.

wh control path, wh ctl path
wh object path
prints the pathname of the segment referenced by the control pointer (i.e., the current object segment).

wh path
prints pathnames of segments referenced by the source and control pointers.

NOTES ON BREAKPOINTS

A breakpoint is a list of one or more probe requests associated with a source statement. When the statement of a breakpoint is executed, the user program is suspended and probe executes the requests associated with the breakpoint. When the requests of the breakpoint have executed, the program resumes (unless one of the requests is a "quit", "halt" or "goto"). A breakpoint can be set "before" or "after" a given line. The difference is that a break before a line is executed immediately before the line is executed, and a break after a line is executed after the line has been executed. Therefore if a transfer is made at line X of a program, a break before line X is executed, but a break after line X is not. If a transfer is made to the statement on line X, a break set before line X is executed, but one set after line X-1 is not. If statement X assigns a new value to a variable, a break set before X is executed before the variable changes, but one after X takes effect after the variable has been changed.

No breaks can be set after any COBOL statement due to restrictions of the compiler. Multiple requests in a breakpoint request list are separated by semicolons. A breakpoint list of more than one request must be enclosed in parentheses; for example:

```
b 495:(v x(1);v y(1);stack 3)
```

which invokes the "before" ("b") request to set a break at line 495 consisting of the three requests "v x(1)", "v y(1)", and "stack 3". Note that the "if" request for writing a conditional break does not use parentheses; for example:

```
b 495: if x(1)<y(1): (v x(1);v y(1))
```

which sets a break executing 2 requests if x(1)<y(1), and

```
b 495:(v x(1);if x(1)<y(1):v y(1))
```

which sets a break that always prints the value of x(1) but prints the value of y(1) only if x(1)<y(1).

Requests that transfer control (continue, continue_to, goto and step) should only be used at the end of a breakpoint request list, since any requests following them are not executed.

Breakpoints that print more than one variable value can benefit from using the call request instead of value; for example:

```
b 221: call ioa_ ("J=^d, x(J)=^5.2f^[, BUF=^a^]", i, x(i), buf_sw, buf)
```

LIST OF BREAK REQUESTS

The next requests deal with setting, printing information about, and resetting breakpoints.

`after, af, a`

Usage: a {LINE} {:REQUEST}
a /REGEXP/ {:REQUEST}

sets a breakpoint after the location in the object segment given by LINE. The default for LINE is the current line ($c) and the default for REQUEST is "halt". If you use /REGEXP/ instead of LINE, a break is set after every line matching the qedx regular expression REGEXP. (Type "breaks".)

A break set after a line happens after all effects of the statement are completed. If the statement sets a variable, the variable will have its new value by the time the break is executed. Depending on the nature of the code generated by the translator for the statement, a break set after a line sometimes cannot be executed. This is often the case for do loops and if-then-else constructions and always the case for return statements and gotos.

The "after" request cannot be used on COBOL statements.

`before, be, b`

Usage: b {LINE} {:REQUEST}
b /REGEXP/ {:REQUEST}

sets a breakpoint before the location in the object segment given by LINE. The default for LINE is the current line ($c) and the default for REQUEST is "halt". If you use /REGEXP/ instead of LINE, a break is set after every line matching the qedx regular expression REGEXP.

`reset, r`

resets breakpoints set by probe at selected lines in selected procedures. You can use it in any of the following forms:

`r`
resets the break at the current statement.

`r LINE`
resets the break at LINE in the current procedure.

`r STRING`
resets breaks at all lines containing STRING in the current procedure.

`r /REGEXP/`
resets breaks at all lines matching the qedx regular expression.

`r OBJECT`
resets all breaks in the object segment OBJECT.

`r -all`
resets all breaks set by you in all segments.

`r *`
same as reset -all.

Precede LINE, "STRING", and /REGEXP/ by "before" to set only breaks before statements or "after" to set only breaks after statements.

As breaks are reset, their line numbers are printed, unless you supply -brief.

Examples:

`r -all`	reset every break probe can find.
`r a $259,1`	reset the break after the first statement after the line labelled "259".
`r -bf`	reset all breaks in the current procedure.
`r <my_dir>work`	reset all breaks in procedure named.

`status, st`
displays line numbers of breaks and optionally the break request list. You can use it in one of the following forms:

`st`
displays all breaks in the current procedure.

`st LINE`
displays break at LINE in current procedure.

`st OBJECT`
displays all breaks in procedure OBJECT.

`st -all`
displays all breaks set by you in all procedures.

`st *`
displays the pathnames of all procedures with breaks set by the you.

The break request list of the break is displayed by default for status LINE and omitted for all other forms. The -long control argument prints the break list; -brief suppresses it.

As for "reset," you can distinguish a break before a line from a break after a line by prefacing the LINE with "before" or "after."

Examples:

`st >udd>m>my_seg`	lists all breaks in my_seg.
`st 35 -lg`	prints the list of breaks before and after line 35.
`st b 7`	lists only the break before line 7.
`st -all`	lists all segments in which breaks have been set.

REQUESTS USEFUL IN BREAKPOINT REQUEST LISTS

`halt, h`
invokes a new probe command level, with the control and source pointers set to the line of the breakpoint. This is the default break request. After a subsequent continuation, probe resumes interpreting the break request list that contains the halt. When the list is empty, your program is resumed. This request has no effect when typed at probe command level. If a break is set with no break request list supplied by you, halt is assumed.

Example:

```
b12:if K>0:h          this breakpoint stops if K>0.
(v a; h; v a)         this list displays the value of a before and after stopping.
```

`pause, p`
is like halt in that it suspends execution of the breakpoint request list and causes probe to read from the terminal; unlike halt, when the breakpoint is continued (by the continue request), the break is immediately reset. The effect is a one-time-only temporary break. This request is used to implement the step and continue_to requests.

FLOW OF CONTROL REQUESTS

Requests are provided for selected execution of program statements. The user can resume execution after a break, call external procedures, or perform explicit "goto"s.

`continue, c`
restarts a program that has been suspended by a probe breakpoint. If this request is used in any other context, probe returns to its caller, which is usually command level.

`continue_to LINE, ct LINE`
inserts a temporary breakpoint before the LINE specified, then continues. The effect is as if the user had typed the following:

```
before LINE: pause
continue
```

When the user resumes after a temporary break, it is automatically reset.

`call OBJECT (ARGUMENTS), cl OBJECT (ARGUMENTS)`
calls the external procedure named with the arguments given. ARGUMENTS should be a list of arguments to the called procedure, separated by commas. If the procedure expects arguments of a certain type, those given are converted to the expected type. The value request (see above) can be used to invoke a function, with the same sort of conversion occurring. If the procedure has no arguments, an empty argument list "()" must be given.

Examples:

PL1:
```
call sub ("abcd", p -> p2 -> bv, 25o, addr(k))
```
COBOL:
```
call eat-master (a of b of new-unit, REC-LEVEL)
```
FORTRAN:
```
call gamma (43, marigold(i), substr(cs,3))
```

`goto LINE, g LINE`
leaves probe and resumes execution of your program at LINE. Don't use this

request if the LINE is in a program that is not active. Because of unpredictable compiler optimizations, this request may be dangerous if you compile the program with -optimize.

Examples:

```
g label_var          transfer to value of label variables.
g action (4)         transfer to value of label constant.
g 110                transfer to statement on line 110.
g $110               transfer to line with label 110.
g $c,1               transfer to the statement after the current one.
```

`step s`

tries to step through the current program one statement at a time. If the program has been stopped before line N, a break is set before line N+1. If you are stopped after line N, the break is set before line N+2. These breaks contain "pause" as their sole request list and thus are self-resetting. If the statements being stepped do not execute in sequence, the stepping may be unsuccessful. PL/I and FORTRAN do-loops and conditional statements in all languages do not execute sequentially.

CONDITIONAL REQUESTS

`if PREDICATE : REQUEST`

evaluates its PREDICATE argument. If the result is true, REQUEST is executed, otherwise it is skipped. This request is useful in break request lists, where you can use it to cause conditional breakpoints; for example, to stop whenever the variable "odin" equals 1, the break request list can include "if odin = 1:halt".

`while PREDICATE : REQUEST`

repeatedly executes REQUEST, testing the conditional expression PREDICATE before each execution. REQUEST can be a single request, or several probe requests, enclosed in parentheses and separated by semicolons.

REQUESTS TO CONTROL PROBE

You can manipulate probe's behavior in a few ways: you can control the length of error messages and the amount of printing done by breaks and by the value request; you can specify explicitly the current language; you can control the streams used by probe for input and output. Unlike other requests, the effects of input_switch, output_switch, input_description and output_description are static to the process and remain from one probe invocation to the next until reverted.

`input_description, ids`
takes an attach description and creates an IOCB, then makes probe read all its input from this IOCB until further notice. The IOCB is destroyed the next time input_description or input_switch is invoked. Example:

```
ids tty_ stereo_console
```

causes further input to be read from the device stereo_console.

`input_switch {SWITCH}, isw {SWITCH}`
causes probe to take all further command input from the switch named. SWITCH is the name of an already-attached IOCB. If you supply no SWITCH, user_input is used. If there are any other requests in the input line or break request list that contain this request, they are ignored without comment. Input is read from the switch until either a new input_switch request is read or all available characters are processed, in which case a message is printed and input is reset to user_input. If any errors occur, input is reset to user_input. SWITCH must be attached and open before you give this request.

`language {LANG}, lng {LANG}`
given the name of one of the languages supported by probe, sets the current language mode; otherwise displays the name of the current language mode. Names accepted are: pl1, fortran, ft, cobol, and algol-68.

`modes {MODE VALUE}, mode {MODE VALUE}`
sets various modes internal to probe that change the way it functions, where MODE is the name of the mode to set, and VALUE is the new value. A mode is a probe variable that specifies how a particular function behaves. The values of all modes can be displayed by using "modes" with no arguments. If you set conflicting modes, the last one in the request determines the setting of the mode. If you give no arguments, the current modes are printed.

Most modes can be set to a value that is either a LENGTH or a BOOLEAN. A LENGTH is "long" (lg), "short" (sh) or intermediate, or "brief" (bf) and is used to specify the amount of printing to be done by probe; in some cases, "short" is synonymous with "brief". A BOOLEAN is used to turn a feature on or off and can be "yes", "on", or "true" or "no", "off", or "false".

MODES can be any combination of the following:

`error_messages LENGTH, em LENGTH`
controls the length of the text used for an error message. (Default: long)

`meter`
controls the printing of meter values when breaks are hit. The meter values are the number of minutes and seconds of real time ("TIME"), the number of seconds of virtual CPU time ("VCPU"), and the number of page faults ("PAGE FAULTS") since the last break was restarted or since "mode meter on." These values do not include any overhead from probe itself. Example:

```
Stopped after line 37 of test_program
   Time = 3 min 42 sec, Vcpu = .057 sec, Page faults = 103
```

These values do no include any overhead from probe itself.

`output_description, ods`
takes an attach description and creates an IOCB, then causes probe to write all its output on this IOCB until further notice. The IOCB is destroyed the next time you invoke output_description or output_switch. Example:

```
ods vfile_ probe_trace.output
```

writes further output to the segment probe_trace.output.

`output_switch {SWITCH}, osw {SWITCH}`
with no argument, writes probe output (except for error output) on the default switch, user_output; otherwise, writes nonerror output on a specified switch. SWITCH is the name of an already-attached IOCB.

`prompt BOOLEAN`
controls whether or not probe prints a prompting string on your terminal when it is listening for a request (see "prompt_string"). (Default: off)

`prompt_string STRING`
specifies the string to be used for prompting. The initial value is "probe^[(^d)^] ". STRING is used in a call to ioa_$nnl, where the first argument is a bit (1) that is on if the current invocation of probe is recursive and the second argument is the current probe depth.

`qualification LENGTH, qf LENGTH`
controls the format of variables names as printed by "value": for brief, prints only the last name of a structure (default); for long, prints names with full qualification. COBOL and FORTRAN are always printed briefly, regardless of the value of this mode.

`truncate_strings`
truncates character and bit string values to 200 characters or bits, printing "<MORE>" at the end if the string is longer than 200. (Default: on)

`value_print LENGTH, vp LENGTH`
controls whether or not the value request prints the name of a variable: for brief, names are never printed; for short, names of array and structure elements are printed (default); for long, names are always printed.

`value_separator STRING, vs STRING`
controls the string printed by the value request between the name of a variable and its value. Only the first 32 characters of STRING are used. (Default: "=")

MISCELLANEOUS REQUESTS

`declare, dcl`
Usage: dcl NAME TYPE {-force} {external} {defined EXPR}

creates new probe variables and declares external variables. NAME is the name of a new variable to be created or the name of an external variable if you supply external (ext). TYPE is a keyword specifying the type of the variable. If you don't give external and a variable named NAME already exists, you are queried whether to delete the old one. The -force (-fc) control argument deletes the old one in this case without a query. There are four possible values for TYPE: The first is equivalent to PL/I fixed bin (35), Fortran or Pascal integer, and COBOL comp-6; it can be referred to as fixed, integer, int, or comp-6. The second is equivalent to PL/I float bin (27) or Fortran real; it can be referred to as float or real. The third is equivalent to PL/I aligned pointer (Multics ITS ptr); it can be referred to as pointer or ptr. The fourth is PL/I unaligned (packed) pointer; it can be referred to as "pointer unaligned" or "ptr unal".

If you give "defined EXPR", where EXPR is a variable expression designating a region of storage having one of the above TYPEs, NAME becomes a synonym for that region. This feature allows you to use a short name in place of a complicated expression.

You are warned about a name conflict if a program variable of the given NAME is known in the current block when the variable is declared. If this warning is given, you must refer to the variable with a percent sign before its name to distinguish it from the program variable.

Examples:

```
dcl gravel comp-6
dcl clover ptr -force
dcl sys_info$max_seg_size fixed -ext
dcl axi fixed defined info_ptr -> aregs (i).x.index
```

`display, ds`

Usage: ds {*} ADDRESS {FORMAT} {COUNT}

displays an arbitrary location in a selected format. The location displayed depends on the type of ADDRESS supplied. If ADDRESS is a reference to a VARIABLE, the address of the VARIABLE is the location displayed. Otherwise, ADDRESS must be an expression that evaluates to a pointer; the value of the expression gives the location to be displayed. If an asterisk appears before a pointer VARIABLE, data pointed to by the pointer's value is displayed.

FORMAT can be one of the following:

`ascii, character, ch`
N is the number of characters dumped. A non-printable character is printed as ".".

`binary, bin, binary35, bin35`
N is the number of fixed binary (up to 35) values dumped.

`binary71, bin71`
N is the number of fixed binary (36 to 71) values dumped.

`bit, b`
N is the number of bit strings dumped.

`code`
prints the error message associated with a status code.

`float, f, float27, f27`
N is the number of float binary (up to 27) values dumped.

`float63, f63`
N is the number of float binary (28 to 63) values dumped.

`instruction, i`
N is the number of instructions dumped. If the instruction has descriptors, they are dumped with the instruction.

`octal, o`
N is the number of (36 bit) words dumped.

`pointer, ptr, its`
N is the number of ITS pointers displayed.

COUNT is the number of elements displayed; the default is one. The size of an element depends on the format displayed. One pointer is 72 bits (two words), and one instruction can be as many as four words (for an EIS instruction).

Examples:

`ds * 253|100 octal 20`	dump 20 words in octal, pointed to by 253|100.
`ds foo ascii 64`	display the first 64 characters of foo.
`ds ip|0 i`	dump current instruction.
`ds error_code code`	displays error message for status code value assigned to error_code variable.

`execute STRING, e STRING`

passes the quoted STRING to the Multics command processor. It is useful in break request lists because the more convenient ".." escape to the Multics command processor is not available there. Example:

```
e "ioa_ ""stopped at a break """""
```

`object {N}, obj {N}`

displays the assembly instructions for the current source line, or for the next N source lines.

SYNTAX OF A VARIABLE

Variables can be simple identifiers, subscripted references, structure qualified references, and locator qualified references. Subscripts can also be expressions.

Spaces are significant in the names of FORTRAN and COBOL names. A FORTRAN name cannot contain embedded spaces. Case is insignificant in COBOL names in FORTRAN names when the object segment was compiled with either "-fold" or "-card".

Examples:

COBOL:

```
data-elem
log-type of gen-record (3)
gen-record.log-type(3)
```

PL1:

```
ignatz (p -> lemma - 3)
```

PASCAL:

```
arrayp^.first [6,2]
```

The block in which a variable reference is resolved is normally determined by the source pointer, but can be altered by providing a different block in brackets after the variable name. A block can be specified in the following ways:

Example:

`level N`	the block and frame at level N.
`-N`	the Nth previous invocation of the current block.
`LINE`	the block that contains LINE, in its most recent invocation.
`OBJECT`	the block named. It can be internal to the current procedure or external.

WARNING: Specifying a block explicitly does not change probe's "current language". It is possible that the block named is in another language than the current block. Even if this is so, data is referenced in terms of the current language.

SYNTAX OF A CONSTANT

The attributes of a constant are determined by the appearance of the constant. Probe recognizes arithmetic constants (fixed or floating point, binary or decimal), string constants (character or bit), and pointer constants. The maximum length of a string constant is 256 characters. You can enter bit constants in any radix from binary to hex. You can enter integers in octal by following them with a lower case "o". FORTRAN double-precision constants are not implemented.

Probe pointer constants are of the form SSS|WWW or SSS|WWW(bbb), where SSS is the segment number in octal, WWW is the word offset in octal, and bbb is the bit offset (optional) in decimal. You can replace SSS with a two-letter code to specify a pointer relative to the current stack frame (sp), linkage section (lp), text segment (tp), or instruction (ip).

Examples:

```
-123                fixed dec (3)
10b                 fixed bin (2)
45.37               fixed dec (4,2)
4.73e10             float dec (3)
4.21f10             fixed dec (3,-8)
2.1-0.3i            complex decimal (2,1)
12345670o           fixed bin (24) entered in octal
"abc"               character string
"quote""instring"   character string with embedded quote
"1010"b             binary bit string
"FA07"b4            hexadecimal bit string
"1222"b2            quatenary bit string
256|1200            pointer
232|7413(9)         pointer with bit offset
true                FORTRAN logical constant
'Nix Olympia'       FORTRAN string constant
123o                fixed bin (35), octal integer, value is
                                              83 decimal
```

Specify the segment number and word offset of a pointer in octal, but any bit offset in decimal.

PROBE BUILT-INS

Many built-in functions are provided. They can be referenced as if they were external functions, but if no argument is needed, then you can omit the argument list. You can use substr and unspec as pseudo-variables.

```
addr (A)                    octal (N)
addrel (P, N)               ptr (P, N)
baseptr (N)                 rel (P)
length (S)                  segno (P)
maxlength (S)               substr (S, N)
null ()                     unspec (A)
```

A stands for any reference to storage, N for any expression that yields a number, P for any expression that yields a pointer value, and S for any expression that yields a string.

All probe built-ins, except "segno" and "ptr", are equivalent to the Multics PL/I built-ins. The ptr built-in is like the Multics PL/I ptr built-in, but you can also supply it with a bit offset after the word offset. The segno built-in is like the Multics PL/I baseno built-in, but its result is an integer instead of a bit string.

The probe command reads numbers in decimal, so a reference to "baseptr(64)" is the same as "baseptr(100o)".

You can preface built-ins with the dollar sign to distinguish them from program variables of the same name.

Assume that the following declarations are more or less equivalent, that cs has the value "abcdef ", and that i is 2:

```
PL1:
    dcl i fixed bin;
    dcl cs char (8);
FORTRAN:
    integer i
    char*8 cs
COBOL:
    77 i usage is comp-6.
    77 cs pic a(8).
```

```
addr (i)                          the address of i.
v substr (cs, i, 3)               displays "bcd".
let substr (cs, 4, 1) = " "       sets cs to "abc ef".
v length (cs)                     displays 8.
value maxlength(cs)               also displays 8.
v baseptr (254o)                  displays 254|0.
```

LIST OF PROBE TERMS

active
: a procedure is said to be active if its execution is ongoing or suspended by an error, quit signal, breakpoint, or call. An active procedure is distinguished from one that has never been run, has completed execution or has been interrupted and aborted by a Multics release command, in that an active procedure has at least one stack frame associated with it.

automatic storage
: a storage class for which space is allocated dynamically in a stack frame upon block invocation. As a result, variables of this class only have storage assigned to them, and hence a legitimate address and value, when the block in which they are declared is active. PL/I variables, by default, belong to this class. FORTRAN variables must appear in an "automatic" statement in order to belong to this class.

block
: corresponds to a PL/I procedure or begin block or FORTRAN program or subroutine, and identifies a particular group of variable declarations.

breakpoint
: a point at which program execution is temporarily interrupted and probe requests executed.

invocation
: when a procedure is called recursively, it appears on the stack two or more times, and has storage allocated for it the same number of times. Each instance of the procedure on the stack is considered a separate and distinguishable invocation of the block. The values of automatic variables can be different in different invocations of the same block. The most recent invocation is the topmost in stack trace.

level number
: an integer used by probe to uniquely designate each block invocation (i.e., each entry in a stack trace). Level one is the first (least recent) procedure invoked. Level number is NOT necessarily the same as either of the numbers given after the word "level" in a ready message. The first of this pair gives the count of command levels in effect and gives the value N+1, where N is the number of programs (or groups of programs) whose execution has been suspended, the second gives the number of stack frames in existence and since the probe stack includes quick blocks, this number is less than or equal to the level number of the last command level in the stack trace.

quick block
internal procedures and begin blocks that satisfy certain requirements (e.g., are not called recursively, do not contain on, signal, or revert statements, etc.) have their automatic storage allocated by the blocks that call them. Hence, they do not actually have their own stack frames, but share the one of the caller. Certain system commands, such as trace_stack, ignore these blocks. The probe command, however, includes them in a stack trace and treats them as if they were the same as any other blocks. You can determine the quickness of a block from a program listing containing information about the storage requirement of the program (produced with the -list, -map, or -symbols control arguments). For example, procedure "quick" shares stack frame of external procedure "main."

stack
if a procedure A calls another procedure B, the execution of A is suspended until B returns. If B in turn calls C, this is an ordered list of procedure or subroutine calls indicating which program called which other program, and which returns to which. This ordered list is called the "stack." In probe, you can display a trace of the stack with the stack request. The list is given in top-down fashion with the most recently called procedure listed first.

```
3     C
2     B
1     A
```

The numbers are level numbers.

stack frame
when a block is invoked (that is, a procedure is called or a begin block is entered), storage is allocated for its automatic variables. The area allocated is called a stack frame, and logically corresponds to each entry in the stack.

statement map
a table in the symbol section of an object segment that relates locations in the text section (executable mode) to source line numbers. This table is produced by a language translator when you specify -table or -brief_table.

static storage
a storage class for which space is allocated once per process, effectively at the time the procedure is first referenced. As a result, variables of this class always have a legitimate address and value. Regular FORTRAN variables, and those in a common block, have static storage. You must explicitly declare PL/I variables.

support procedure
a system utility routine that provides runtime support for other procedures (e.g., the procedure that allocates storage as requested by a PL/I allocate statement).

symbol table
a table in the symbol section of an object segment that contains information about the variables (symbols) used in the program. A symbol table is produced by a language translator when you give the -table control argument.

SUMMARY OF REQUESTS

Request	*Abbrev*	*Function*
after	a	sets a break after a statement.
arguments	args	prints arguments to current procedure.
before	b	sets a break before a statement.
call	cl	calls an external procedure.
continue	c	restarts after a breakpoint.
continue_to	ct	inserts a temporary break and continues.
declare	dcl	creates probe variables, declares external variables.
display	ds	displays storage in a selected format.
execute	e	passes string to the command processor.
goto	g	transfers control to a statement.
halt	h	in break text, establishes a probe level.
handlers		displays condition handler information.
help		prints information about probe.
if		executes probe requests if condition is true.
input_description	ids	creates an IOCB and causes probe to read all its input from it.
input_switch	isw	reads probe requests from switch.
language	lng	sets probe language.
let	l	assigns a value to a variable.
list_builtins	lb	prints a summary of probe builtins.
list_help	lh	lists probe topics for which there is info.
list_requests	lr	prints a summary of probe requests.
list_variables	lsv	lists all variables that have been declared by the "declare" request.
modes	mode	controls probe's behavior.
object	obj	displays assembly instructions.
output_description	ods	creates an IOCB and causes probe to write all its output on it.
output_switch	osw	directs probe output to a switch.
pause	pa	stops a program once.
position	ps	sets the source pointer.
quit	q	returns from current probe invocation.
reset	r	deletes breaks.
source	sc	prints source lines.
stack	sk	traces the stack.
status	st	lists breakpoints.
step	s	advances one statement and halts.
symbol	sb	displays attributes of a variable.
use	u	sets source pointer.
value	v	displays value of an expression.
where	wh	displays value of probe pointers.
while	wl	executes commands while condition is true.
.		causes probe to identify itself.
..		escapes to command processor.

Name: process_dir, pd

SYNTAX AS A COMMAND

```
pd
```

SYNTAX AS AN ACTIVE FUNCTION

```
[pd]
```

FUNCTION

returns the pathname of the process directory of the process in which you invoke it.

Name: process_switch_off, pswf

SYNTAX AS A COMMAND

```
pswf switch_names
```

FUNCTION

turns off specified per-process switches.

ARGUMENTS

switch_names
: are the names of per-process switches.

LIST OF SWITCHES

256k_switch, 256ksw, 256k
: allows 256K segments, currently used by FORTRAN programs, to hold very large arrays.

Name: process_switch_on, pswn

SYNTAX AS A COMMAND

```
pswn switch_names
```

FUNCTION

turns on specified per-process switches.

ARGUMENTS

switch_names
are the names of per-process switches.

LIST OF SWITCHES

256k_switch, 256ksw, 256k
allows 256K segments, currently used by FORTRAN programs, to hold very large arrays.

Name: profile, pf

SYNTAX AS A COMMAND

```
pf {program_names} {-control_args}
```

FUNCTION

analyzes the time spent executing each source statement of a program, along with other parameters of interest, after the program is run.

ARGUMENTS

program_names
are pathnames or reference names of programs to be analyzed. Any program name that does not include "<" or ">" characters is assumed to be a reference name. You need not specify them if you use -input_file.

CONTROL ARGUMENTS

Apply to all programs specified and can be given in any order.

-brief, -bf
use it with -print to exclude from the output all information for statements that have never been executed. (Default)

-comment STR, -com STR
use it with -output_file to include STR with the stored profile data as a comment. You can also use it with -plot. Enclose STR in quotes if it contains blanks or other special characters. STR can be up to 128 characters long.

-first N, -ft N
use it with -sort to print only the first N values.

-from N, -fm N
use it with -print or -plot to begin the output with the data for line number N. (Default: 1)

-hardcore, -hard
indicates that the specified programs are supervisor (hardcore) segments. The current (internal static) profile data for such programs is retrieved from the address space of the supervisor. Install hardcore programs compiled with -profile or -long_profile by generating a Multics system tape and rebooting Multics (see generate_mst). You cannot reset (zero) the current profile data for hardcore programs. This control argument and -reset are mutually exclusive.

-input_file path, -if path
retrieves the profile data from the profile data file (pfd) specified by path. This control argument ignores any current profile data. The pfd suffix is appended to path if it is not already present. If you supply any program_names, they select a subset of the stored data for analysis; if you give none, all data stored in the profile data file is used. This control argument is inconsistent with -output_file and -reset.

-line_length N, -ll N
use it with -list to specify an output width of N characters. (Default: 132)

-list, -ls
creates a profile listing for all specified programs. The profile listing file (pfl) is given a name consisting of the first program name with the language suffix replaced by the pfl suffix. It is placed in your working directory. The information described for -print is placed in columns to the left of each source line in the profile listing.

-long, -lg
use it with -print to include in the output information for statements that have never been executed.

-max_points N, -mp N
use it with -plot to specify the maximum number of points (line numbers) to be plotted (the graphics resolution). The Multics graphics system is capable of plotting up to 1024 points. (Default: 250)

-no_header, -nhe
use it with -print to suppress column headings.

-output_file path, -of path
stores the profile data for the given program_names in the pfd specified by path. The file is created if it does not already exist and is overwritten if it does. The pfd suffix is added to path if it is not already present. The profile data is stored in a format acceptable to -input_file. The format of pfd data files is described by the PL/I include file pfd_format.incl.pl1. The stored data is determined by -comment, the program_names specified, and whether you did the compilation using -profile or -long_profile. The compiling program name is saved in the profile data file. If program_name specifies a bound object segment, profile data about each component of the bound object segment is saved.

-plot STR
plots a bar graph, on any supported graphics terminal, of the values of the specified field STR. STR can be any of the fields in the "List of Fields" section below. This control argument requires that your site has installed the Multics Graphics System and that you have executed the setup_graphics command. (See the *Multics Graphics System*, Order No. AS40.)

-print, -pr
prints the following information for each statement in the specified program(s):

1. Line Number

2. Statement Number
   if more than one statement on the line.

3. Count
   the number of times the statement was executed.

4. Cost
   an approximation to the accumulated execution time for the statement. Equal to the number of instructions executed plus 10 times the number of external operators called.

5. Stars (asterisks)
   an indication of the percentage of total cost (or time, for long_profile data) used in the statement. The number of stars is selected according to the following table:

4 stars:	20% to 100%
3 stars:	10% to 20%
2 stars:	5% to 10%
1 star:	2.5% to 5%
no stars:	0% to 2.5%
one period:	Statement was not executed.

6. Names of all external operators called by the statement.

For -long_profile (actual accumulated time) data, item 4 is changed to the following:

4a. Time
actual execution time for the statement in virtual CPU microseconds, including all time spent in any operators or subroutines invoked by the statement.

4b. Average Time
Time divided by Count (the average execution time for one execution of the statement).

4c. Page Faults
page faults incurred in executing the statement.

-reset, -rs
resets all current profile data for the named program(s). When specified, the resetting is done last if -print, -list, -plot, or -output_file is also given.

-search_dir path, -srhd path
use it with -hardcore to add path to an internal search list of hardcore object directories. You can supply up to eight directories. If you give no search list, >ldd>hard>o is searched for copies of the specified program(s).

-sort STR
use it with -print to sort in descending order profile information of the specified field STR, which can be any of the fields in "List of Fields."

-source_dir path, -scd path
use it with -list when the source segments to be listed have been moved from the directories in which they were compiled. If given, only the directory specified by path is searched for source segments.

-to N
use it with -print or -plot to end the output with the data for line number N. (Default: line number of the last executable statement)

LIST OF FIELDS

count
the number of times the statement was executed.

time
the vpcu time of the statement (available if you compile the program with -long_profile).

cost
the approximate cost of the statement (available if you compile the program with -profile).

page_faults (pfs)
page faults taken during the statement (available if you compile the program with -long_profile).

NOTES

If you supply no control arguments, the default ones are -print and -brief.

Object segments that are created with the PL/I compiler can record up to 19 hours of statement time.

Compile the program using cobol, fortran, and pl1 commands' -profile, or using pl1's -long_profile. The latter is used to acquire exact elapsed time statistics and is much more expensive than -profile.

When analyzing several runs of the same program(s) on various test cases, select -reset. If you give no -reset, the current profile data is accumulated (added) for all runs.

If you supply several identical control arguments, only the last one is used, except for -search_dir.

The current data acquired by programs compiled with -long_profile may have small perturbations due to asynchronous events outside the control of the data acquisition mechanism. Therefore, -long_profile results are most reliable when obtained from long-executing programs or from multiple executions of the same program.

The execution time for -long_profile programs can be up to 10 times as long as normal due to the overhead of acquiring CPU time and paging data from the supervisor. This overhead is subtracted from the current profile data before any further processing is done.

There are two forms of profile data: current and stored. Current data is in a form suitable for direct incrementing by the program(s) being analyzed and is stored using the pl1 internal static storage class. Current profile data (except for hardcore programs) can be zeroed by -reset. Stored profile data is permanent data as stored by -output_file. The pfl's and pfd's are automatically stored as multisegment files if they are too large to fit into a single segment. This feature allows very-large bound object segments to be analyzed and very-large source segments to be listed.

Profile data generated from statements in include files are printed only if you specify no -from or -to. Include file profile data cannot be plotted. Include files that generated profile data are listed after the main source program. If you give -source_dir, include files are searched for first in the specified source directory and then in the directory in which they are found when the program is compiled.

EXAMPLES

The following command lines compile a PL/I program with -profile, execute the program once to acquire current profile data, and print the five most expensive statements.

```
! pl1 factorial -profile
  PL/I 24c
! factorial
  n       n!
  1       1
  2       2
  3       6
  4       24
  5       120
  6       720
  7       5040
  8       40320
  9       362880
  10      3628800

! profile factorial -sort cost -first 5

  Program: factorial
  LINE STMT   COUNT       COST STARS   OPERATORS
    20           45       1710 ****    call_int_other, return
    10           10        440 ***     call_int_this
                                           call_ext_out_desc
    18           55        220 **
    18           10        120 *       return
    11           10         50
  -------
  Totals:       144       2604
```

The following command line saves the current profile data in factorial.pfd.

```
! profile factorial -of factorial
```

The following command line creates a profile listing in factorial.pfl from the source segment factorial.pl1 and the profile data file factorial.pfd. The listing is prepared for a printer with only 50 columns.

```
! profile -if factorial -ls -ll 50
```

The contents of the profile listing segment factorial.pfl is shown below. Profile information for a then clause has the same line number and statement number as the if statement.

```
Profile listing of >udd>Work>Mann>r>factorial.pl1
Profile data file >udd>Work>Mann>r>factorial.pfd
Created by Mann.Work.a on 12/11/84  1547.5 edt Tues
Date: 12/11/84  1548.3 edt Tues
Total count: 144   Total cost: 2604

  COUNT     COST STARS LINE SOURCE
                        1 /* Program to print a table of the first 10
                          factorials. */
                        2
                        3 factorial:
                        4      procedure;
                        5           declare n fixed binary (35);
                        6           declare ioa_ entry options (variable);
                        7
      1        19       8           call ioa_ ("n^-n!");
      1         2       9           do n = 1 to
     11        33          10;
     10       440 ***     10                call ioa_ ("^d^-^d", n, fact (n));
     10        50         11           end;
      1        10         12           return;
                        13
                        14 fact:
                        15      procedure (n) returns (fixed binary (35));
                        16           declare n fixed binary (35);
                        17
     55       220 **      18           if n <= 1
     10       120 *
                        19           then return (1);
     45      1710 ****    20           else return (n * fact (n - 1));
                        21      end fact;
                        22
                        23      end factorial;
```

Name: program_interrupt, pi

SYNTAX AS A COMMAND

pi

FUNCTION

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

NOTES

To abort a subsystem request, use the quit (break) key to interrupt execution and then issue program_interrupt. If the subsystem supports the use of this command, it aborts the interrupted request and asks you for a new one; if it does not, the command prints an error message. You 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 your stack, the stack is searched for a program that supports program_interrupt and any intervening programs are released.

Name: progress, pg

SYNTAX AS A COMMAND

```
pg {command_line} {-control_arg}
```

FUNCTION

executes a specified command line and prints information about how its execution is progressing in terms of CPU time, real time, and page faults.

ARGUMENTS

command_line
is any string that is executable as a command line. If given, no control arguments to progress can appear on the same line except for -brief. *

CONTROL ARGUMENTS
you can supply only one control argument. *

-brief command_line, -bf command_line
prints only the message at completion of the specified command_line.

-cput N
prints incremental messages every N seconds of virtual CPU time. (Default: 10)

-off
suppresses the incremental messages printed during execution of a command line previously initiated, but does not suppress the message printed when that command line is finished (see "Notes on Output Messages" below). You can use -off to suppress messages while debugging.

-on
restores the printing of incremental messages during execution of the command line.

-output_switch name, -os name
directs output from the progress command to be printed on the I/O switch named name. (Default: user_i/o)

-realt N
prints incremental messages every N seconds of real time instead of virtual CPU time.

NOTES ON OUTPUT MESSAGES

After every 10 seconds of virtual CPU time (assuming the default triggering value is used), progress prints out a message of the form:

```
ct/rt = pt%, ci/ri = pi% (pfi)
```

where:

ct
is the number of virtual CPU seconds used by the command line so far.

rt
is the total real seconds used so far.

pt
is the ratio of virtual to real time used by the command so far.

ci
is the incremental virtual CPU time (since the last message).

ri
is the incremental real time.

pi
is ci expressed as a percentage of ri.

pfi
is the number of page faults per second of virtual CPU time (since the last message).

When the command line finishes, progress prints the following message:

```
finished: ct/rt = pt% (pft)
```

where:

pft
is the number of page faults per second of virtual CPU time for the execution of the entire command.

EXAMPLES

In the following example, you want to see how execution is progressing for the compilation of a PL/I source program (named newseg.pl1) using -list to the pl1 command.

```
! progress pl1 newseg -list
  PL/I
  10/30 = 33%, 10/30 = 33% (26)
  20/50 = 40%, 10/20 = 50% (17)
  30/123 = 24%, 10/73 = 13% (20)
  finished: 33/150 = 22% (22)
```

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. This description summarizes the editing requests and addressing features provided by qedx. Complete tutorial information on qedx is available in the *qedx Text Editor User's Guide* (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. The editor executes the qedx requests contained in the segment given and then waits for you to type further requests. If macro_path is omitted, the editor waits for you to type a qedx request. The archive component pathname convention (::) is accepted.

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 segment selected 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

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

-pathname path, -pn path
causes qedx to read the segment given 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 given, you are placed immediately in the editor request loop. The archive component pathname convention (::) is accepted.

NOTES

Once the qedx editor is invoked, you 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 you 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 you 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 indicating 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/`). The current line is denoted by a period (.); the last line of the buffer, by a 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 designate 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 (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, terminates the input request, and returns you to edit mode. It is used constantly when editing a document, and is the key to understanding the difference between the input and edit modes.

\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 your terminal change the value of the current line (.). Usually, the value of "." is set to the last address used (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 you the ability to execute commands in the Multics system without leaving the editor and also to effect global changes. You should learn the input and basic edit requests before the extended 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 you 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 designated line.

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

Default: `.a`

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

change (c)
replaces the indicated 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 designated 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)
exits the editor but first checks for modified buffers. If any modified buffers are present, qedx displays their status and asks for permission to exit. If permission is granted, all changes made to those buffers since they were last written are lost.

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`

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

quit force (qf) (Q)
exits the editor without checking for modified buffers. If any modified buffers are present, all changes made to those buffers since they were last written are lost.

Format: `qf or Q`

read (r)
appends the contents of the segment named "path" after the given line. The archive component pathname convention (::) is accepted. If path is omitted, a default pathname is used if possible. (See "Notes on Default Pathnames" below.)

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 supplied 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 the specified lines of the buffer into the segment named "path". The archive component pathname convention (::) is not accepted. If path is omitted, a default pathname is used if possible; however, if the default pathname identifies an archive component, an error message is printed. (See "Notes on Default Pathnames" below.)

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.

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 you wish to abort a command line invoked with the execute request by issuing the QUIT signal, program_interrupt 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 designated 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 line(s) from the current buffer to a chosen 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 you have created the additional buffers alpha and beta and have 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 your 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 DEFAULT PATHNAMES

The qedx editor maintains a default pathname for each buffer. This default pathname is used whenever a read (r) or write (w) request is given without a pathname.

Initially, the default pathname for a buffer is null; i.e., any attempt to read or write without a pathname results in an error message. Whenever a read request is issued with a pathname and the buffer is empty, qedx saves that pathname as the default for the buffer. Whenever a write request is issued with a pathname that writes the entire contents of the buffer (i.e., no address range is given), qedx saves that pathname as the default for the buffer.

If a read request is given when the buffer is not empty or a write request is given that does not write the entire buffer, qedx considers the default pathname of that buffer to be no longer trustworthy. The next use of the read or write request without a pathname in that buffer causes qedx to ask for permission to use the default pathname. If permission is given, qedx once again considers the pathname to be dependable.

For example, given the following sequence:

```
qedx
r first
r second
w
```

qedx asks for permission to write the buffer to the segment named "first" because the "r second" request was issued when the buffer was not empty.

On the other hand, given the following sequence:

```
qedx
r first
<editing requests>
1,$d
r second
<editing requests>
w
```

qedx writes the buffer to the segment named "second" without asking permission because the buffer was empty when the "r second" request was given.

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 program_interrupt. 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 interrupt an invocation of qedx (e.g., via use of the quit signal) and invoke qedx again before using the start, program_interrupt, or release commands, qedx informs you that you have one or more suspended invocations and asks if you wish to continue. If you answer "?" to this query, qedx prints an explanation of the implications of answering "yes" to this query along with our recommendation of the proper response to this situation.

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 your terminal. For convenience, the qedx editor description assumes that your 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 you a question and prints or returns the value "true" if your answer is "yes" or "false" if your answer is "no"; if you type anything else, the active function asks for a "yes" or "no" answer.

ARGUMENTS

arg
is the question to be asked. Enclose the question in quotes if it contains spaces or other command language characters.

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.

-input_switch STR, -isw STR
specifies the I/O switch to use for input of your response. (Default: user_input)

-long, -lg
adds a leading newline and three trailing spaces to the question. (Default)

-output_switch STR, -osw STR
specifies the I/O switch to use for output of the question to you. (Default: user_output)

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

NOTES

You can use the format_line active function 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 can define the default to be either "enable" or "disable". (See the command_query_ subroutine.)

EXAMPLES

The following lines from an exec_com segment allow you 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: quotient

SYNTAX AS A COMMAND

```
quotient numA numB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[quotient numA numB]
```

FUNCTION

returns the result of numA divided by numB.

NOTES

See the description of divide, which returns only the integer portion of quotient, in this manual.

EXAMPLES

The following interaction illustrates the quotient active function.

```
!  string [quotient 5 4]
   1.25
!  string [quotient 1 3]
   0.33333333333333333333333333333333333333333333333333333333333333
!  string [quotient 5 2]
   2.5
```

Name: rank

SYNTAX AS A COMMAND

```
rank CHAR {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[rank CHAR {-control_arg}]
```

FUNCTION

Prints or returns the position of a character in the Multics ASCII collating sequence. The null (NUL) character is at rank 0.

ARGUMENTS

CHAR
is a single character whose rank is to be returned. If it is a special character to the command processor (i.e., space, tab, semicolon, etc.), enclose it in quotes.

CONTROL ARGUMENTS

-decimal, -dec
returns the rank as a decimal number. (Default)

-octal, -oc
returns the rank as an octal number.

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.

ARGUMENTS

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

LIST OF MBX_SPECIFICATIONS

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

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

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

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

-user STR
specifies either a user's default mailbox or an entry in the system mail table (see "Notes on Mailbox Selection by User" below).

STR
is any noncontrol argument and is first interpreted as -mailbox STR; if no mailbox is found, STR is then interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

CONTROL ARGUMENTS

-abbrev, -ab
enables abbreviation expansion of request lines.

-accessible, -acc
selects only those messages in the mailbox that you are permitted to read. If you have read (r) extended access on the mailbox, read_mail selects all messages in the mailbox; if you have own (o) extended access on the mailbox, read_mail selects only those messages that you sent to the mailbox. (Default)

-acknowledge, -ack
acknowledges messages which request acknowledgement. (Default)

-all, -a
selects all messages in the mailbox regardless of who sent them. It requires read (r) extended access on the mailbox.

-brief, -bf
shortens some informative messages and suppresses others.

-count, -ct
prints the number of messages being read before entering the request loop. (Default)

-debug, -db
enables read_mail's debugging facilities. Use of this control argument is not recommended for normal users of read_mail.

-interactive_messages, -im
includes interactive messages in the mailbox in addition to ordinary mail as the messages that you can manipulate.

-list, -ls
lists the messages in the mailbox before entering the request loop.

-long, -lg
prints the long form of all informative messages. (Default)

-mail, -ml
includes only ordinary messages in the mailbox. (Default)

-no_abbrev, -nab
does not enable abbreviation expansion of request lines. (Default)

-no_acknowledge, -nack
does not acknowledge messages which request acknowledgement.

-no_count, -nct
does not print the message count.

-no_debug, -ndb
disables read_mail's debugging facilities. (Default)

-no_interactive_messages, -nim
includes only mail in the mailbox. (Default)

-no_list, -nls
does not list the messages before entering request loop. (Default)

-no_mail, -nml
does not include ordinary messages. It is incompatible with -nim.

-no_print, -npr
does not the print messages before entering request loop. (Default)

-no_prompt
suppresses the prompt for request lines in the request loop.

-no_request_loop, -nrql
does not enter the request loop if there are no messages in the mailbox. (Default

-not_own
selects only those messages in the mailbox that were not sent by you. It requires read (r) extended access on the mailbox.

-own
selects only those messages in the mailbox that you sent to the mailbox. This control argument requires own (o) extended access on the mailbox.

-print, -pr
prints all messages in the mailbox before entering the request loop.

-profile path, -pf path
specifies the pathname of the profile to use for abbreviation expansion. The suffix "profile" is added if necessary. This control argument implies -abbrev.

-prompt STR
sets the request loop prompt to STR. The default is

```
^/read_mail^[ (^d)^]:^2x
```

-quit
exits read_mail after performing any operations given by the -list, -print, or -request control arguments; this control argument must be given in combination with one of those.

-request STR, -rq STR
executes STR as a read_mail request line before entering the request loop.

-request_loop, -rql
enters the read_mail request loop even if there are no messages in the mailbox.

-totals, -tt
prints the number of the messages in the mailbox and exits without entering the request loop. This control argument is incompatible with -print, -list, -request, and -quit.

NOTES ON MAILBOX SELECTION BY USER

A user's default mailbox is specified in the form Person_id.Project_id. For an entry in the mail table, STR is usually in the form of Person_id. The mail table permits you to address mail by Person_id without knowing the Project_id of the recipient. The mail table is described in the The mail table is described in the *Extended Mail System User's Guide* (CH23) and the *Multics System Administration Procedures* (AK50) manuals.

If STR contains one period and no white space, it is interpreted as a User_id that specifies a user's default mailbox; otherwise it is interpreted as the name of an entry in the mail table.

For example:

```
-user DBuxtehude.SiteSA
```

is interpreted as a User_id that identifies a default mailbox. On the other hand,

```
-user "George G. Byron"
-user L.v.Beethoven
-user Burns
```

are all interpreted as the names of entries in the mail table: the first because it contains white space; the second because it contains more than one period; the third because it contains no period.

When interpreted as a User_id, the STR cannot contain any angle brackets (<>) and must have the form Person_id.Project_id, where "Person_id" cannot exceed 28 characters in length and "Project_id" is limited to 32 characters. In this case, "-user STR" is equivalent to the mbx_specification "-mailbox >udd>Project_id>Person_id>Person_id.mbx."

When interpreted as the name of a mail table entry, STR cannot contain any commas, colons, semicolons, backslashes (\), parentheses, angle brackets, braces ({}), quotes, commercial at-signs (@), or white space other than spaces. The query of the mail table is performed in a case-insensitive manner. The display_mailing_address command can be used to determine the actual address corresponding to the STR. The address in the mail table must identify a mailbox.

NOTES ON CONTROL ARGUMENTS AFFECTING INDIVIDUAL REQUESTS

Control arguments can be specified on the read_mail command line to change the default beavior of individual requests. Use of these control arguments on the command line is identical to specifying them for each use of the particular request. Of course, the modified default behavior of a request can be overriden for individual uses of the request by use of the appropriate control argument. Type:

```
! help request_name
```

within read_mail for more detail on the effect of the following control arguments.

LIST OF CONTROL ARGUMENTS AFFECTING THE FORWARD REQUEST

-acknowledge, -ack
: specifies that each recipient of your forwarded message will send an acknowledgement to you after they have read the message.

-no_acknowledge, -nack
: specifies that you do not want to receive acknowledgements. (Default)

-add_comments
adds a comment to a message before forwarding the message. You are prompted for the text of the comment, which can be edited in the same way that the message created by send_mail is edited before transmission. The text of the comment is placed in the new "Redistributed-Comment" field that is added by this request in addition to the standard redistribution header fields. The original copy of the message is not modified by this operation.

-log
places a copy of the forwarded message in your logbox. If the logbox does not exist, it is created and a message to that effect is displayed.

-notify, -nt
sends a "You have mail." notification to each recipient of the message. (Default)

-no_notify, -nnt
does not send notification messages.

-save path, -sv path
places a copy of the forwarded message into the savebox with the specified pathname. The suffix "sv.mbx" is added if necessary. If the savebox does not exist, you are queried for permission to create the savebox. If you refuse to give permission, the forward request is aborted without actually sending the message to any of the recipients.

LIST OF CONTROL ARGUMENTS AFFECTING THE PRINT REQUEST

-header, -he
displays all information from the message header including user-defined fields while excluding the message trace and redundant information. (Default)

-brief_header, -bfhe
displays the minimal amount of information from the message header. The date and authors are always displayed; the subject is displayed if it is not blank; the number of recipients is displayed either if there is more than one recipient or if you are not the sole recipient of the message; if the message was forwarded with comments, these comments are also displayed.

-long_header, -lghe
displays all information from the message header including network-tracing information even if some of the information is redundant (i.e., if the "From:", "Sender:", and "Delivery-By:" fields are all equal, this option forces the print request to display all three fields when it prints the message).

-no_header, -nhe
displays no information from the message header. Only the message number, message body line count, and message body are displayed.

LIST OF CONTROL ARGUMENTS AFFECTING THE PRINT_HEADER REQUEST

-default
displays all information from the message header including user-defined fields while excluding the message trace and redundant information. (Default)

-brief, -bf
displays the minimal amount of information from the message header (see -brief_header above).

-long, -lg
displays all information from the message header (see -long_header above).

LIST OF CONTROL ARGUMENTS AFFECTING THE REPLY REQUEST

-include_authors, -iat
includes the author(s) of the message as recipients of the reply. (Default)

-no_include_authors, -niat
does not include the author(s) of the message as recipients of the reply.

-include_recipients, -irc
includes the recipients of the message as recipients of the reply.

-no_include_recipients, -nirc
does not include the recipients of the message as recipients of the reply. (Default)

-include_self, -is
allows you to be a recipient of the reply without explicit use of -to or -cc.

-no_include_self, -nis
does not include you as a recipient of the reply unless explicitly requested via -to or -cc. (Default)

-include_original, -io
includes the original message as part of the text of the reply.

-no_include_original, -nio
does not put the original message into the reply's text. (Default)

-indent N, -ind N
indents the original message when -include_original is specified. (Default: 4)

-fill, -fi
causes the reply message to be filled before transmission. (Default for -terminal_input)

-no_fill, -nfi
causes the reply message to not be filled before transmission. (Default for -input_file)

-line_length N, -ll N
specifies the line length used when filling the reply message. (Default: 72)

-notify, -nt
sends a "You have mail." notification to each recipient of the message. (Default)

-no_notify, -nnt
does not send notification messages.

NOTES

Messages are not actually deleted until read_mail is exited via the quit request. While within read_mail, messages which are accidently marked for deletion can be restored by using the retrieve request.

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.

`abbrev {-ca}, ab {-ca}`
controls abbreviation processing of request lines.

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

`answer STR -ca request_line`
provides preset answers to questions asked by another request.

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

`apply {specs} {-ca} cmd_line, ap {specs} {-ca} cmd_line`
executes a Multics command line on the ASCII form of the messages.

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

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

`debug_mode {-ca}`
enables/disables read_mail's debugging facilities.

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

`do rq_str {args}, [do rq_str args]`
executes/returns a request line with argument substitution.

`exec_com ec_path {ec_args},`
`ec ec_path {ec_args},`
`[exec_com ec_path {ec_args}],`
`[ec ec_path {ec_args}]`
executes a file of read_mail requests which can return a value.

`execute cmd_line,`
`e cmd_line,`
`[execute active_str],`
`[e active_str]`
executes a Multics command line/evaluates a Multics active string.

`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.

if expr -then line1 {-else line2},
[if expr -then STR1 {-else STR2}]
conditionally executes/returns one of two request lines.

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

last_seen, [last_seen]
prints or returns the message number of the last nondeleted message that has been printed with the print request by a user having d extended access to the mailbox.

last_unseen, [last_unseen], lu, [lu]
prints or returns the message number of the last nondeleted message that has not been printed with the print request by a user having d extended access to 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 your logbox.

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

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

new, [new]
prints or returns the message number of all nondeleted messages since the last one that have been printed with the print request by a user having d extended access to the mailbox.

next_seen {specs}, [next_seen {specs}], ns {specs}, [ns {specs}]
prints or returns the message number of the first nondeleted message after the specified message (or after the current message by default) that has been printed with the print request by a user having d extended access to the mailbox.

next_unseen {specs}, [next_unseen {specs}],
nu {specs}, [nu {specs}]
prints or returns the message number of the first nondeleted message after the specified message (or after the current message by default) that has not been printed with the print request by a user having d extended access to the mailbox.

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 or returns the message number of the last message of the specified type before the current message.

previous_seen {specs}, [previous_seen {specs}],
ps {specs}, [ps {specs}]
prints or returns the message number of the last nondeleted message before the specified message (or before the current message by default) that has been printed with the print request by a user having d extended access to the mailbox.

previous_unseen {specs}, [previous_unseen {specs}],
pu {specs}, [pu {specs}]
prints or returns the message number of the last nondeleted message before the specified message (or before the current message by default) that has not been printed with the print request by a user having d extended access to the mailbox.

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.

seen, [seen]
    prints or returns the message number of all nondeleted messages that have been printed with the print request by a user having d extended access to the mailbox.

subsystem_name, [subsystem_name]
    prints or returns the name of this subsystem

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

switch_off switch_name {specs}, swf switch_name {specs}
    turns off a specified switch for each selected message.

switch_on switch_name {specs}, swn switch_name {specs}
    turns on a specified switch for each selected message.

unseen, [unseen]
    prints or returns the message number of all nondeleted messages that have not been printed with the print request by a user having d extended access to the mailbox.

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

Name: read_tape_and_query, rtq

SYNTAX AS A COMMAND

```
read_tape_and_query volume_id {-control_args},
                           rtq volume_id {-control_args}
```

FUNCTION

allows the user to interactively inspect and determine the contents of a magnetic tape. Physical tape file processing capabilities are also provided. Note that once the command is invoked, the user is placed in the read_tape_and_query subsystem where the user may use the read_tape_and_query requests. The read_tape_and_query requests are listed below under "List of requests".

ARGUMENTS

volume_id
is the local tape library designation of the requested tape volume.

CONTROL ARGUMENTS

-abbrev, -ab
enables abbreviation processing within read_tape_and_query. If this argument is specified and the -profile control argument is not given, the user's default profile segment (>udd>Project_id>Person_id>Person_id.profile) is used.

-block N, -bk N
specifies the maximum physical record size to be processed, where N is the number of bytes. The default is 11200 bytes (2800 36-bit words).

-comment STR, -com STR
displays STR as a message on the operators console at the time that tape volume <volume_id>is mounted. If STR contains spaces, tabs or special characters, the entire STR must be enclosed in quotes.

-density N, -den N
specifies the initial density setting for tape attachment, where N is the number of bits per inch (bpi). The default is 800 bpi. Although the density is automatically determined (see "Notes" below), some tape subsystems may not have tape drives capable of handling the default density.

-no_abbrev, -nab
specifies that abbreviation processing is not to be done by the read_tape_and_query request processor. (Default)

-no_prompt
suppresses printing of the prompt character string ("Ha:>") for read_tape_and_query requests.

-no_request_loop, nrql
does not enter the read_tape_and_query request loop.

-profile PATH, -pf PATH
specifies that abbreviation processing is to be done using PATH. The suffix ".profile" need not be given; however, ".profile" must be the last component of PATH. If this control argument is given, the "-abbrev" control argument need not be given.

-prompt STR
changes the prompt for the read_tape_and_query request loop to STR. If STR is a null string, "", no prompt is given. (Default is to prompt with "rtq:").

-quit
exits after performing any operations specified by control arguments. (Default is to enter the read_tape_and_query request loop).

-request STR, -rq STR
specifies that an initial request line of STR is to be executed before entering the read_tape_and_query request loop.

-request_loop, -rql
specifies that the read_tape_and_query request loop be entered. (Default).

-ring, -rg
specifies that the tape is to be mounted with a write ring. This allows a tape that is already mounted with a write ring to be attached without operator intervention. The default is to mount the tape with no write ring.

-track N, -tk N
where N is 7 or 9 for 7 or 9 track tapes. If this control argument is not specified, 9 track is assumed.

NOTES

The read_tape_and_query command requests the specified tape volume to be mounted. After the mount request has been satisfied, read_tape_and_query automatically determines the tape density and checks for a recorded tape label. If the density can be determined, an informative message is displayed that includes the density. If the tape has a standard Multics, GCOS, IBM, ANSI or CP5 tape label, an informative message is displayed that includes the standard label type and the recorded volume name. If the tape contains a valid IBM or ANSI label, a second message is displayed informing the user of the physical block size and logical record length (in bytes) of the first data file. For all standard labeled tape volumes, the tape is then positioned to the beginning of the first data file. If the tape label is not recognized as one of the five standard types mentioned above, it is designated as unlabeled and the tape volume is repositioned to the beginning of the tape.

The read_tape_and_query command then goes into a request loop whose requests are listed below.

LIST OF REQUESTS

The rtq request loop displays the prompt character string "rtq:" unless "-no_prompt" control argument has been specified. Some requests acceptable to read_tape_and_query take arguments that are optional. These optional arguments are enclosed in braces. The valid user responses while in this request loop are as follows:

?
lists the available read_tape_and_query requests and active requests.

abbrev {-off | -on | -profile PATH}, ab {-off | -profile PATH}
turns abbreviation processor on or off and changes profile segments. As an active request, [ab], returns "true" if abbreviation expansion of request lines is currently enabled within the subsystem and "false" otherwise.

.. <rest_of_line>
passes <rest_of_line> to the command processor for execution as a Multics command.

.
displays the command name read_tape_and_query with its short name (rtq) in parentheses.

answer STR {-brief, (bf) | -call STR | -match STR | -exclude STR, -ex STR | -query | -then STR | -times N} request_line
provides preset answers to questions asked by another request, where STR is the desired answer to any question and request_line is any subsystem request line.

bof
positions to the beginning of the current physical tape file.

bsf {N}
backspaces N files. If N is not specified, 1 is assumed.

bsr {N}
backspaces N records. If N is not specified, 1 is assumed. bsr will not cross backward to the previous file.

density N, den N
sets the tape density to N bits per inch (bpi), where N can be 6250, 1600, 800, 556, or 200. Density requests must be issued while the tape is positioned at the BOT marker or a request reject status results. Normally the tape density need not be set as it is automatically set by read_tape_and_query before the request loop is entered.

do request_string {args}
or: do -long, -lg | -brief, -bf | -nogo | -go | -absentee | -interactive
expands a request line by substituting the supplied arguments into the line before execution. The request_string is a request line in quotes and args are character string arguments that replace parameters in request_string. As an active request, [do "request_string" args] returns the expanded request_string rather than executing it.

dump {offset} {n_words} {char_types}
displays the contents of the record buffer (filled with the read_record request) on the users terminal. If no arguments are specified, the contents of the entire tape buffer are displayed in octal format.

If the n_words argument is specified, it must follow offset. However, these arguments may be positioned before or after any char_type arguments that may be specified. The offset and n_words arguments must be specified in octal. If offset is specified without being followed by n_words, then the tape buffer is dumped starting with the <offset>th word and ending with the last word in the tape buffer. The char_type optional arguments allow interpretation of the data contained in the tape buffer in various character formats. If more than one char_type argument is specified, then the tape buffer is dumped with the first character interpretation, followed by the next character interpretation, and so on until all requested data formats have been dumped.

The value of char_type can be selected from the following:

-ascii
displays the contents of the record buffer in octal with an ASCII interpretation of the data on the right side.

-bcd
displays the contents of the record buffer in octal with a BCD interpretation of the data on the right side

-ebcdic
displays the contents of the record buffer in octal with an EBCDIC interpretation of the data on the right side.

-hex
displays the record buffer in hexadecimal format.

exec_com ec_path {ec_args}, ec ec_path {ec_args}
executes a program written in the exec_com language which is used to pass request lines to the rtq subsystem and to pass input lines to read_tape_and_query requests which read input. ec_path is the pathname of an exec_com program. The suffix ".rtq" is assumed if not specified. ec_args are optional and are substituted for parameter references in the program such as &1. As an active request, [ec ec_path {ec_args}], the exec_com program specifies a return value of the exec_com request by use of the &return statement.

execute LINE, e LINE
executes the supplied line as a Multics command line. As an active request, [e LINE], evaluates a Multics active string and returns the result to the subsystem request processor. LINE is the Multics command line to be executed or the Multics active string to be evaluated. It need not be enclosed in quotes.

execute_string {-control_args} {control_string {args}}
[exs {-control_args} control_string {args}
substitutes arguments into a control string. The expanded control string is then passed to the command processor or the rtq subsystem request processor for execution, where control_string is a character string which may contain substitution constructs and args are zero or more character string arguments. Any argument supplied but not referenced by an argument substitution designator is ignored. As an active function,

eof
positions to the end of the current physical tape file, after the last record.

fsf {N}
forward spaces N files. If N is not specified, 1 is assumed.

fsr {N}
forward spaces N records. If N is not specified, 1 is assumed.

help {topics} {-ca}
prints information about request names or topics, where topics are the topics on which information is to be printed.

if EXPR -then LINE1 {-else LINE2}
conditionally executes one of two request lines depending on the value of an active string. EXPR is the active string which must evaluate to either "true" or "false". LINE1 is the subsystem request line to execute if EXPR evaluates to "true" and LINE2 is the subsystem request line to execute if EXPR evaluates to "false". If LINE2 is omitted and EXPR is "false", no additional request line is executed. As an active request, returns one of two character strings to the subsystem request processor depending on the value of an active string.

list_help {topics}, lh {topics}
displays the names of all subsystem info segments pertaining to a given set of topics.

list_requests {STRs} {-all, -a | -exact},
lr {STRs} {-all, -a | -exact}
prints a brief description of selected subsystem requests, where STRs specifies the requests to be listed.

list_tape_contents {-long} {-label}, ltc {-lg} {-lbl}
displays information about each record on the tape. The tape is positioned to BOT and each record is read in. If the tape is one of the five known standard types, the current record is inspected to determine if it is a valid label or trailer record; if so, information pertinent to that particular label or trailer record is displayed, in interpreted format. If the -long argument is used, the contents of the label record is displayed (in ASCII) as well. Otherwise, the length of the current record is compared to the length of the last record read. If the lengths are the same, a tally of the number of records with the same length is incremented. If the length of the current record is different from that of the last record, or if an end of file mark is detected, a message is displayed that includes: the number of records of equal length, and the record length in bits, words, 8-bit bytes, 9-bit bytes, and 6-bit characters.

This display of record lengths can be circumvented by using the -label argument, which only displays the label records. This operation continues until the logical end of tape is reached (two end of file marks in succession or an end of volume trailer record, followed by an end of file mark). The tape is repositioned to BOT after the list_tape_contents request is complete. Use of the -label argument with unlabeled tapes is treated as an error.

mode STR
sets the hardware mode for reading tape to STR, which can be one of the following modes:

bin
eight bit bytes are read in and packed (nine eight bit bytes per memory double word). This is the default mode.

bcd
reads in tape that was originally written in binary coded decimal (BCD). The hardware performs input character conversion.

nine
eight bit bytes are read in and converted to nine bit bytes by forcing the most significant bit of each nine bit byte to "0"b.

position, pos
displays the current physical tape file and record position for the user.

quit, q
detaches the tape and returns control to the current command processor.

read_file {args}, rdfile {args}
reads the current tape file into the segment described by args. The default action of this request with no arguments queries the user as to the segment name he wishes the tape file to be read into and then issues a warning telling the user that the current tape file will be read in as a stream file with no conversion. The user is asked if he wishes to continue. If he answers yes, then the tape file is read into the designated segment and a newline character is appended to each physical record. If the user answers no, then control is returned to the

read_tape_and_query

request loop. If the tape is one of the five standard types, each record is checked to determine if it is a valid label or trailer record. If it is, pertinent information about the record is displayed and the record is not written to the output segment. The optional arguments associated with the read_file request are:

-output_file {STR}, -of {STR}
where STR specifies the segment name for the tape file to be read into. If STR is omitted, the user is queried for the segment name.

-count N, -ct N
allows reading up to N files, or until logical end of tape is encountered. After the first file is read in, the -count iteration count is appended to the end of the user-designated output file name as a second component. For example:

```
rdfile -ct 3 -of file1
```

names the first output file file1, the second file1.2, and the third file1.3.

-multics, -mult
specifies that the input tape file is in Multics standard system format. The data portion of each unrepeated record is written to the specified stream output file. No attempt is made to separate the contents of the physical record into a logical format. Since standard Multics tape format specifies that an EOF mark be written every 128 records, the "-extend" and "-count" arguments should be used to ensure that all of the data is recovered.

-gcos, -gc
specifies that the input tape file is in GCOS standard system format. That is, each record has a block control word and several record control words dividing the physical record into logical records. Each record is processed accordingly. BCD records are converted to ASCII. ASCII records are copied directly. Binary compressed deck card images are decompressed and converted to ASCII. If a BCD card image is identified as a "$ object" card, this card image and all successive binary card images, until a "$ dkend" card image is identified, are copied to a separate file whose name is formed from columns 73 - 76 of the $ object card with a suffix of ".obj". If a BCD card image is identified as a "$ snumb" card, this card and all following card images, until another $ snumb card or end of file, are copied into a file whose name is formed from columns 16 - 21 of the $ snumb card with a suffix of ".imcv". If a BCD card image is identified as a "$ <language>" card, this card and all following card images, until another $ <language> card or end of file, are copied into a file whose name is formed from columns 73 - 76 of the $ <language> card with a suffix of ".ascii". This file is also surrounded by sufficient GCOS "JCL cards" so that the completed "deck" can be assembled using the Multics GCOS Environment Simulator. If columns 73 - 76 of the $ <language> card are blank, the $f <language> card image is displayed and the user is queried for the filename.

-cp5
specifies that the input tape file is in CP5 standard system format, which consists of variable length records, recorded in EBCDIC. Each variable length logical record is written to the specified stream file, with a newline character appended to the end. The data read from the tape is automatically converted from EBCDIC to ASCII.

-dec
specifies that the input tape file is in Digital Equipment Corporation (DEC) standard system format. Each DEC word is 40 bits long, of which the first

32 bits and the last four bits are concatenated to form one 36-bit word. The other four bits are discarded. The converted data is then written onto the specified file in raw format.

-ibm_vb {STR}
specifies that the input tape file has standard IBM VB-formatted variable-length records with embedded block and control words. STR can be ebcdic, ascii, or binary (bin). (Default: ebcdic)

-ansi_db {STR}
specifies that the input tape file has ANSI-standard DB-formatted variable-length records with embedded record control words. STR can be ascii, ebcdic, or binary (or bin). (Default: ascii)

-output_description, -ods
allows you to specify a standard Multics I/O attach description to receive the tape file data. User queries ask you to input the attach description and the opening mode. You can express opening modes in long form or in abbreviation form (e.g., sequential_output, sqo).

-extend
allows you to concatenate the contents of several tape files into one output file. This control argument has meaning only if you also specify -count.

-nnl
allows escape from the read_file default of appending a new line character to the end of each physical record, when you give no other format specification.

-truncate N, -tc N
allows you to truncate each physical record to a length of N characters.

-skip N
allows you to skip N characters (e.g., a record or block control word) at the beginning of the physical tape record. It is useful when you are processing tapes of an unfamiliar format.

-logical_record_length N, -lrl N
allows you to divide each physical tape record into several logical records of length N. Each logical record is written to the specified file with a new line character appended to the end. Logical records cannot span physical blocks.

-convert STR, -conv STR
allows you to convert the data format of each tape record, where STR can be one of the following:

ebcdic_to_ascii, ebcdic
converts input EBCDIC data to ASCII.

bcd_to_ascii, bcd
converts input BCD data to ASCII.

comp8_to_ascii, comp8
converts input comp8 (four-bit-packed decimal) data to its equivalent ASCII representation.

read_record {-count N}, rdrec {-ct N}
reads the current record into a temporary buffer. If the tape is one of the five known standard labeled tapes, the record is checked to determine if it is a label or trailer record; if it is, information pertinent to that particular record type is displayed. Otherwise, information pertaining to the physical record length in bits, words, 8-bit bytes, 9-bit bytes, and 6-bit characters is displayed. When the -count argument is specified, N records are read, overlaying each other in the temporary buffer.
Note that when read_record encounters a tape mark, it leaves you positioned at the beginning of the next file.

records_in_file, rif
displays the total number of records in the current physical tape file. This operation reads each of the records in the file, repositions the tape to its original position, prior to this operation, and displays the count of records read.

rewind, rew
issues a rewind command and positions the tape to the beginning of tape (BOT) marker.

substitute_arguments {-control_args} {control_string {args}}
[sbag control_string {args}]
substitutes arguments into a control string and prints the result on user_output. As an active function, the result is returned.

Tape Positioning: When inspecting multifile tape reels, you may find the action of various positioning requests confusing. The table below illustrates the starting and ending position when using various tape positioning requests:

Start Position	Operation	End Position
file 6, record 7	rewind	file 1, record 1
file 6, record 7	bof	file 6, record 1
file 6, record 7	bsf	file 5, record 1
file 6, record 7	fsf	file 7, record 1
file 6, record 7	bsr	file 6, record 6
file 6, record 7	fsr	file 6, record 8
file 6, record 7	bsf 8 (1)	file 1, record 1
file 6, record 7	bsr 10 (2)	file 6, record 1
file 6, record 1	read_file -count 3	file 9, record 1

Note 1: This causes a rewind operation to occur, since the resultant file number would be less than one.
Note 2: This causes a bof operation to occur, since the resultant record number would be less than one.

Examples:

A typical example of a read_tape_and_query invocation follows, including the initial information displayed for a labeled tape.

```
read_tape_and_query usert1
Tape usert1,blk=2800 will be mounted with no write ring.
Tape usert1,blk=2800 mounted on drive tape_02 with no write ring.
Tape density is 1600 bpi
Tape usert1 is a labeled ANSI tape
Volume name recorded on tape label is USERT1
Setting tape dim to read in nine mode
First data file format:
 ANSI HDR2 label record. Next file format:
 Record format DB; Block length 4000; Record length 4000; Mode ASCII;
Positioning to beginning of physical tape file # 2,
 (logical file # 1)
rtq:
```

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

types a ready message on your terminal after each command line has been processed.

NOTES

Since automatic printing of the ready message is in effect until you invoke ready_off, ready_on is generally used only to "cancel" ready_off.

See the ready, ready_off, and general_ready commands.

Name: rebuild_dir

SYNTAX AS A COMMAND

```
rebuild_dir path {-control_arg}
```

FUNCTION

compares a saved directory information segment created by the save_dir_info command with the current version of the directory in the storage system. If any subdirectories are missing, rebuild_dir recreates them; if any links are missing, it relinks them; if any segments are missing, it prints a comment.

ARGUMENTS

path
: is the pathname of a directory information segment. If path does not have the dir_info suffix, it is assumed.

CONTROL ARGUMENTS

-brief, -bf
: suppresses the comments "creating directory X" and "appending link X".

-long, -lg
: prints full information about any missing segments.

-priv
: sets quotas and the sons' logical volume identifier. You need access to the hphcs_ gate here.

Name: reconnect_ec_disable

SYNTAX AS A COMMAND

```
reconnect_ec_disable
```

FUNCTION

reverses the effect of the reconnect_ec_enable command. Following reconnection to a disconnected process, no attempt is made to find or invoke the exec_com "reconnect.ec". If a standard process overseer (e.g., process_overseer_ or project_start_up) is in use (the normal case), reconnect_ec_enable is in effect by default.

Name: reconnect_ec_enable

SYNTAX AS A COMMAND

```
reconnect_ec_enable
```

FUNCTION

reverses the effect of the reconnect_ec_disable command. When the user reconnects to a disconnected process, an attempt is made to find the segment reconnect.ec. First the user's home directory, then the user's project directory (`>user_dir_dir>Project_name`), then `>system_control_dir` will be searched. At the first success, the command `"exec_com >Directory_name>reconnect.ec"` will be executed.

NOTES

The use of reconnect.ec is enabled automatically by the standard process overseer process_overseer_.

The current command processor is used to execute the command. Thus, if the user is using the abbrev command processor, any applicable abbreviation will be expanded.

Invocation of the reconnect.ec is not automatically enabled by the project_start_up_ process overseer. Thus, when using project_start_up_, the project administrator may enable the invocation of reconnect.ec at any point in the project_start_up.ec.

Name: reductions, rdc

SYNTAX AS A COMMAND

```
rdc path {-control_args}
```

FUNCTION

generates language translators. It compiles a segment containing reductions and action routines into a PL/I source segment; it then invokes the pl1 compiler to compile the PL/I source. The reductions specify the syntax and semantics of a new language; the action routines perform the basic operations (e.g, generating code) required to translate
* the language.

ARGUMENTS

path
: is the pathname of a translator source segment that is to be compiled by rdc. If path does not have a suffix of rd, one is assumed; however the rd suffix must be the last component of the name of the source segment.

CONTROL ARGUMENTS

-brief, -bf
: prints all error messages with only a brief summary of the error that has occurred.

-long, -lg
: prints all error messages with a detailed description of the error that has occurred.

| -trace {STR}
: adds a tracing facility to the generated translator. STR defines whether tracing is enabled or disabled by default. It can be "on" (default) or "off." (See "Notes on Tracing" below.)

| -no_trace
: generates a translator without the tracing facility. (Default)

In addition, you can give any control argument accepted by the pl1 command.

NOTES

Reductions are expressed in a highly compact form that emphasizes the syntax and semantics of the new language. This command compiles these reductions into tables that drive an rdc-provided semantic analyzer for the language. This analyzer compares the tokens (basic units) of a program written in the new language with the valid token phrases defined in the reductions. When a valid phrase is found, the action routines defined by the reduction are invoked to translate the phrase.

Translators generated by the command can be written more quickly than hand-programmed translators because rdc provides the semantic analyzer for the language. They are easier to understand and to maintain because the all-important language syntax and semantics is concentrated in the reductions, rather than being spread throughout the semantic analyzer.

This command can generate translators for the simplest type of language, a right-linear (finite state automaton) language. Often such languages are composed of keywords with operands, such as the control language of the bind command.

The organization of an rdc translator, the translation process, and the reduction language are described below.

If you supply neither -brief nor -long, a detailed description is printed the first time an error occurs in a given compilation and a brief description is printed in subsequent occurrences of that error.

NOTES ON SEVERITY VALUES

The following severity values are returned by the severity active function when the "reductions" keyword is used:

VALUE	MEANING
0	No compilation yet, or no error.
2	Correctable error. This error has been bypassed, but may result in other, more severe errors.
3	Fatal error, but rdc translation continues to report other errors.
4	Unrecoverable error. Translation stops.

In addition, the severity value can be any of those associated with compilation of the resultant PL/I source segment by the pl1 command.

NOTES ON TRACING

The tracing facility helps in debugging the reductions for a new translator by showing which reductions are being applied, along with the source tokens of the language being compiled that match each reduction. This gives you a running view of the flow of control through the reductions and of the processing of tokens.

Because the tracing facility generates large amounts of output, it can be selectively turned on and off during debugging by setting a variable declared as follows:

```
dcl TRACING bit(1) aligned int static init("1"b);
```

A value of "1"b turns tracing on; "0"b turns it off. The initial value is set by the operand of -trace.

The translator can accept a control argument to turn TRACING on or off, or it can have a debugging entry point to set the switch, or the switch can be set at a particular probe breakpoint. For example, to trace from the 28th through the 40th reductions, you can use the following set of probe requests:

```
! probe translator
! ps "RD_ACTION(NRED)"
  go to RD_ACTION(NRED);
! b: if NRED = 28: halt

When halted, type:

! let TRACING = "1"b
! reset
! b: if NRED = 40: h
! c

And when the 40th reduction was reached, type:

! let TRACING = "0"b
! reset
! c
```

OVERVIEW OF THE TRANSLATION PROCESS

A translator for a given language must perform three steps to translate a source string written in the language:

1. Parse the source string into a list of tokens. These tokens are the basic units of the language being translated. They are character strings in the source that are separated from one another by language-defined delimiter characters.

2. Analyze the syntax of the tokens to identify groups of tokens (token phrases) that are valid or invalid in the language.

3. Assign some semantic meaning to each valid token phrase by performing a translation action. Print error messages diagnosing invalid token phrases.

Parsing the source string into tokens is a process that depends upon the types of units that make up the language, the delimiter characters defined by the language, and other language characteristics. For most languages, the lex_string_ subroutine provides facilities that are sufficient to parse the language. However, some languages have syntax characteristics that cannot be handled by lex_string_; your must code a special parsing routine for such languages (see "Parsing the Source into Tokens" below). See lex_string_ in the Subroutines manual for more information about its parsing capabilities.

Once the source string is parsed into a list of tokens, these tokens are analyzed by a semantic analysis procedure that is generated by the reductions command from the reductions specified in the translator source segment. The procedure contains tables generated from the reductions, tables that define all of the valid token phrases accepted by the language.

For each valid token phrase, the semantic analyzer invokes the programmer-supplied action routines given in the reductions to translate the phrase. For invalid phrases, the reductions command provides a mechanism for diagnosing the errors in printed error messages.

CONTENTS OF A TRANSLATOR

The source segment for a translator to be compiled by the reductions command contains the following items, which are organized as shown in Figure 3-1.

1. An optional copyright notice and other PL/I comments.
2. A set of reductions consisting of reduction attribute declarations and reduction statements. The delimiters /*++ and ++*/ open and close the set of reductions.
3. A PL/I procedure statement for the main procedure of the translator.
4. PL/I declarations for the translator's variables.
5. An optional PL/I declaration for an error_control_table, which defines the text of error messages to be generated by the translator. This error_control_table is used by the error-reporting mechanism provided by the reductions command.
6. Code to parse source strings of the new language into tokens. This is shown in Figure 3-1 as a call to lex_string_, but programmer-supplied code could be used here instead.
7. A PL/I call statement invoking SEMANTIC_ANALYSIS, the semantic analyzer procedure generated by the reductions command from the reductions.
8. A PL/I return statement to return after the translation process is complete.
9. One or more optional PL/I function subprograms (relative syntax functions) that are used to define the syntax of valid token phrases.
10. One or more PL/I subroutines (action routines) that are invoked to translate valid token phrases.

```
/* ********************
   * c Copyright ...  *
   ******************** */

/*++
     MAX_DEPTH 5 \
     BEGIN
       /   /   /        \
       /   /   / RETURN \
                        ++*/

translator: procedure;

     dcl .... ,
         .... ,
         .... ;

     dcl error_control_table...;


     call lex_string_$lex(..);
     Pthis_token = ...;
     call SEMANTIC_ANALYSIS();
     return;


     fcn: procedure returns
       (bit(1) aligned);
     end fcn;


     action: proc(...);
     ...
     end action;
```

copyright notice

reduction statements and attribute declarations

translator's declarations

calls to parse translator input into tokens, translate these tokens, & return

relative syntax functions

action subroutines

Figure 3-1. Organization of a Translator

The definition of the reductions, the error-reporting mechanism, the parsing routine, relative syntax functions, and action routines are described further below.

No PL/I end statement is included for the main procedure of the translator in the translator source segment. The reductions command appends code for the SEMANTIC_ANALYSIS subroutine and for other utility programs to the contents of the translator source segment. It then appends the end statement for the translator's main procedure. Therefore, when coding the translator source segment, make sure that all the relative syntax functions and action routines are ended correctly, and that no end statement is included for the main procedure of the translator.

COMPILING THE TRANSLATOR

A typical translator source segment, translator.rd, is compiled by a two-step process as shown in Figure 3-2. First, the reductions command compiles translator.rd into a PL/I source segment, translator.pl1, which it creates in your working directory. Second, rdc invokes the pl1 command to compile the translator.pl1 into an object segment called translator. The translator object segment is placed in your working directory.

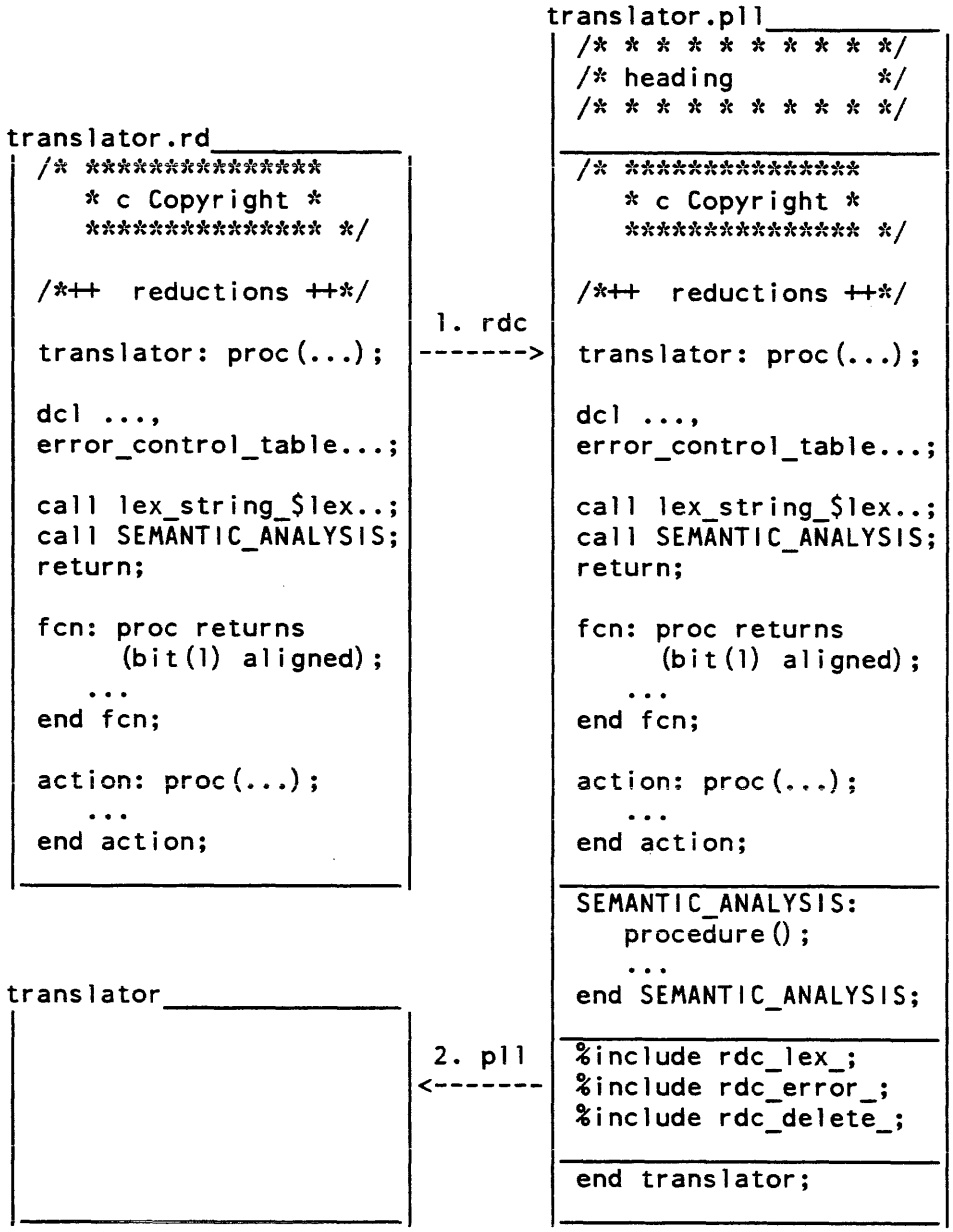


Figure 3-2. Two Steps of Compiling

The output of the reductions command is a PL/I source segment that contains

1. A heading that identifies the translator source segment, the version of the reductions command used to compile the translator source segment into the PL/I source segment, and the date and time of compilation.

2. The contents of the translator source segment.

3. The SEMANTIC_ANALYSIS procedure generated by the reductions command from the reductions in the translator source segment.

4. PL/I %include statements for utility procedures used in SEMANTIC_ANALYSIS and perhaps in the action routines to perform various functions.

5. A PL/I end statement for the main procedure of the translator. This is provided by the reductions command.

THE TRANSLATION PROCESS

The next few paragraphs describe the process of translating a language source string. It is important to understand how these steps are performed in rdc-generated translators. The reductions command or the lex_string_ subroutine provide code to perform many of these steps. For others, the programmer must supply a procedure to perform the steps.

PARSING THE SOURCE INTO TOKENS

As mentioned above, the first step of the translation process is for a translator to parse its input source string into a list of tokens. These tokens are the basic units of the language. For many languages, the lex_string_ subroutine provides sufficient facilities to parse the language. However, some languages may have a syntax that requires the special parsing facilities of a programmer-supplied parsing routine.

The lex_string_ subroutine or the supplied parsing routine must generate a chained list of token descriptors, as shown in Figure 3-3. Each descriptor describes one of the tokens in the source string. The token descriptors are chained together (forward and backward) in the order in which their respective tokens appear in the source string.

```
Volume: 70092;
Write;
File 4;
```

might be

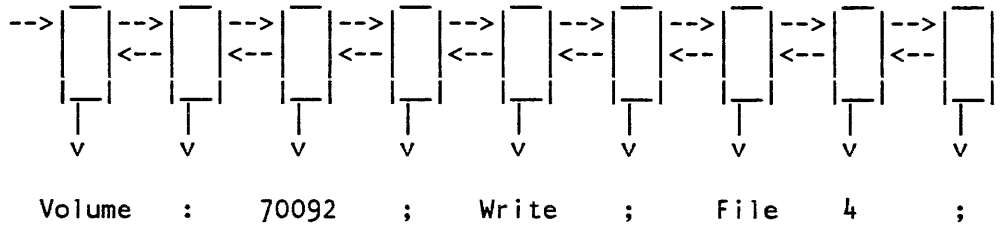


Figure 3-3. Input Tokens and Their Descriptors

Languages whose syntax includes statements separated by explicit statement delimiters can use a statement descriptor to identify the group of tokens forming each statement. The statement descriptor points to the descriptors for the first and last tokens in the statement. In turn, each token descriptor points to its respective statement descriptor. The statement descriptors are chained together (forward and backward) to create an ordered list of the statements appearing in the source string, as shown in Figure 3–4.

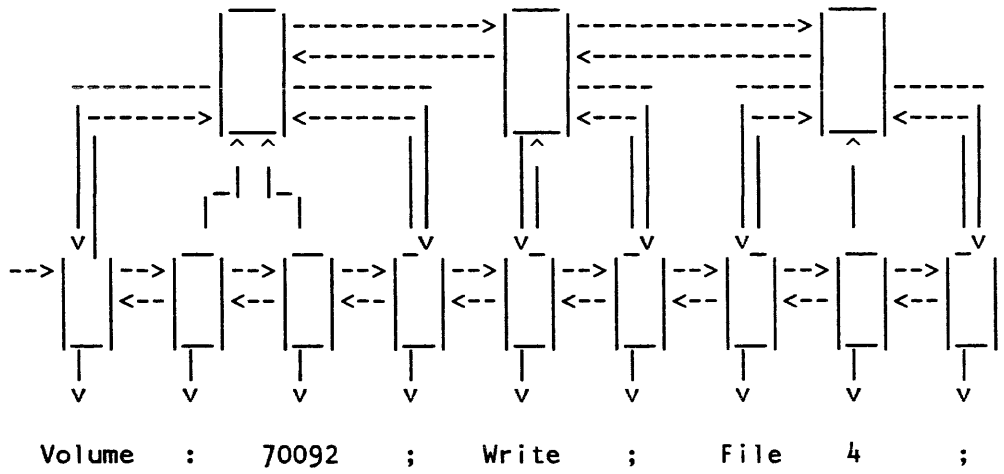


Figure 3–4. Tokens, Token Descriptors, and Statement Descriptors

There are no special requirements for a programmer–supplied parsing routine other than that it create a list of token descriptors and optional statement descriptors. The format of these descriptors is defined in the description for the lex_string_ subroutine. Refer to this description for more information about descriptors, as well as for information on the use of the lex_string_ subroutine. Figure 3–5 shows the lex_string_ subroutine being invoked first to initialize the lex_delims and lex_control_chars break definition strings, and then to parse the translator's source string (described by Pinput and Linput) into tokens. In this example: a double quote (") character is used to open and close quoted strings; the characters /* open comments, which are closed by */; a semicolon (;) is the statement delimiter; and the colon (:), comma (,), space ( ), and all of the ASCII control characters including the PAD character operate as delimiters. The space character and all control characters except backspace are ignored delimiters that are not returned as tokens themselves, even though they separate tokens. Both token descriptors and statement descriptors are generated by the lex_string_ subroutine in this example. No descriptors are generated for the double quotes that enclose quoted strings, although descriptors are generated for the quoted strings themselves.

```
breaks = substr(collate,1,33) || ":," || substr(collate,128,1);
ignored_breaks = substr(collate,1,8) || substr(collate,10,24) ||
     substr(collate,128,1);
call lex_string_$init_lex_delims("""""", """""", "/*", "*/", ";", "10"b,
     breaks, ignored_breaks, lex_delims, lex_control_chars);
call lex_string_$lex(Pinput, Linput, Linput_ignore, Psegment, "100"b, """""",
     """""", "/*", "*/", ";", breaks, ignored_breaks, lex_delims,
     lex_control_chars, Pfirst_stmt_descriptor, Pfirst_token_descriptor,
     code);
Pthis_token = Pfirst_token_descriptor;
call SEMANTIC_ANALYSIS();
return;
```

Figure 3-5. Parsing Translator Input Into Tokens, Semantically Analyzing Those Tokens, and Returning

ANALYZING AND TRANSLATING THE TOKENS

Once the source string has been parsed into tokens, the translation continues by analyzing the syntax of the source tokens. The syntax specifications of the language are used to identify groups of tokens (token phrases). Valid token phrases are translated according to the language semantics (translation action specifications), and invalid token phrases are diagnosed to you.

The language syntax and translation action specifications are coded in the set of reductions contained in the translator source segment. The reductions command uses these reductions to generate a SEMANTIC_ANALYSIS internal procedure that is appended to the translator when it is compiled.

When the SEMANTIC_ANALYSIS procedure is invoked as shown in Figure 3-5, it compares token phrases found in the list of source tokens with the syntax specifications defined in the reductions. If a token phrase matches the syntax specifications of a given reduction, the translation action routines associated with the reduction are invoked to translate the phrase. Then action routines are invoked to move on to the next token phrase, which is translated in a similar manner.

The translation is complete when each of the token phrases in the list of source tokens has been identified as a valid token phrase and translated, or has been diagnosed as an invalid token phrase.

REDUCTION LANGUAGE

The reductions that define the syntax and semantics of a language to be translated are written in the reduction language. This translator generation language consists of two kinds of statements: reduction statements and attribute declarations.

Reduction statements specify the syntax of token phrases in the language being translated. They also name action routines that are invoked to translate valid phrases and to diagnose invalid token phrases.

Attribute declarations control the size of some fixed-length tables that the generated translator uses and cause translation action routines provided by the reductions command to be included in the translator. They are described below under "Attribute Declarations."

THE SYNTAX OF REDUCTION STATEMENTS

All reduction statements contain four parts: a reduction label field, a syntax specification field, an action specification field, and a next-reduction field. A reduction statement has the form

labels / syntax specifications / action routines / next-reduction label \

All of the fields must appear in each reduction, in the order shown above. The fields are separated from one another by a right slant (/) character, and the next-reduction field is terminated by a left-slant (\) statement delimiter. The fields of a reduction statement can span any number of lines in the translator source segment.

The syntax specifications, action routines, and other items that appear in a reduction statement are separated from one another by one or more of the delimiters shown in Table 3-1 below. When these delimiter characters are used, they are treated as part of the reduction. The meaning of left and right slant was described above. The double quote (") character is used as a quoting character to delimit quoted character strings in the PL/I convention. When any of the delimiter characters appears in a quoted string, it is treated as a regular character rather than as a delimiter. The use of the other delimiters is described in more detail as each field of the reduction statement is described below.

Table 3-1. DELIMITING CHARACTERS USED BY rdc

/	separates fields of a reduction statement.
\	ends each reduction statement or attribute declaration.
< >	delimits a syntax function in the syntax field of a reduction. For example, `<no-token>` .
[ ]	delimits a PL/I statement in the action field of a reduction. For example, `[file_no = token.Nvalue]` .
;	separates PL/I statements in the action field of a reduction when more than one statement is given between [ ] delimiters. For example, `[a=b; c=d]` .
( )	delimits the argument list of a PL/I subroutine call in the action field of a reduction. For example, perform_io (volume, file_no) .

, separates arguments in the argument list of a PL/I subroutine call given in the action field of a reduction.

" begins and ends quoted strings within a reduction statement. Inside a quoted string, a double quote (") character is expressed by two double quotes ("").

- used to detect the special PL/I character sequence, -> , which can appear in an action specification.

= used to detect the special PL/I character sequences, ^= <= >= , which can appear in an action specification.

^ used to detect the special PL/I character sequences, ^= ^< ^> , which can appear in an action specification.

<BS> (backspace) used in the syntax field of a reduction to detect an underlined delimiter character. The special meaning of such a character is ignored, and the character is treated as a syntax specification character.

\" begins a comment in a reduction statement. The comment ends with the next newline character.

There are also five delimiters that delimit items in a reduction but are ignored by the reductions command unless enclosed in a quoted string. These characters have no meaning in the reduction language but serve mainly to separate the specifications in a reduction statement

space newline horizontal tab vertical tab newpage

SEMANTICS OF REDUCTION STATEMENTS

The most important part of any set of reductions are the syntax fields given in the reduction statements. These fields describe the syntax of the valid and invalid token phrases in the language to be translated. The syntax specifications can require a token in a particular phrase to have a specific character string value, or to have a value that meets some general list of requirements defined in a syntax function (a PL/I function subprogram).

When a token phrase does not match the syntax requirements of a reduction it is compared with, it is compared with the syntax requirements of the reduction that follows. This process continues until the syntax requirements of some reduction are matched.

When a token phrase matches the syntax specifications of a particular reduction, the phrase is translated by invoking the action routines given in the action field of that reduction. Action routines can be simple PL/I statements or calls to PL/I subroutines with arguments. The routines can perform some constant translation operation, or an operation that depends on the values of one or more tokens in the matching token phrase. They can also skip over one or more of the tokens in the matching token phrase to permit the next token phrase to be examined.

After the action routines have been invoked, the next-reduction field of the matched reduction controls which reduction syntax field the next token phrase is compared with. The next reduction can be identified by label, using one of the reduction labels given in a label field. Also, the reduction following the matched reduction can be used next. In addition, special next-reduction operations are provided to return from the SEMANTIC_ANALYSIS procedure, and to return from a group of reduction statements used as a reduction subroutine.

LABEL FIELD OF A REDUCTION STATEMENT

One or more labels can be specified in the label field of a reduction statement to identify the reduction. A label is a character string that begins with an alphabetic character, and contains 32 or fewer alphanumeric or underline (_) characters.

The labels on a reduction statement can be referenced in the next-reduction field of reduction statements to direct the order in which tokens are compared with the reduction syntax specifications. To prevent any ambiguities in these references, each of the labels defined in a set of reductions must be unique.

In every set of reductions, any attribute declarations that are given must appear before all of the reduction statements. To distinguish between the attribute declarations and reduction statements, the first reduction statement must have a special first label called BEGIN.

The BEGIN label acts as a keyword that separates the attribute declarations from the reduction statements. It also identifies the first reduction with which token phrases are compared. Thus, the comparison of token phrases with reductions starts with this beginning reduction, the first reduction following the attribute declarations, the reduction with the BEGIN label.

With the exception of the BEGIN label on the first reduction statement, no labels are required on any reduction statement. Their use is optional, and is intended to facilitate the division of the set of reductions into groups of reduction statements or reduction subroutines. However, every reduction statement must have a label field even if it consists of an empty label field with a field delimiter (/). All four of the fields mentioned above must appear in every reduction statement.

Use of reduction labels is discussed further in the description of "The Next-Reduction Field" below.

SYNTAX FIELD OF A REDUCTION STATEMENT

The syntax field of a reduction statement defines the syntax of one token phrase in the language being translated. The tokens in the input list are compared with the syntax fields of one or more reductions. When the tokens match the syntax field of a reduction, then the action field of that reduction is invoked to perform a translation action. If the reduction specifies the syntax for a valid token phrase, the translation action can compile code to implement the semantic meaning of the phrase or it can immediately interpret the phrase or store a value in a table or perform any other translation action. If the reduction specifies the syntax for an invalid token phrase, then the translation action can diagnose the error in an error message.

CURRENT TOKEN PHRASE

Before learning how syntax specifications are defined, some terminology for dealing with the tokens in the token list must be developed.

In Figure 3-3 above, a list of tokens is described by token descriptors that are chained together. The reductions command declares a pointer in the main procedure of the translator that points to the particular tokens being compared with reductions at any given time. This pointer is called Pthis_token, and it points to the descriptor of the "current token." The current token and those tokens that follow it in the list of tokens are the tokens being compared with the reduction syntax specifications. This group of tokens is called the "current token phrase." These relationships are shown in Figure 3-6 below.

Notice that the current token phrase does not contain a fixed number of tokens. Instead, the number of tokens varies to accommodate the number of syntax specifications in the reduction being examined. Of course, if there are fewer tokens remaining to be translated than syntax specifications in a reduction, the current token phrase cannot match that reduction.

At any point in time, one of the tokens in the current token phrase is being compared with its corresponding syntax specification in a reduction. The descriptor for this token is pointed to by Ptoken, another pointer variable declared by the reductions command in the main procedure of the translator.

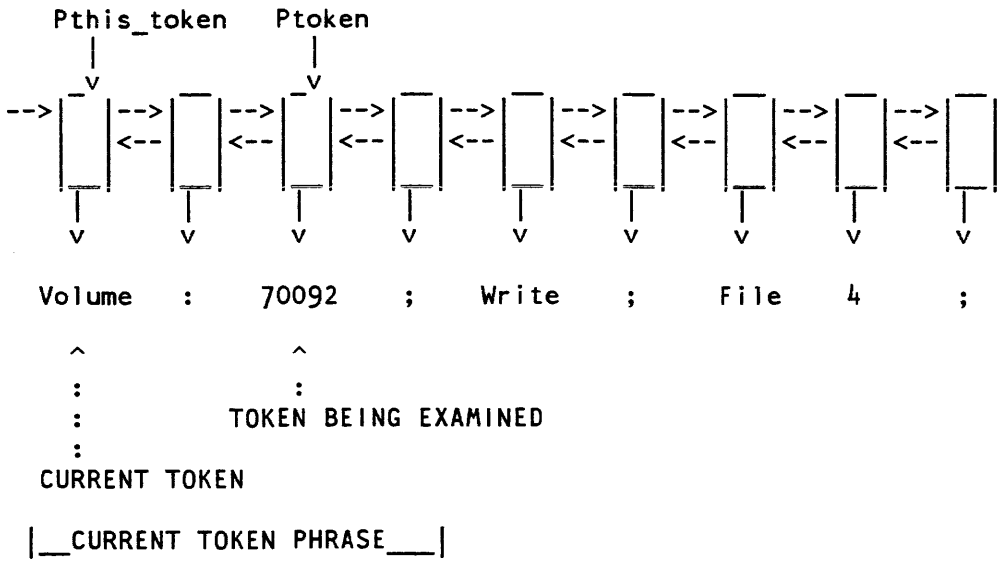


Figure 3–6. The Current Token Phrase Used by Reductions

SYNTAX SPECIFICATIONS

The tokens in the current token phrase are compared consecutively with the syntax specifications in a reduction syntax field to identify valid and invalid token phrases. The syntax specifications place requirements on the tokens in the current token phrase. If each token in the phrase meets the requirements of its corresponding syntax specification in the reduction, the entire phrase matches the reduction, and the reduction action field is invoked.

Three types of syntax specifications are allowed by the reduction language: absolute syntax specifications, relative syntax functions, and built-in syntax functions.

ABSOLUTE SYNTAX SPECIFICATIONS

Absolute syntax specifications require that their corresponding input token equal a particular character string. Absolute specifications are defined in the syntax field of a reduction statement by using their character string value. For example, a reduction statement that would identify the first two tokens in Figure 3–6 might be

```
vol_stmt  / Volume :          /                /        \
```

If reductions were written to translate all of the tokens in Figure 3–6 then "Volume," "Write," "File," ":", and ";" would probably be specified as absolute syntax specifications.

The delimiter characters used in the reduction language (see Table 3-1 above) can be used in an absolute specification by enclosing the entire specification in quotes. For example, "and/or", ">udd>Project_id>prog", """""", "(", and ",". In addition, the delimiters that have a special meaning within the syntax field (/ < >) can be used as one-character absolute specifications by underlining the delimiter character. and thus are treated as single-character absolute syntax specifications.

RELATIVE SYNTAX FUNCTIONS

Relative syntax functions are a second type of syntax specification. A relative syntax function requires that its corresponding input token meet some special requirements that are defined by a PL/I function subprogram. The requirements defined by such functions can be quite specific or very general in nature.

A relative syntax function is defined as a specification in the syntax field of a reduction by enclosing the name of the function in angle brackets (i.e., <function_name>). For example, if the volume_id function defines the requirements for a volume identifier like that used in Figure 3-6, the following reduction would match the first four tokens of Figure 3-6.

```
vol_stmt  / Volume : <volume_id> ; /              /        \
```

Other examples of relative syntax functions might be a <relative_pathname> function that requires that a token be a relative pathname, and that calls the absolute_pathname_ subroutine to associate an absolute pathname as the semantic value of this pathname token; a <positive_integer> function that requires that the token be a character string representation of a positive integer; and <date_time> that requires a token that is acceptable as input to the convert_date_to_binary_ subroutine.

Relative syntax functions must be coded by the programmer and included in the main procedure of the translator source segment. Their calling sequence is shown below.

```
declare function_name entry returns (bit(1) aligned);

token_meets_requirements = function_name();
```

where the function returns a value of "1"b if the input token meets the requirements of the function, and "0"b otherwise. The function can have any valid PL/I function name that is 32 or fewer alphanumeric or underline characters in length, and that contains at least one lowercase alphabetic character. The lowercase letter is required to avoid naming conflicts with variables and procedures declared by rdc for use in the SEMANTIC_ANALYSIS procedure.

Relative syntax functions must be internal procedures of the main procedure of the translator so that they can reference the token to be examined. Ptoken points to the descriptor for this token as shown in Figure 3-6. The token descriptor itself is a structure variable named token that is based on Ptoken, as described in the lex_string_ subroutine description. The character string value of the token can be referenced by way of the token_value variable. Ptoken, the token structure, and token_value are variables declared by the reductions command in the main procedure of the translator.

A relative syntax function can associate a semantic value with the token being examined in one of three ways. It can set a variable that has been declared in the main procedure of the translator. It can set token.Nvalue to some integer semantic value, such as the numeric value of a token that matches the <positive_integer> function. Or it can allocate a semantic value structure in the temporary segment used for token descriptors, and chain this structure onto the token descriptor using the token.Psemant pointer. Refer to the description of the lex_string_ subroutine for a complete declaration of the token structure.

LIST OF BUILT-IN SYNTAX FUNCTIONS

The third type of syntax specification that can be used in a reduction syntax field is the built-in syntax function. These are relative syntax functions that have been predefined by the reductions command. Although several of these built-in syntax functions make requirements on the input token string that would be difficult to implement directly as relative syntax functions, most of the built-in syntax functions are defined merely to facilitate the implementation of the reductions command itself. Below is a list of the built-in syntax functions and the requirements they place on the input tokens.

<no-token>
: requires that no corresponding token exists in the current token phrase, that the list of input tokens is exhausted, and that no more tokens remain to be translated. It differs from other syntax functions that require the existence of a corresponding token in the token phrase. It is used to determine when the translation is complete.

<any-token>
: requires that a corresponding token exist in the current token phrase. It places no other requirements on the token. It is used when any token value is acceptable in the language being translated.

<name>
: requires that a corresponding token exist in the current token phrase, and that the token is a character string that begins with an alphabetic character and contains 32 or fewer alphanumeric, underline (_), or dollar sign ($) characters.

<decimal-integer>
: requires that a corresponding token exist in the current token phrase, and that the token is a valid, optionally signed decimal integer (as defined by the cv_dec_check_ subroutine). The numeric value of the token is stored as its semantic value in the token.Nvalue element of the token descriptor.

<quoted-string>
: requires that a corresponding token exist in the current token phrase, and that the token.S.quoted_string bit is turned on in the descriptor of the token. The lex_string_ subroutine turns on this bit if the token is enclosed within quoting delimiters when the input to the translator is parsed.

<BS>
requires that a corresponding token exist in the current token phrase, and that the token is a single backspace character. It is used as a convenience for defining syntax specifications for one-character, underlined tokens.

COMPLETENESS OF THE SYNTAX SPECIFICATIONS

One of the most difficult aspects of writing a translator is identifying all possible invalid token phrases that could be received as input so that error messages can be issued. This problem must be addressed in each set of reductions, and in each group of reductions within a set as well, if the translator is to operate deterministically and to perform the expected translation.

A typical solution for the problem is to have a group of reductions that identify all possible valid token phrases, followed by one or more reductions that use the <any-other> built-in syntax function or an empty syntax field to identify all other invalid token phrases. For example, if the language for the tokens in Figure 3–6 requires that a colon, volume identifier, and semicolon always follow the Volume keyword, then the following group of reductions might be used to diagnose an error.

```
vol_stmt  / Volume : <volume_id> ; /              /          \
          / Volume : <any-token> ; /              /          \
            \" check for bad volume identifier.
          / Volume                 /              /          \
            \" check for bad volume statement.
          /                        /              /          \
            \" check for unknown or unexpected statement.
```

THE NEXT-REDUCTION FIELD OF A REDUCTION STATEMENT

The next-reduction field governs the flow of control between reductions. When the translator calls the SEMANTIC_ANALYSIS procedure, control passes to the reduction whose label is BEGIN. The first of the current token phrases is compared with this beginning reduction and those that follow until it matches the syntax requirements of one of the reductions. The action field of that reduction is then invoked to translate to the current token phrase, and to make the next token phrase current.

The next-reduction field of the matched reduction controls which reduction the new current token phrase is compared with. The next-reduction field can be blank, or it can contain a reduction label. If it is blank, the reduction immediately following the matched reduction is used in the next comparison. If a reduction label is specified, then the reduction identified by that label is used in the next comparison. In either case, comparison of the new current token phrase with reductions continues until a matching reduction is found.

This process of analyzing token phrases continues until all of the input tokens have been translated. Each set of reductions must contain one or more reductions that use the <no-token> built-in syntax function to detect when all the input tokens have been translated. When such a <no-token> reduction is invoked, its next-reduction field usually contains the RETURN keyword, instead of a reduction label, to specify that the flow of control should return to the caller of the SEMANTIC_ANALYSIS procedure. On return from SEMANTIC_ANALYSIS, the translation is complete.

Often if several <no-token> reductions appear in a set of reductions, a reduction label is used in their next-reduction field (rather than a RETURN keyword) to branch to a final <no-token> reduction that performs epilogue actions and then returns via a RETURN keyword. Having only one of the <no-token> reductions perform the epilogue actions reduces the amount of translation code generated by rdc.

SAMPLE REDUCTIONS

Figure 3-7 shows the Backus-Naur Form (BNF) for the syntax of a language that identifies records to be read or written from a tape file on a particular volume, using a given record format. Several examples below employ this language to illustrate the use of reductions.

```
<spec>  ::=     Volume : <volume-id>[,{9track|7track}] ;
                {Read|Write} ;
                File <number> ;
                Records : <number>[, <number>]... ;
                Format : {F|FB|FBS|V|VB|VBS|U} ;
```

Figure 3-7. BNF Syntax for a Tape Language

Figure 3-8 shows how reduction statements can be used to define the syntax of the tape language (see "Relative Syntax Functions" above.)
Note that reductions containing only an <any-token> or <no-token> syntax specification are included in each group of reductions to detect errors. The <any-token> reduction matches any token phrase except the empty token phrase (a phrase containing no tokens because all of the input tokens have been translated). The <no-token> reduction matches empty token phrases.

```
BEGIN
stmt     / Volume : <volume_id>          /          / vol      \
         / Read ;                        /          / stmt     \
         / Write ;                       /          / stmt     \
         / File <positive_integer> ;     /          / stmt     \
         / Records :                     /          / numbers  \
         / Format :                      /          / format   \
         / <any-token>                   /          / stmt     \
         / <no-token>                    /          / RETURN   \

vol      / ;                             /          / stmt     \
         / , 9track ;                    /          / tmt      \
         / , 7track ;                    /          / stmt     \
         / <any-token>                   /          / stmt     \
         / <no-token>                    /          / RETURN   \

numbers  / <positive_integer>            /          / punct    \
         / <any-token>                   /          / punct    \
         / <no-token>                    /          / RETURN   \

punct    / ,                             /          / numbers  \
         / ;                             /          / stmt     \
         / <any-token>                   /          / numbers  \
         / <no-token>                    /          / RETURN   \

format   / F ;                           /          / stmt     \
         / FB ;                          /          / stmt     \
         / FBS ;                         /          / stmt     \
         / V ;                           /          / stmt     \
         / VB ;                          /          / stmt     \
         / VBS ;                         /          / stmt     \
         / U ;                           /          / stmt     \
         / <any-token>                   /          / stmt     \
         / <no-token>                    /          / RETURN   \
```

Figure 3-8. Reductions for the Tape Language

ACTION FIELD OF A REDUCTION STATEMENT

When a valid token phrase matches the syntax specifications of a reduction statement, the phrase must be translated according to the semantics of the source language. The translator does this by invoking the action routines specified in the action field of the matched reduction. These routines are invoked in the order of their appearance in the action field.

There are two types of action routines: those that perform some translation action on the current token phrase, and those that perform a lexing action to make another token the current token so that a new token phrase can be translated. Translation action routines are described below, and lexing routines are described under "Lexing Action Routines," which follows.

TRANSLATION ACTION ROUTINES

Translation action routines translate token phrases that match reductions according to the semantics of the source language. For example, they can construct tables; build compilation trees; generate object code, ALM statements, or PL/I statements; or perform any other type of translation function that can be expressed in the PL/I language.

There are two kinds of translation action routines: action statements and calls to action subroutines.

ACTION STATEMENTS

An action statement is a PL/I statement that appears in the action field of a reduction, enclosed in square brackets without its semicolon statement delimiter. For example, a tape language source string of

```
Write;
```

might be translated by setting a mode variable as follows:

```
[mode = "w"]
```

Action statements can be used to perform the simplest translation operations, such as turning on a bit or assigning a particular value to a variable. Such simple operations occur frequently in translators, and are most clearly and easily expressed as a PL/I statement. Action statements can use the token_value variable, just as relative syntax functions do, to reference the character string value of the current token. For example, the tape language string

```
Volume: 70092;
```

might be translated by making 70092 the current token, and then invoking an action statement like

```
[volume = token_value]
```

Action statements can also use the token structure to reference the descriptor of the current token or a semantic value structure chained to the descriptor. For example, the tape language source string

```
File 4;
```

might be translated by a reduction of the form

```
/ File <positive_integer> ;     / LEX [file_no=token.Nvalue]
                                          LEX(2)      /         \
```

where LEX and LEX(2) are a lexing action routines that make the next, or second next, token be the current token. Notice that, in the reduction above, the <positive_integer> relative syntax function sets token.Nvalue when it validates the syntax of the "4" token.

More than one PL/I statement can be used as an action statement if the PL/I statements are separated by a semicolon (;). This allows compound PL/I statements to be used as action statements. For example, the action statement

```
[if token_value = "SCRATCH" then volume = "scratch";
                            else volume = token_value]
```

checks for the special tape volume name SCRATCH and uses scratch in its place if found; otherwise, the token value given in the source string is used as the volume name.

ACTION SUBROUTINES

An action subroutine is a PL/I subroutine that performs some translation operation. It appears in the action field of a reduction as a PL/I call statement, without the call keyword or the semicolon statement delimiter. For example, the subroutine

```
call perform_io ("tape_input", volume, file_no, mode, "1"b);
```

appears in the action field as:

```
perform_io ("tape_input", volume, file_no, mode, "1"b)
```

A subroutine with no arguments, such as:

```
call set_record_no();
```

appears as:

```
set_record_no
```

An example of a reduction containing action subroutines is

```
/ <no-token>          / perform_io("tape_input",
                        volume, file_no, mode, "1"b)/        \
```

The programmer must supply these action subroutines as part of the translator. Usually they are internal procedures defined in the main procedure of the translator. This facilitates references to the tokens being translated and to other data declared in the translator. However, an external procedure can be used as an action subroutine by calling it with arguments to pass any required information.

NAMING REQUIREMENTS FOR ACTION ROUTINES

Several facts must be considered when defining action subroutines and other variables used in the action field of a reduction. First, action statements and subroutines are executed within the SEMANTIC_ANALYSIS procedure. Therefore, all variables used in action statements or as arguments to action subroutines must be declared in the main procedure of the translator. Similarly, internal action subroutines must be defined in this main translator procedure, and external action subroutines must be declared there. Figure 3-2 illustrates the relationship between the main translator procedure and the SEMANTIC_ANALYSIS procedure.

Second, care must be taken to avoid naming conflicts between the variables declared by SEMANTIC_ANALYSIS and the variables and subroutines used in the action field. With only a few exceptions, the variables used by SEMANTIC_ANALYSIS have uppercase names. Therefore, the programmer can avoid name conflicts by using names with one or more lowercase letters or digits.

There are three classes of exceptions to the uppercase naming rules used in SEMANTIC_ANALYSIS. First, SEMANTIC_ANALYSIS has declared the following PL/I built-in functions: addr, max, null, search, substr, and verify. Second, SEMANTIC_ANALYSIS has declared the cv_dec_check_ subroutine to implement the `<decimal-integer>` built-in syntax function. Third, the variables and structures required to reference tokens and their descriptors are declared by the reductions command in the main procedure of the translator. SEMANTIC_ANALYSIS assumes the existence of these declarations, which have lowercase names. (Refer to the description of the lex_string_ subroutine for a complete declaration of these variables.) All three classes of exceptions must be avoided when naming variables and action subroutines.

LIST OF LEXING ACTION ROUTINES

Lexing action routines are useful in two ways. They can skip over a token phrase once it has been translated so that the next token phrase can be analyzed. Also, they can skip from the first token of a phrase to another of its tokens so that a translation action routine can reference that token. By default, the first token of the phrase that matches the reduction syntax field is the current token when the routines in the action field are invoked.

The following lexing action routines are provided by the reductions command.

LEX(N)
: makes the Nth token the new current token, where N is the token number relative to the existing current token. The current token has a relative token number of 0. Positive relative token numbers denote tokens following the current token, while negative numbers denote tokens preceding the current token. Thus, LEX(3) makes the third token following the current token the new current token.

LEX
: is equivalent to LEX(1).

NEXT_STMT
: makes the first token of the next statement (the statement following the statement that contains the current token) the new current token. This lexing routine can only be used when the tokens have been parsed with statement descriptors. NEXT_STMT is most useful to skip over the remaining tokens of a statement when an unrecoverable error has been detected in the statement.

DELETE(M,N)
: unthreads tokens from the token list so that they are not scanned by subsequent reductions. Tokens are unthreaded (deleted) from the Mth token relative to the current token through the Nth relative token. Thus, DELETE(2,3) deletes the second and third tokens following the current token. When the current token is one of those being deleted, the next token following those deleted becomes the current token. Thus, DELETE(-1,+1) deletes the token preceding the current token, the current token, and the token following the current token, and makes the second token following the current token the new current token.

DELETE(N)
: is equivalent to DELETE(N,N).

DELETE
: is equivalent to DELETE(0,0).

DELETE_STMT
: deletes all tokens of the current statement, making the first token of the next statement the new current token. The current statement is the statement containing the current tokens. DELETE_STMT can only be used when the tokens have been parsed with statement descriptors.

USING LEXING ROUTINES IN TRANSLATION SUBROUTINES

Lexing action routines can be invoked from translation action subroutines if it is necessary for the subroutine to examine more than one token in the current token phrase. However, use of lexing routines from translation subroutines can obscure the translation process because the lexing is performed unexpectedly by a translation subroutine, rather than in the action field of a reduction where it is highly visible. If a translation routine examines only one token, it is best to place a LEX operation in the action field to make the desired token current before the translation routine is

invoked. If the routine must examine several tokens, it is best to position to the first of these tokens before the routine is invoked and to include the number of tokens skipped over by the routine in its subroutine name (e.g., process_names_and_LEX2). Such naming makes the lexing action of the translation routine more visible when reading the reductions.

A translation subroutine can call the internal procedures that rdc defines in a translator to perform the lexing actions. These internal procedures have the following calling sequences.

```
declare LEX entry (fixed bin);

call LEX(N);
```

where N is the token number relative to the existing current token.

```
declare NEXT_STMT entry;

call NEXT_STMT();

declare DELETE entry (fixed bin, fixed bin);

call DELETE (M, N);
```

where tokens are unthreaded (deleted) from the Mth token relative to the current token through the Nth relative token.

```
declare DELETE_STMT entry;

call DELETE_STMT();
```

Notice that only the two-argument version of DELETE and the one-argument version of LEX can be used from translation routines. If the particular routine to be called has not been used in any reduction, it must be explicitly included in the translator by using an INCLUDE attribute declaration statement, as described below under "Attribute Declarations."

SAMPLE REDUCTIONS

Figure 3-9 shows the reductions for our tape language, with the action fields filled in. Notice that only one of the <no-token> reductions performs epilogue functions, and that this reduction receives control from all other <no-token> reductions. The action field of reductions that identify invalid phrases have not, as yet, been specified.

```
BEGIN
stmt    / Volume : <volume_id>       / LEX(2) [volume=token_value]
                                       [track = 9] LEX   / vol    \
        / Read ;                     / LEX(2) [mode="r"] / stmt   \
        / Write ;                    / LEX(2) [mode="w"] / stmt   \
        / File <positive_integer> ;  / LEX [file_no=token.Nvalue]
                                       LEX(2)            / stmt   \
        / Records :                  / LEX(2)            / numbers\
        / Format :                   / LEX(2)            / format \
        / <any-token>                / NEXT_STMT         / stmt   \
        / <no-token>                 / perform_io("tape_input",
                                       volume, file_no,
                                       mode, "1"b)       / end    \

vol     / ;                          / LEX               / stmt   \
        / , 9track ;                 / LEX(3)            / stmt   \
        / , 7track ;                 / [track = 7] LEX(3)/ stmt   \
        / <any-token>                / NEXT_STMT         / stmt   \
        / <no-token>                 /                   / end    \

numbers / <positive_integer>         / set_record_no LEX / punct  \
        / <any-token>                / LEX               / punct  \
        / <no-token>                 /                   / end    \

punct   / ,                          / LEX               / numbers\
        / ;                          / LEX               / stmt   \
        / <any-token>                / LEX               / numbers\
        / <no-token>                 /                   / end    \

format  / F ;                        / LEX(2) format(1)  / stmt   \
        / FB ;                       / LEX(2) format(2)  / stmt   \
        / FBS ;                      / LEX(2) format(3)  / stmt   \
        / V ;                        / LEX(2) format(4)  / stmt   \
        / VB ;                       / LEX(2) format(5)  / stmt   \
        / VBS ;                      / LEX(2) format(6)  / stmt   \
        / U ;                        / LEX(2) format(7)  / stmt   \
        / <any-token>                / NEXT_STMT         / stmt   \
        / <no-token>                 /                   /        \

end     / <any-token>                / epilogue          / RETURN \
        / <no-token>                 / epilogue          / RETURN \
```

Figure 3–9. Reductions for the Tape Language
(Error-Diagnosing Actions Omitted)

ERROR-DIAGNOSING ACTION ROUTINES

Translators must identify and translate all valid token phrases in the source string, and must identify and diagnose all invalid token phrases to aid in their correction. Invalid token phrases can be detected in several ways.

1. Following a series of reductions that identify the valid token phrases for a given language construct, a reduction with an <any-token> syntax specification can be used to match all other invalid token phrases.

2. Specific reductions can identify predictable errors, such as tokens that do not match the specifications of the relative syntax function in a preceding reduction, or token phrases that have missing or invalid punctuation, misspelled or invalid keywords, and the like.

3. A reduction with a <no-token> syntax specification can be used to detect a premature end of the source string.

4. Action routines can detect an inconsistency in the semantic meaning of the source string and can diagnose the error.

When an error is detected, the translator must notify you of the type and location of the error. The reductions command provides two facilities for printing error messages: the ERROR action subroutine and the lex_error_ external subroutine.

THE ERROR ACTION SUBROUTINE

The ERROR action subroutine is an internal procedure provided by the reductions command to print error messages. It can be called as follows:

```
declare ERROR entry (fixed bin(17));

call ERROR (error_number);
```

The error_number can be an arithmetic constant or the name of a PL/I variable that can be converted to a fixed binary number. For example,

```
call ERROR (5);
```

or

```
declare missing_semicolon_error fixed bin(17) internal static
                                options(constant) init(5);
call ERROR (missing_semicolon_error);
```

ERROR can be used in the action field of a reduction that identifies invalid token phrases. For example,

```
/ <any-token> / ERROR(1) NEXT_STMT / stmt \
```

or it can be called from an action subroutine to diagnose a semantic inconsistency.

ERROR prints messages that have the following form:

```
prefix error_number, SEVERITY severity_no IN STATEMENT k OF LINE l.
error_message_text
SOURCE:
statement_containing_current_token_phrase
```

For example,

```
ERROR 7, SEVERITY 2 IN STATEMENT 2 OF LINE 2.
A bad track specification was given in a Volume statement.
9track has been assumed.
SOURCE:
Volume: 70082, 8track;
```

ERROR prints the error messages declared in an error_control_table structure array variable that the programmer declares in the main procedure of the translator. Each structure element in the array defines an error message, and the error_number is the array index of the desired error message. The structure contains a severity level associated with the error, a switch that controls the printing of the current statement as part of the error message, a long form of the error message text, and a brief form of the error message text. The error_control_table must be declared as a one-dimensional array of structures, with a lower bound of one, and an upper bound equal to the highest error_number that can be used. Figure 3-10 below shows a typical error_control_table declaration.

```
dcl 1 error_control_table (7) internal static options(constant),
      2 severity fixed bin(17) unaligned init (3, 2, 3, 2, 3, 2, 2),
      2 Soutput_stmt bit(1) unaligned
        init ("1"b, "1"b, "0"b, "1"b, "1"b, "1"b, "1"b),
      2 message char(70) varying init(
        "An unknown statement has been encountered.",
        "'^a' is an invalid record number.",
        "Translator input ends with an incomplete statement.",
        "'^a' is invalid punctuation in a list of record numbers.",
        "'^a' is an invalid record format.",
        "Input follows the end of the tape file specification.",
        "A bad track specification was given in a Volume statement.
        9track has been assumed."),
      2 brief_message char(28) varying init(
        "Unknown statement.",
        "Bad record number '^a'.",
        "Incomplete statement.",
        "Invalid punctuation '^a'.",
        "Invalid record format '^a'.",
        "Too much input.",
        "Bad track in Volume.");
```

Figure 3-10. error_control_table for the Tape Language

The severity_no associated with an error controls the prefix that is placed in the error message, as shown in Table 3-2 below.

Table 3-2. RELATIONSHIP OF error_control_table.severity_no TO ERROR MESSAGE PREFIX

SEVERITY	PREFIX	EXPLANATION
0	COMMENT	Comment. The error message is a comment, which does not indicate that an error has occurred, but merely provides information for you.
1	WARNING	Warning only. The error message warns of a statement that may or may not be in error, but compilation continues without ill effect.
2	ERROR	Correctable error. The message diagnoses an error that the translator can correct, probably without ill effect. Compilation continues, but correct results cannot be guaranteed.

3	FATAL ERROR	An uncorrectable but recoverable error. The translator has detected an error that it cannot correct. Translation continues in an attempt to diagnose further errors, but no output is produced by the translation.
4	TRANSLATOR ERROR	An unrecoverable error. The translator cannot continue beyond this error. The translation is aborted after the error message is printed.

The statement and line numbers in the printed message are obtained from the descriptor of the current statement, if statement descriptors are available, or from the descriptor of the current token.

The phrase "IN STATEMENT k OF LINE l" appears if statement descriptors are available. Line l is the line number on which the statement containing the current token begins. Statement k identifies which statement in line l is in error, if more than one statement appears in line l. "STATEMENT k OF" is omitted from the message if only one statement appears in Line l.

If no statement descriptors are available, the phrase "STATEMENT k OF" is omitted from the message. Line l is the line number on which the current token appears.

If Pthis_token is null, the phrase "IN STATEMENT k OF LINE l" is omitted altogether, since there is no current statement and no current token. When the output_stmt_sw of an error is on, the current statement is included in the printed error message. The stmt.output_in_err_msg switch is turned on in the statement descriptor to prevent the source statement from being reprinted in subsequent error messages. Since the current statement is obtained from its statement descriptor, the translator must parse its source string with statement descriptors. If statement descriptors are not present, error_control_table.output_stmt_sw has no effect. Refer to the description of the lex_string_ subroutine for information about the structure, contents, and generation of statement descriptors.

The printed error message contains either the error_message_text or the brief_message_text, depending upon the value of the SERROR_CONTROL variable. This variable is declared by the reductions command, in the main procedure of the translator, as follows:

```
dcl SERROR_CONTROL bit(2) initial ("00"b);
```

Table 3-4 below shows how the setting of these bits controls the r_message_text in the printed error message.

Table 3-3. SERROR_CONTROL BITS CONTROL THE error_message_text

SERROR_CONTROL	INTERPRETATION
"00"b	The printed error contains the error_message_text the first time the error occurs and the brief_message_text for subsequent occurrences of that error during a given translation.
"10"b	The printed error always contains the error_message_text.
"11"b	The printed error always contains the error_message_text.
"01"b	The printed error always contains the brief_message_text.

The reductions command declares the SERROR_PRINTED variable in the main procedure of the translator as follows:

```
dcl SERROR_PRINTED (dimension(error_control_table),1) bit(1) unaligned
   initial(dimension(error_control_table,1) (1)"0"b);
```

The ERROR routine turns on SERROR_PRINTED(error_number) whenever an error message is printed, and uses the current value of SERROR_PRINTED to control the printing of the error_message_text or brief_message_text when SERROR_CONTROL equals "00"b.

The translator can be implemented with control arguments to alter the use of error_message_text or brief_message_text in error messages. For example, the reductions command uses the normal value ("00"b) by default, but implements the -brief (-bf) control argument to set a brief value ("01"b) and the -long (-lg) control argument to set a long value ("10"b).

The error_message_text and brief_message_text of an error are defined as ioa_ control strings that can contain up to three occurrences of the ^a control code. Each occurrence of ^a is replaced by the token_value character string value of the current token. In addition, any number of the following ioa_ control codes that do not require an input argument can be used in the error_message_text and brief_message_text strings: ^-, ^/, ^|, ^x, and ^^. The ioa_ subroutine imposes a maximum length of 256 characters on the error_message_text and on the brief_message_text after all ioa_ substitutions have been performed.

The ERROR routine maintains the severity of the highest severity error encountered during a translation in the variable

```
dcl MERROR_SEVERITY fixed bin(17) initial (0);
```

which the reductions command declares in the main procedure of the translator. The translator can reference the value of this variable to determine whether an uncorrectable error has occurred or to determine when to abort the translation due to an unrecoverable error.

The ERROR routine is also controlled by the values of MIN_PRINT_SEVERITY and PRINT_SEVERITY_CONTROL. These variables are declared by the rdc command, in the main procedure of the translator, as follows:

```
dcl MIN_PRINT_SEVERITY fixed bin initial (0);
dcl PRINT_SEVERITY_CONTROL bit (2) aligned init ("11"b);
```

MIN_PRINT_SEVERITY defines the minimum severity of error that is printed; that is, all calls to ERROR having error_control_table (error_number).severity_no greater or equal to MIN_PRINT_SEVERITY results in calls to lex_error_. The lex_error_ subroutine operates on the values of MERROR_SEVERITY and SERROR_PRINTED as part of its function. If the severity associated with the error is less than MIN_PRINT_SEVERITY, the ERROR routine does not call lex_error_; however, according to the value of PRINT_SEVERITY_CONTROL, it manipulates the values of MERROR_SEVERITY and SERROR_PRINTED. Table 3-4 shows this interaction.

Table 3-4. PRINT_SEVERITY_CONTROL Bits Control the Values of MERROR_SEVERITY and SERROR_PRINTED

PRINT_SEVERITY_CONTROL	INTERPRETATION
"00"b	Neither MERROR_SEVERITY nor SERROR_PRINTED are changed.
"01"b	SERROR_PRINTED is updated as though the error had been printed.
"10"b	MERROR_SEVERITY is updated as though the error had been printed.
"11"b	MERROR_SEVERITY and SERROR_PRINTED are both updated as though the error had been printed.

The ERROR action routine and declarations for SERROR_CONTROL, SERROR_PRINTED, and MERROR_SEVERITY are automatically included in the main procedure of the translation when ERROR is used in the action field of one or more reductions. An INCLUDE attribute declaration can be used to include these error diagnostic facilities when the ERROR routine is used only by other action routines, and does not appear in any reductions. Refer to "Attribute Declarations" below for more information.

THE lex_error_ SUBROUTINE

The ERROR action routine is a very simple diagnostic tool, but this simplicity is possible only because ERROR does not generate highly specific error messages containing several different variable information fields. ERROR only allows the character string value of the current token to be included in the message.

ERROR uses lex_error_ to print its error messages. The translator can call lex_error_ directly to produce more flexible error messages. In this way, error messages containing more than one token value, or containing variables defined by the translator, can be printed using a standard mechanism. (See the lex_error_ subroutine.)

SAMPLE REDUCTIONS--COMPLETE

Figure 3-11 shows the reductions for the tape language with errors being diagnosed by the ERROR action routine.

```
BEGIN
stmt    / Volume : <volume_id>       / LEX(2) [volume=token_value]
                                       [track = 9] LEX      / vol    \
        / Read ;                     / LEX(2) [mode="r"]    / stmt   \
        / Write ;                    / LEX(2) [mode="w"]    / stmt   \
        / File <positive_integer> ;  / LEX [file_no=token.Nvalue]
                                       LEX(2)               / stmt   \
        / Records :                  / LEX(2)               / numbers\
        / Format :                   / LEX(2)               / format \
        / <any-token>                / (ERROR(1) NEXT_STMT  / stmt   \
        / <no-token>                 / perform_io("tape_input",
                                       volume, file_no,
                                       mode, "1"b)          / end    \
vol     / ;                          / LEX                  / stmt   \
        / , 9track ;                 / LEX(3)               / stmt   \
        / , 7track ;                 / [track = 7] LEX(3)   / stmt   \
        / <any-token>                / ERROR(7) NEXT_STMT   / stmt   \
        / <no-token>                 / ERROR(3)             / end    \

numbers / <positive_integer>         / set_record_no LEX    / punct  \
        / <any-token>                / ERROR(2) LEX         / punct  \
        / <no-token>                 / ERROR(3)             / end    \
punct   / ,                          / LEX                  / numbers\
        / ;                          / LEX                  / stmt   \
        / <any-token>                / ERROR(4) LEX         / numbers\
        / <no-token>                 / ERROR(3)             / end    \
format  / F ;                        / LEX(2) format(1)     / stmt   \
        / FB ;                       / LEX(2) format(2)     / stmt   \
        / FBS ;                      / LEX(2) format(3)     / stmt   \
        / V ;                        / LEX(2) format(4)     / stmt   \
        / VB ;                       / LEX(2) format(5)     / stmt   \
        / VBS ;                      / LEX(2) format(6)     / stmt   \
        / U ;                        / LEX(2) format(7)     / stmt   \
        / <any-token>                / ERROR(5) NEXT_STMT   / stmt   \
        / <no-token>                 / ERROR(3)             /        \
end     / <any-token>                / ERROR(6) epilogue    / RETURN \
        / <no-token>                 / epilogue             / RETURN \
```

Figure 3-11. Complete Reductions for the Tape Language

REDUCTION SUBROUTINES

Often a new language contains phrases that are similar in form but that differ in their use of keywords, types of keyword operand values expected, or in other minor ways. As an example, the value language specified in Figure 3-12 below includes three types of statements, each of which begins with a keyword followed by a punctuated list of keyword operand values.

```
<stmt>   ::=   Name : <name>[,<name>]... ;
             | Attribute : <attr>[,<attr>]... ;
             | Value : <number>[,<number>]... ;

<name>   ::=   is the name of a variable.

<attr>   ::=   fixed | float | decimal | binary

<number> ::=   is a numeric value to be assigned to a variable.
```

Figure 3-12. BNF Specification for the Value Language

Since all the punctuated lists used in each statement have the same form, a single group of reductions can be written to translate the punctuation for all three types of statements. This sharing of reductions reduces the total number of reductions needed to translate the value language. Reduction subroutines provide the facility for shared reductions.

A reduction subroutine is a group of reductions. As with a PL/I subroutine, a reduction subroutine has a primary entry point named by the label given in the label field of its first reduction. Alternate entry points are identified by the labels on other reductions in the subroutine. For example, the following reduction subroutine has a primary entry point of punct and an alternate entry of punct_no_comma.

```
punct      / ,                  / LEX                    / STACK_POP\
punct_no_comma
           / ;                  / LEX                    / STACK_POP\
           / <any-token>        / ERROR(7) NEXT_STMT     / stmt     \
           / <no-token>         / ERROR(4)               / RETURN   \
```

A reduction calls a reduction subroutine by storing a return label in a label stack (a pushdown stack of label values), and then giving the subroutine entry point name in the next-reduction field. The subroutine reduction labeled by that entry point name is then the next reduction that is compared with the current token phrase. When the reduction subroutine has completed its translation of input tokens, it returns to the calling reduction (or group of reductions) at a label that the caller stores in a label stack prior to the call. For example, the punct subroutine shown above is called by each reduction in the group shown below.

```
attr      / fixed          / LEX attr(1) PUSH(attr)    / punct    \
          / float          / LEX attr(2) PUSH(attr)    / punct    \
          / <any-token>    / ERROR(5) LEX PUSH(attr)   / punct    \
```

The label stack performs the same function as the activation stack for PL/I subroutines. A caller stores the desired return point label on the top of the stack by giving that return point label in the PUSH label stacking action routine. The caller then transfers to the desired subroutine entry point by giving that entry point label in its next-reduction field. The called subroutine returns by using the STACK_POP keyword in the next-reduction field of one or more of its reductions. STACK_POP causes a transfer to the label on top of the label stack as it removes that label from the stack.

The next few paragraphs describe more fully the facilities for writing and calling reduction subroutines. A set of reductions for translating the value language follows this description.

LABEL STACK ACTION ROUTINES

Two action routines manipulate the label stack used by reduction subroutines: PUSH and POP.

PUSH
: pushes the named label onto the top of the stack. Up to 10 labels can be stored in the stack by default.

POP
: pops the top label off the top of the label stack. The label below the popped label becomes the new top of the stack. If the popped label is the only label in the stack, the stack becomes empty. If no labels are on the stack before popping, the POP is ignored.

If a PUSH would cause the label stack to overflow, then PUSH calls the lex_error_ subroutine to report a severity 4 error and then calls the cu_$cl entry point to return to command level. A start command cannot be given, but translator maintenance personnel can perform debugging operations to determine why the stack has overflowed.

By default, only 10 labels can be stored in the stack. This number can be increased by use of the MAX_DEPTH attribute declaration. See "Attribute Declarations" below for more information.

POP is useful for reduction subroutines that are called by stacking two return labels, a normal return label, and an error return label, before transferring to the subroutine. The following example illustrates this usage.

```
attr        / fixed         / LEX attr(1) PUSH(attr)
                              PUSH(syntax_err)              / punct     \
            / float         / LEX attr(2) PUSH(attr)
                              PUSH(syntax_err)              / punct     \
            / <any-token>   / ERROR(5) LEX PUSH(attr)
                              PUSH(syntax_err)              / punct     \
syntax_err
            /               / ERROR(6) POP NEXT_STMT      / stmt      \

punct       / ,             / LEX POP                     / STACK_POP\
            / ;             / LEX POP                     / STACK_POP\
            / <any-token>   /                             / STACK_POP\
```

The label stack is implemented as an array of fixed binary integers. The reductions command converts all labels appearing in a PUSH action routine to a reduction number that is passed as an argument to a PUSH internal procedure provided by the reductions command. The PUSH procedure increments a STACK_DEPTH variable that records the array index of the top stack element, and then stores its input reduction number in the new top-of-label-stack element. POP pops the top label from the stack by decrementing STACK_DEPTH. It is sometimes useful to clear the label stack when an error occurs. This can be done by an action statement that sets STACK_DEPTH to zero.

LABEL STACK NEXT-REDUCTION KEYWORDS

Two keywords can be given in the next reduction field of a reduction to perform reduction subroutine return operations: STACK and STACK_POP.

STACK
: transfers to the label stored on top of the label stack. If the label stack is empty, then no STACK operation occurs, and a transfer occurs to the next reduction (the reduction following the one that used the STACK keyword) just as if an empty next-reduction field had been given.

STACK_POP
: performs a STACK operation followed by a POP operation. This implements the typical subroutine return operation.

SAMPLE REDUCTIONS USING REDUCTION SUBROUTINES

Figure 3-13 below shows the reductions required to translate the value language described in Figure 3-10. The punct reduction subroutine is called to process the list punctuation symbols by the names, attr, and values reduction groups. These groups in turn are reduction subroutines that are called to process statement operands by the stmt group of reductions.

The error messages used below can be summarized as follows: ERROR(1)—severity 2, unrecognized statement; ERROR(2)—severity 2, unexpected '^a' punctuation mark in a name list; ERROR(3)—severity 2, invalid name '^a' in a Name list; ERROR(4)—severity 3, incomplete statement; ERROR(5)—severity 2, invalid attribute '^a' in an Attribute list; ERROR(6)—severity 2, invalid number '^a' in a Value list; and ERROR(7)—severity 3, unexpected '^a' when a punctuation mark was expected in a name list.

```
MAX_DEPTH 2 \
BEGIN
stmt      / Name :          / LEX(2) PUSH(stmt)          / names      \
          / Attribute :     / LEX(2) PUSH(stmt)          / attr       \
          / Value :         / LEX(2) PUSH(stmt)          / values     \
          / <any-token>     / ERROR(1) NEXT_STMT         / stmt       \
          / <no-token>      /                            / RETURN     \

names     / <name>          / set_name LEX PUSH(names)   / punct      \
          / ;               / ERROR(2) LEX               / STACK_POP  \
          / ,               / ERROR(2) LEX               / names      \
          / <any-token>     / ERROR(3) LEX PUSH(names)   / punct      \
          / <no-token>      / ERROR(4)                   / RETURN     \

attr      / fixed           / attr(1) LEX PUSH(attr)     / punct      \
          / float           / attr(2) LEX PUSH(attr)     / punct      \
          / decimal         / attr(3) LEX PUSH(attr)     / punct      \
          / binary          / attr(4) LEX PUSH(attr)     / punct      \
          / ;               / ERROR(2) LEX               / STACK_POP  \
          / ,               / ERROR(2) LEX               / attr       \
          / <any-token>     / ERROR(5) LEX PUSH(attr)    / punct      \
          / <no-token>      / ERROR(4)                   / RETURN     \

values    / <decimal_number>
                            / set_num  LEX PUSH(values)  / punct      \
          / ;               / ERROR(2) LEX               / STACK_POP  \
          / ,               / ERROR(2) LEX               / values     \
          / <any-token>     / ERROR(6) LEX PUSH(values)  / punct      \
          / <no-token>      / ERROR(4)                   / RETURN     \

punct     / ;               / LEX POP                    / STACK_POP  \
          / ,               / LEX                        / STACK_POP  \
          / <any-token>     / ERROR(7) NEXT_STMT POP     / STACK_POP  \
          / <no-token>      / ERROR(4)                   / RETURN     \
```

Figure 3-13. Reductions for the Value Language

ATTRIBUTE DECLARATIONS

Two attribute declarations control the maximum depth of the reduction subroutine label stack and inclusion of rdc-provided internal procedures for use in translator-provided action subroutines. These attribute declarations are described below.

MAX_DEPTH number \
defines number, a decimal integer, as the maximum depth of the reduction subroutine label stack.

INCLUDE action_routine \
causes the reductions command to include an internal procedure that implements the named lexing or error action routine. NEXT_STMT, ERROR, LEX, DELETE, and DELETE_STMT can be given as action_routine values. The action routine internal procedures can then be called by the translator's action routines.

SUMMARY OF THE REDUCTION LANGUAGE

Table 3-5 below summarizes the elements of the reduction language.

Table 3-5. ELEMENTS OF THE REDUCTION LANGUAGE

```
labels      syntax           actions         next reduction
-------     ------------     -----------     --------------

MAX_DEPTH label_stack_depth_number \
INCLUDE NEXT_STMT
\ INCLUDE ERROR \
INCLUDE LEX \
INCLUDE DELETE \
INCLUDE DELETE_STMT \
BEGIN    / absolute_spec  / semant(...) / label      \
         / <relative_fcn> / [var="1"b]  /            \
label
label2  /                /              /            \
/ <no-token>             / LEX          / RETURN     \
/ <any-token>            / LEX(n)       / STACK      \
/ <name> /               / NEXT_STMT    / STACK_POP  \
/ <decimal-integer>
                         / DELETE       /            \
/ <quoted-string>/       / DELETE(n)    /            \
/ <BS>                   / DELETE(m,n)  /            \
/                        / DELETE_STMT  /            \
/                        / ERROR(n)     /            \
/                        / PUSH(label)  /            \
/                        / POP          /            \
```

Name: release, rl

SYNTAX AS A COMMAND

```
rl {-control_arg}
```

FUNCTION

releases the stack history that was automatically preserved after an unclaimed or quit 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 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: release_resource, rlr

SYNTAX AS A COMMAND

```
rlr type STR1...STRN {-control_arg}
```

FUNCTION

releases a resource into the free pool. A resource may only be released by its accounting owner or privileged processes.

ARGUMENTS

type
: is a resource type defined in the resource type description table (RTDT).

STRi
: is the unique identifying name of the particular resource being released. If STR looks like a control argument, precede it with -name (-nm).

CONTROL ARGUMENTS

-priv
: specifies that the you wish to perform a privileged release of this resource from the accounting owner, even though you may not be the accounting owner (see "Access Restrictions.")

ACCESS REQUIRED

The -priv control argument requires execute access to the rcp_admin_ gate.

You must have rew effective access to the resources named.

Name: rename, rn

SYNTAX AS A COMMAND

```
rn path1 name1 {...pathN nameN} {-control_args}
```

FUNCTION

replaces a specified segment, multisegment file (MSF), data management (DM) file, directory, link, or extended entry name by a specified new name, without affecting any other names the entry might have.

ARGUMENTS

paths
: are the pathnames of a segment, multisegment file, data management file, directory, link, or extended entry. This argument can consist of "-name STR" to specify a nonstandard entryname STR which already exists and which begins with a hyphen or contains ASCII control characters or any of the nonstandard characters ", <, >, $, %, ?, *, =, (,), [,], ::.

names
: are additional names to be added. This argument can consist of "-name STR" when the entryname begins with a hyphen. The other nonstandard characters detailed above are not recommended for entrynames and this command will not generate entrynames which contain them.

CONTROL ARGUMENTS

-interpret_as_extended_entry, -inaee
: interpret the selected entry as an extended entry type.

-interpret_as_standard_entry, -inase
: interpret the selected entry as a standard entry type.

ACCESS REQUIRED

You require modify permission on the containing directory.

EXAMPLES

The command line

```
! rn alpha beta >sample_dir>gamma delta
```

renames alpha to beta in your working directory and renames gamma to delta in the directory >sample_dir.

The command line

```
! rn -name *stuff junk
```

renames the segment *stuff to junk in your working directory.

Name: reorder_archive, ra

SYNTAX AS A COMMAND

```
ra {-control_arg} path1 ... {-control_arg} pathN
```

FUNCTION

provides a convenient way of reordering the contents of an archive segment, eliminating the need to extract, order, and replace the entire contents of an archive. This command places designated components at the beginning of the archive, leaving any unspecified components in their original order at the end of the archive. For more information on archives and how they can be sorted, see the archive and archive_sort commands.

ARGUMENTS

pathi
: is the pathname of the archive segment to be reordered. If pathi does not have the archive suffix, one is assumed.

CONTROL ARGUMENTS

-console_input, -ci
: indicates the command is to be driven from terminal input. (Default)

-file_input, -fi
: indicates the command is to be driven from a driving list (see "Notes").

CONTROL ARGUMENTS

-console_input, -ci
indicates the command is to be driven from terminal input. (Default)

-file_input, -fi
indicates the command is to be driven from a driving list (see "Notes").

This page intentionally left blank.

NOTES

When the command is invoked with -console_input or with no control arguments, the message "input for archive_name" is printed, where archive_name is the name of the archive segment to be reordered. Component names are then typed in the order desired, one component name per line. A period (.) on a line by itself terminates input. The two-character line ".*" causes the command to print an asterisk (*). This feature can be used to make sure there are no typing errors before typing a period. The two-character line ".q" causes the command to terminate without reordering the archive.

When reorder_archive is invoked with -file_input, it reads a driving list to determine the order of components. That list resides in the working directory with the name "name.order" (where "name.archive" is the name of the archive segment to be reordered) and it consists of a list of component names in the order desired, one component name per line. No period is necessary to terminate the list. Any errors in the list (name not found in the archive segment, name duplication) cause the command to terminate without altering the archive. A temporary segment named "ra_temp_.archive" is created in the user's process directory, which is created once per process and is truncated after it is copied into the directory supplied by pathi. If the command cannot copy the temporary segment, it attempts to save it in the process directory by renaming it with the name of the archive.

This command does not operate on archive segments containing more than 1000 components.

Name: repeat_line, rpl

SYNTAX AS A COMMAND

```
rpl {N} {string}
```

FUNCTION

allows certain limited testing of the performance of your interactive terminal by "echoing" an arbitrary message you typed in.

ARGUMENTS

N
: is the number of times the message is to be printed. If you don't give N or if N is 0, its previous value is used; the default first-time value is 10.

string
: is the arbitrary message you typed in to be printed. Quote it if it contains blanks.

NOTES

The first time you use repeat_line in a process, a canned message, consisting of "The quick brown fox..." (alternate words in red and black shift), followed by three separate lines, each containing one horizontal tab character plus ASCII graphics in ascending numeric sequence, is used. If you don't supply string, you are requested to type in a new string. Once the message to be printed has been determined, it is printed N times. (In the case of "The quick brown fox" message, 4N lines are printed.) If string is an asterisk, the previous message is reused.

When printing of the message is completed or when you don't give an initial message, the line

```
Type line (or q or <NL>):
```

is printed. Typing only the newline (<NL>) character prints the previous message another N times. The lowercase letter q followed by <NL> returns repeat_line to its caller. Any other line is interpreted as a new message to be printed N times.

Name: repeat_query, rq

SYNTAX AS A COMMAND

```
rq
```

FUNCTION

repeats the last query (by command_query_, described in the Subroutines and I/O Modules manual) if it has not yet been answered.

NOTES

The repeat_query command is useful for reinterpreting questions (asked by other commands) that are garbled.

If no question has been asked, or if the latest question was answered, the error message "No pending query." is printed.

This command does not completely restore the environment in effect at the time of the original query. For example, nonstandard attachments of I/O switches are not restored.

EXAMPLES

Suppose that the system starts to print a question while you are typing. The query looks like

```
E@foo.pl1?
```

The user signals QUIT and invokes the repeat_query command. The system prints

```
Do you want to delete the old segment foo.pl1?
```

The user answers and continues.

Alternatively, you can use the command escape (..) to issue repeat_query.

```
E@foo.pl1?  ! ..rq
```

The system responds with

```
Do you want to delete the old segment foo.pl1?
```

The user then answers and continues.

Another use is to return to a query after interrupting a command line issued within the query

```
   Do you want to delete the old segment foo.pl1?

!  ..print foo.pl1 1
   foo: proc;
   (user signals QUIT)

!  rq
   Do you want to delete the old segment foo.pl1?  ! yes
```

If there is more than one suspended command in your stack, the stack is searched for the program that asked a question, and any intervening programs are released.

Name: reprint_error, re

SYNTAX AS A COMMAND

```
re {-control_args}
```

FUNCTION

makes the system condition handler 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. Make -depth appear only once per command line. (Default: 1)

-long, -lg
prints the long form of the message.

NOTES

If you select no control argument, the default selects less extensive information than -long.

The message mode options for reprint_error have no effect on the operation.

Name: reserve_resource, rsr

SYNTAX AS A COMMAND

```
rsr -control_arg
```

FUNCTION

reserves a resource or group of resources for use by the calling process. The reservation takes effect immediately and lasts until canceled by cancel_resource or by process termination. (See Section 5 of the Programmer's Reference Manual for more information on resource reservation.)

CONTROL ARGUMENTS

-resource STR, -rsc STR
specifies a description of the resources to be reserved. If the description contains spaces or special characters, enclose it in quotes. (See "Notes on Resource Description.")

NOTES ON RESOURCE DESCRIPTION

A resource description describes certain devices and volumes by name or by attributes and an optional number. It has the following format:

```
{-resource_type} resource_spec1...{-resource_type resource_specN}
```

which is a series of at least one resource_spec where all but the first must be preceded by -resource_type (-rsct).

The format of a resource_spec can be any of the following:

```
volume_type name1 {names}

device_type {names}

device_type {-control_args}
```

where:

volume_type
can be either tape_vol or disk_vol. You must supply at least one name with volume_type, and it is the name of the volume, for example, 050102.

device_type
can be either tape_drive or disk_drive; "names" are the names of devices such as tape_01; if you select no names, you can choose from these control arguments:

-attributes STR, -attr STR
is a list where STR consists of a string of attributes with values separated by commas with no spaces. For tape drives the attributes are:

```
mode=
track=
den=
speed=
```

For disk drives the only attribute is model=.

Use list_resource_types to find suitable values for these attributes.

-number N, -nb N
is the number of identical resources of the type desired.

EXAMPLES

```
! rsr -rsc "tape_vol 50102 u-309 -rsct tape_drive -attr
    track=9,den=800 -nb 2"
```

This command line reserves four resources: two tapes, 050102 and u-309, and two tape drives that are 9 track and capable of 800 bpi operation.

Name: reset_external_variables, rev

SYNTAX AS A COMMAND

```
rev names {-control_arg}
```

FUNCTION

reinitializes system-managed variables to the values they had when they were allocated.

ARGUMENTS

names
are the names of the external variables, separated by spaces, to be reinitialized.

CONTROL ARGUMENTS

-unlabeled_common, -uc
indicates unlabeled (or block) common.

NOTES

A variable cannot be reset if the segment containing the initialization information is terminated after the variable is allocated.

Name: reset_ips_mask

SYNTAX AS A COMMAND

```
reset_ips_mask {signal_names} {-control_args}
```

FUNCTION

sets the IPS mask for the current process to unmask some or all IPS signals.

ARGUMENTS

signal_names
are the names of one or more IPS signals to be unmasked. The signal names must be defined in sys_info$ips_mask_data. Presently the defined signal names are quit, alrm, neti, cput, trm_, sus_, wkp_, pgt_, system_shutdown_scheduled_, and dm_shutdown_scheduled_. Supply either signal_names or -all.

CONTROL ARGUMENTS

-all, -a
sets the IPS mask to unmask all IPS signals.

-brief, -bf
suppresses printing of the previous state of the IPS mask after setting it.

-long, -lg
prints the previous state of the IPS mask after setting it. (Default)

NOTES

If all undefined IPS signals are either masked or unmasked and you give -long, they are not mentioned; if, however, some are masked and others are not, an octal list is printed. This can only happen when you specify an invalid (probably uninitialized) value in a call to set that mask.

Name: resolve_linkage_error, rle

SYNTAX AS A COMMAND

```
rle virtual_pointer
```

FUNCTION

satisfies the linkage fault taken when a process encounters a linkage error by locating the virtual pointer specified as an argument and patching the linkage information of the process. When you issue the start command, the process continues as if the original linkage fault had located the specified virtual pointer.

ARGUMENTS

virtual_pointer
is a virtual pointer specifier (see Section 1 for a description of virtual pointers).

NOTES

The program locates the virtual pointer specified as an argument and patches the linkage information of the process so that when the start command is issued the process continues as if the original linkage fault had located the specified virtual pointer.

EXAMPLES

This example is typical. The program is running and a linkage error is encountered. The command is issued, correcting the linkage error and allowing the program to continue.

```
! myprog
  Error: Linkage error by >udd>m>vv>myprog|123
  referencing subroutine$entry
  Segment not found.
  r 1234 2.834 123.673 980 level 2, 26

! rle mysub$mysub_entry
  r 1234 0.802 23.441 75 level 2, 26

! start
  <myprog is running>
```

Name: resource_status, rst

SYNTAX AS A COMMAND

```
rst type STR1...STRN {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[rst type name -control_arg]
```

FUNCTION

prints selected information about the status of a given resource. As an active function, returns the value requested by the specified control argument.

ARGUMENTS

type
: is a resource type defined in the resource type description table (RTDT). (For more information and for a list of the resource types on the system, use the list_resources and list_resource_types commands.)

STRi
is the unique identifying name of the particular resource desired. If STR looks like a control argument, precede it by -name.

CONTROL ARGUMENTS

-access_class, -acc
prints the AIM access class or the access class range of the resource.

-acs_path
prints the pathname of the ACS for this resource.

-all, -a
prints all information maintained about this resource. It is not allowed in the active function.

-alloc
prints the state of your allocation switch for this resource.

-attributes, -attr
prints the current and protected attributes of this resource.

-charge_type, -crgtp
prints the charge type for this resource.

-comment, -com
prints the user-settable comment associated with this resource.

-location, -loc
prints the location field associated with this resource.

-lock
prints the status of the resource lock for this resource. In the active function, returns "true" if the lock is on, "false" if it is off.

-mode, -md
prints your effective mode to the resource.

-owner, -ow
prints the name of the owner of the resource.

-potential_access_class, -pacc
prints the potential access class or potential access class range for this resource.

-potential_attributes, -pattr
prints the potential attributes of this resource.

-priv
returns your privileged effective access to the resource (see "Access Restrictions" below).

-release_lock, -rll
prints the status of the lock that prevents the owner from releasing this resource. In the active function, returns "true" if the lock prevents the owner from releasing the resource, "false" otherwise.

-uid
prints the unique identifier of this resource.

ACCESS REQUIRED

You need execute access to the rcp_admin_ gate to use -priv.

Name: resource_usage, ru

SYNTAX AS A COMMAND

```
ru {-control_arg}
```

FUNCTION

prints a month-to-date report of your resource consumption.

CONTROL ARGUMENTS
are used to select portions of the available resource usage information. You can give only one of the following:

-brief, -bf
prints a header describing the resource usage reporting period, followed by the month-to-date dollar charge, the monthly spending dollar limit, and three dollar-totals figures giving your interactive, absentee, and I/O daemon usage.

-long, -lg
prints the most comprehensive picture of your resource usage. In addition to including the information selected by -brief, it gives an expanded report of interactive, absentee, I/O daemon, and device usage.

For interactive usage, the dollar charge is broken down according to shift, monthly dollar limit per shift, charged virtual CPU time, charged terminal connect time, and charged memory units expressed in thousands. Absentee usage is presented giving usage per queue: number of dprint/dpunch requested pieces, charged virtual CPU time, and charged lines of printed or punched output.

The device usage category includes charges for tape (time spent with a drive assigned), tape mounts, disk (time spent with a disk drive assigned), disk mounts, and logical volumes (time spent with a private logical volume attached). In addition, a site can define devices corresponding to the various lines (tty or network) by which the system is accessed and set prices for their usage.

-total, -tt
prints your dollar-totals figures for the month-to-date dollar charge, the monthly spending dollar limit, and the absolute total spending.

If the project administrator has set an absolute dollar limit for you (which is independent of the monthly spending limit), it is printed, along with the date on which the limit was last reset and the limit's reset interval. The absolute total spending is the dollar charge against this absolute limit. In cases where no absolute limit has been set, the absolute total spending represents charges running from your registration date.

NOTES

If you supply no control argument, the default selects slightly less resource usage information than that printed by -long.

The system calculates your month-to-date dollar charges when it creates your process. Issue the new_proc command prior to typing resource_usage if you wish the most updated figures.

In a given usage report, shift and queue numbers may not appear in consecutive order because only shifts or queues with accrued charges are listed.

If no dollar limit stop has been set by your project administrator, the resource usage report prints "open" as the dollar limit entry.

You can't invoke resource_usage to obtain information about another's resource consumption.

Name: response

SYNTAX AS A COMMAND

```
response arg {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

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

FUNCTION

asks you a question and returns the answer you typed. The answer is not returned in quotes; the command processor therefore treats the answer as several strings if it contains spaces. You can use the command language ||[feature if you want the command processor to treat the returned string as a single argument. (See also the query active function.)

ARGUMENTS

arg
is the question to be asked. If arg contains spaces or other command language characters, enclose it in quotes.

CONTROL ARGUMENTS

-accept STRs
where STRs are the only responses accepted from you. If a STR contains spaces or other command language characters, enclose it in quotes. If you respond to the question with an answer that is not one of the specified STRs, the active function prints a message explaining that your answer is unacceptable, lists the acceptable answers, and repeats the question.

-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 that ability.

-input_switch STR, -isw STR
specifies the I/O switch to use for input of your response. (Default: user_input)

-long, -lg
adds a leading newline and three trailing spaces to the question. (Default)

-non_null
indicates that you must give a response. If you reply with an empty (blank) line, the active function prints a message explaining that a null response is not allowed and repeats the question.

-output_switch STR, -osw STR
specifies the I/O switch to use for output of the question to you. (Default: user_output)

-repeat DT, -rp DT
repeats the question every DT if you have not responded (see Section 1 for a description of valid DT values).

NOTES

You can use the format_line active function 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 can define the default to be either "enable" or "disable." (See the command_query_ subroutine for details.)

EXAMPLES

Assume that dpc is an abbreviation for

```
do "dp -cp [response ""Number of copies?     ""] &f1"
```

then the following interaction:

```
! dpc report_1.runout memo_phone.runout
  Number of copies?    ! 2
```

gets you two copies of each runout segment.

Assume that the exec_com segment named x.ec contains the following line:

```
dp -he [response "What header?" -non_null] -cp 2 report.print
```

then the following interaction:

```
! ec x
  What header?    ! <carriage return>
  response:  Null response not allowed, please retype.

  What header?    ! Aulin
```

prints two copies of report.print with the header Aulin. Use of -non_null ensures that a header follows the -header control argument to the enter_output_request (eor) command; otherwise, the -copy control argument to the eor command would be interpreted as the header and the number 2 as the segment name.

Assume the k.ec exec_com segment contains the following line:

```
dp -rqt [response "Which rqt?" -accept printer unlined remote] resume
```

then the following interaction:

```
! ec k
  Which rqt?  ! plotter
  response:  'plotter' is not an acceptable answer.
  Acceptable answers are:
        'printer'
        'unlined'
        'remote'

  Which rqt?  ! printer
```

enters a printer request for one copy of the resume segment.

The following illustrates the usage of format_line to insert other active function values into the response:

```
[response [format_line "Enter date (default is ^a)" [date]]]
```

Name: reverse, rv

SYNTAX AS A COMMAND

```
rv STR
```

SYNTAX AS AN ACTIVE FUNCTION

```
[rv STR]
```

FUNCTION

returns the characters of a specified string in reverse order.

EXAMPLES

```
! string [rv abcdef]
  fedcba
```

Name: reverse_after, rvaf

SYNTAX AS A COMMAND

```
rvaf STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[rvaf STRA STRB]
```

FUNCTION

performs the same function as the after command/active function, but in reverse order.

NOTES

The active function returns that part of STRA following the last occurrence of STRB in STRA (after uses the first occurrence). If STRB occurs last in STRA or does not occur at all, a null string is returned.

```
[reverse_after STRA STRB]
```

is the same as:

```
[reverse [before [reverse STRA] [reverse STRB]]]
```

when STRB appears in STRA. It is a null string when STRB does not appear in STRA.

EXAMPLES

```
string [reverse_after abcdef123def456 def]
456
string [rvaf acebdf g]

string XY[rvaf 17.245et17 17]ZZ
XYZZ
```

Name: reverse_before, rvbe

SYNTAX AS A COMMAND

```
rvbe STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[rvbe STRA STRB]
```

FUNCTION

performs the same function as the before command/active function, but in reverse order.

NOTES

The active function returns that part of STRA preceding the last occurrence of STRB in STRA (before uses the first occurrence). If STRB occurs first and nowhere else in STRA, a null string is returned. If STRB does not occur in STRA, STRA is returned.

```
[reverse_before STRA STRB]
```

is the same as:

```
[reverse [after [reverse STRA] [reverse STRB]]]
```

when STRB appears in STRA. It is the same as STRA when STRB does not appear in STRA.

EXAMPLES

```
string [reverse_before abcdef123def456 def]
abcdef123
string [rvbe acebdf g]
acebdf
string XY[rvbe 17.245e+17 17]ZZ
XY17.245e+ZZ
```

Name: reverse_decat, rvdecat

SYNTAX AS A COMMAND

```
rvdecat STRA STRB C
```

SYNTAX AS AN ACTIVE FUNCTION

```
[rvdecat STRA STRB C]
```

FUNCTION

performs the same function as the decat command/active function, but in reverse order.

NOTES

The active function returns the "decatenation" of STRA with respect to the last occurrence of STRB in STRA (decat uses the first occurrence). The value for C is any three-digit bit string expressed as 0 or as 1 characters such as 000,001,...,111. The last occurrence of STRB found in STRA divides STRA into three parts: the part prior to STRB, the part matching STRB, and the part following STRB. Digits of C correspond to these three parts. The return string contains the parts of STRA whose corresponding bit in C is 1. The parts are returned in their original order of appearance in STRA.

```
[reverse_decat STRA STRB C]
```

is the same as:

```
[reverse [decat [reverse STRA] [reverse STRB] [reverse C]]]
```

when STRB appears in STRA. It is also the same as:

```
[decat STRA STRB C]
```

when STRB does not appear in STRA.

EXAMPLES

```
string [rvdecat abcdef123defghi def 110]
abcdef123def
string [rvdecat abcdef g 100]
abcdef
```

Name: reverse_index, rvindex

SYNTAX AS A COMMAND

```
rvindex STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[rvindex STRA STRB]
```

FUNCTION

performs the same function as the index command/active function, but in reverse order.

NOTES

The active function returns the index (character position) in STRA of the beginning of the last occurrence of STRB (index uses the first occurrence). If STRB does not appear in STRA, 0 is returned.

```
[reverse_index STRA STRB]
```

is the same as:

```
[length STRA] - [index [reverse STRA] [reverse STRB]] + 2 - [length STRB]
```

when STRB appears in STRA. It is 0 when STRB does not appear in STRA.

EXAMPLES

```
string [rvindex abc123defghi123jkl 123]
13
string [rvindex "Now is the time." hte]
0
string [rvindex abcdefghi ef]
5
```

Name: reverse_search, rvsrh

SYNTAX AS A COMMAND

```
rvsrh STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
rvsrh STRA STRB
```

FUNCTION

performs the same function as the search command/active function, but in reverse order.

NOTES

The active function returns the index (character position) of the last character in STRA that appears in STRB (search returns the first such character). If no characters of STRA appear in STRB, 0 is returned.

```
[reverse_search STRA STRB]
```

is the same as

```
[length STRA] - [search [reverse STRA] STRB] + 1
```

when a character of STRB appears in STRA. It is 0 when a character of STRB does not appear in STRA.

EXAMPLES

```
string [rvsrh "abc = 213" 0123456789]
9
string [rvsrh "abc = def" 0123456789]
0
```

Name: reverse_verify, rvverify

SYNTAX AS A COMMAND

```
rvverify STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[rvverify STRA STRB]
```

FUNCTION

performs the same function as the verify active function, but in reverse order.

NOTES

The active function returns the index (character position) of the last character in STRA that does not appear in STRB (verify returns the first such character). If all characters of STRA appear in STRB, 0 is returned.

```
[reverse_verify STRA STRB]
```

is the same as

```
[length STRA] - [verify [reverse STRA] STRB] + 1
```

when the characters of STRA do not appear in STRB. It is 0 when all characters of STRA appear in STRB.

EXAMPLES

```
string [rvverify "abc = 123" 0123456789]
6
string [rvverify "abc = def" 0123456789]
9
string [rvverify 21435 0123456789]
0
```

This page intentionally left blank.

Name: reverse_substr, rvsubstr

SYNTAX AS A COMMAND

```
rvsubstr STR J {N}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[rvsubstr STR J {N}]
```

FUNCTION

performs the same function as the substr command/active function, but counts characters from right to left. The returned string has its characters in the same order as the input string.

NOTES

The active function returns that portion of STR starting with the character in position J (the characters in the string being numbered from right to left starting with 1) and continuing for N characters, where J and N are decimal integers; J must be greater than 0 and N must be greater than or equal to 0. If you omit N, the remainder of STR is returned. If J is greater than the length of STR, the null string is returned; if N is greater than the remainder of STR, the remainder is returned.

The string

```
[rvsubstr STR J {N}]
```

is the same as

```
[reverse [substr [reverse STR] J {N}]]
```

EXAMPLES

```
rvsubstr abcdefg 2 3
def

rvsubstr frobozz 4
frob

rvsubstr spatula 5 4
spa
```

Name: revert_output, ro

SYNTAX AS A COMMAND

```
ro {-control_args}
```

FUNCTION

reverts the effect of the file_output, syn_output, and terminal_output commands, i.e., releases the most recent preceding command.

CONTROL ARGUMENTS

-all, -a
reverts all file_output, syn_output, and terminal_output attachments for specified I/O switches or for all switches if you specify none.

-source_switch STR, -ssw STR
specifies the name of an I/O switch to be redirected. (Default: user_output)

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.

EXAMPLES

The command line

```
! revert_output -ssw STR
```

reverts the latest attachment by one of the following command lines:

```
file_output -ssw STR
syn_output target -ssw STR
terminal_output -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.

See the file_output, syn_output, and terminal_output commands.

Name: rtrim

SYNTAX AS A COMMAND

```
rtrim STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[rtrim STRA STRB]
```

FUNCTION

returns a character string trimmed of specified characters from the right.

NOTES

The rtrim active function finds the last character of STRA not in STRB, trims characters from STRA following this character, and returns the trimmed result. Space characters are trimmed if STRB is omitted.

EXAMPLES

```
! string [rtrim 000305.000 0]
  000305.
! string [rtrim [ltrim 000305.000 0] 0]
  305.
! string X[rtrim "   This is it.   "]Y
  X   This is it.Y
```

Name: run

SYNTAX AS A COMMAND

```
run {-control_args} {program} {program_args}
```

FUNCTION

provides the user with a temporary, somewhat isolated, environment for the execution of programs.

ARGUMENTS

program
: is the reference name or pathname of the main program for the run unit.

program_args
are the arguments passed to the main program or the exec_com specified by -exec_com. See "Notes on the exec_com Feature" below.

CONTROL ARGUMENTS

-copy_reference_names, -crn
begins the run unit with a copy of the reference names initiated prior to the run unit. See "Notes on Reference Name Control Arguments" below.

-exec_com path, -ec path
executes the exec_com path after the environment of the run unit is established.

-limit N, -li N
interrupts the run unit every N seconds of virtual CPU time to ask the user "Time limit reached. Do you want to continue the program?"

-new_reference_names, -nrn
begins the run unit without any reference names known. The reference name control arguments are explained further in "Notes on Reference Name Control Arguments". This is the default.

-no_exec_com, -nec
always invokes the main program directly.

-old_reference_names, -orn
uses the same reference names inside and outside the run unit. See "Notes on Reference Name Control Arguments" below.

NOTES

The -new_reference_names, -copy_reference_names, and -old_reference_names control arguments are mutually incompatible.

NOTES ON FUNCTION

The run command is primarily intended to aid users in executing programs written on systems that have a different definition of program than the Multics system has. Within the Multics system, a process is a program. Although many separately compiled "programs" can be executed in a process, they all share such program environment items as FORTRAN common blocks, PL/I external static variables, reference names, and file openings. This can cause problems for users with programs that depend on, for example, FORTRAN common being reinitialized each time the program is executed. However, within the Multics system a run unit is also a program. The run command executes the specified program in a temporary program environment that is separate from the rest of the process and from any other run unit.

The program attributes that are managed (restored, reset, etc.) by the run command are:

1) PL/I internal static storage.

2) PL/I external static variables whose names do not contain "$".

3) FORTRAN common blocks whose names do not contain "$".

4) allocations of PL/I based and controlled storage when the program does not specify a particular area.

5) files used only through programming language I/O constructs.

6) reference names and search rules.

All of these are restored to their prior state when the run unit terminates. An optional feature, specified by the -old_reference_name control argument, does not restore search rules or reference names.

NOTES ON REFERENCE NAME CONTROL ARGUMENTS

The three reference name control arguments affect the management of the address space (the segments known to the process). Reference names are the names by which the known segments are initiated. The default action during environment restoration is to terminate all segments made known (initiated for the first time) inside the run unit. When this happens, all the reference names used inside the run unit are discarded and the names in use prior to the run unit are restored.

The -nrn control argument causes segments to be terminated and reference names and search rules to be restored at the end of the run unit. The run unit begins without any reference names. This option should be chosen when the user wants to be sure that the program is calling all the right subroutines and not inadvertantly getting some subroutine that happened to have the same name but was used earlier in the process.

The -crn control argument also causes segments to be terminated and reference names and search rules to be restored at the end of the run unit. The run unit begins with the reference names already initiated. This option is suitable when the user does not expect name conflicts with already known segments.

The -orn control argument does not terminate segments and does not manipulate reference names at the end of the run unit. It is generally safer. It is necessary in certain situations, such as when files are opened before the run unit begins.

NOTES ON THE EXEC_COM FEATURE

The run command uses an exec_com segment if the -exec_com control argument is specified or if -no_exec_com is not specified and the segment program_name.run.ec is found in the same directory as the main program. Otherwise, the specified main program is invoked directly.

If an exec_com segment is used, all command arguments after the run control arguments are passed to the exec_com.

If an exec_com is not used, the first non-control argument is interpreted as the name of the main program for the run.

NOTES ON SEARCH RULES

The search rules in effect at the beginning of the run unit are always the same as those used just before the run unit. In order to get the default system search rules, an exec_com must be used that invokes the set_search_rules command with no arguments before executing the main program.

If -old_reference_names (-orn) is specified, any changes to the search rules remain when the run unit ends. Otherwise, the search rules are restored to the values they had at the beginning of the run.

NOTES ON TERMINATING RUN UNIT

There are several ways to terminate a run unit.

1) return from the main program or exec_com invoked by the run command.

2) execute a stop statement in PL/I or FORTRAN.

3) invoke the stop_run command. This can be done from a program, an exec_com, or at interactive command level.

4) invoke the release command.

Either executing a stop statement or calling stop_run causes the finish condition to be signaled. User code can optionally be called during run unit termination. Refer to the description of add_epilogue_handler_ in the Subroutines manual.

NOTES ON RUN UNIT LIMITATIONS

Run units incur significant overhead costs. Run units should be used primarily for debugging and executing user-written programs.

Any files attached/opened via iox_ or io_call during the run unit must be explicitly detached/closed before the run unit terminates.

If any files used in the run unit are opened before the run unit begins, -orn must be used.

Run units are not recursive.

The trace and change_error_mode commands should not be used inside run units.

The answer command's command line cannot include the run command.

The profile command cannot be used on an object segment that was executed inside a run unit unless the object segment's per-process static switch was turned on. (See "Notes to Subsystem Writers" below.)

The list_external_variables, delete_external_variables, and reset_external_variables commands do not handle any PL/I external static variables or FORTRAN common blocks in the prerun unit environment that object segments with per-process static use.

NOTES TO SUBSYSTEM WRITERS

If a procedure's internal static storage and linkage section are to be left alone during the run unit, the object segment must be given the perprocess_static attribute. This can be done by including the global keyword "Perprocess_Static;" in the bindfile (with no parameters) for bound segments, or by including the perprocess_static pseudo-operation in alm procedures, or by typing:

```
! switch_on perprocess_static_switch segname
```

See the description of the switch_on command.

Object segments without per-process static that are used both inside and outside the run units should not have internal static pointers to named temporary external segments. Each execution of such an object segment in a run unit destroys the previous contents of the segment. Instead, the internal static pointer should point to a segment managed by the Multics temporary segment facility (see get_temp_segments_ in the Subroutines manual). If using temporary segments is inappropriate because the information must be cumulative, the object segment must have per-process static.

The run command sets up a "condition wall" so that procedures before the run command on the stack do not get control (as the result of a signaled condition) until the run unit is terminated.

EXAMPLES

The following command line shows how to run the program prog2 with -old_reference_names:

```
run -orn prog2 prog2_arg1
```

The following exec_com uses the default search rules and invokes a program whose arguments were given to the run command:

```
&command_line off
set_search_rules
prog2 &f1
&quit
```

The run command line that uses this exec_com might be:

```
run prog2_ec prog2_arg1
```

The example shown above is useful when the user wants to call a library subroutine and not a private subroutine that has the same name and that was already used in the process. The example should not be used with -orn because the search rules would be changed permanently, and the main purpose of the example is to avoid using existing reference names.

The following example shows how to invoke an arbitrary command from an exec_com within the environment of a run unit:

```
&command_line off
&f1
&quit
```

Name: run_cobol, rc

SYNTAX AS A COMMAND

```
rc name {-control_args}
```

FUNCTION

This command is not needed to execute COBOL object programs on Multics; it is used to simulate an environment in which traditional COBOL concepts can be defined easily. This command cannot be called recursively.

ARGUMENTS

name
is the reference name or pathname of the "main program" in which execution is to be initiated. If a pathname is specified, the specified segment is initiated with a reference name identical to the entryname portion of the pathname. Otherwise, the search rules are used to locate the segment. If the name specified in the PROG-ID statement of the COBOL program (i.e., the entry point name) is different from the current reference name of the object segment, then the name specified here must be in the form A$B where A is the pathname or reference name of the segment and B is the PROG-ID as defined in the IDENTIFICATION DIVISION of the source program.

CONTROL ARGUMENTS

-cobol_switch N, -cs N
sets one or more of the eight COBOL-defined "external switches" on, where N is a number from 1 to 8 (or a series of numbers separated by spaces) that corresponds to the numbered external switch. At the outset of the run unit, the default setting of these external switches is off. (The eight external switches are defined in the *Multics COBOL Reference Manual*, Order No. AS44.)

-no_stop_run, -nsr
avoids establishment of a handler for the stop_run condition. (See "Notes" below.)

-sort_dir path, -sd path
specifies the directory to be used during execution of this run unit for temporary sort work files. If this control argument is not specified, the process directory is assumed.

-sort_file_size N, -sfs N
is the floating point representation of the estimated average size in characters of the files to be sorted during execution of this run unit. This information is used to optimize sorting. If not specified, 1e6 is assumed (i.e., one million characters).

NOTES

This command enables the user to explicitly define and start execution of a COBOL run unit. A run unit is either explicitly started by the execution of the run_cobol command or implicitly started by the execution of a COBOL object program either by invocation from command level or from a call by another program written in COBOL or another language. A run unit is stopped either by the execution of the STOP RUN statement in a COBOL object program or by invocation of the stop_cobol_run command. For the duration of time after a run unit is started and before it is stopped, it is said to be active. All COBOL programs executed while a run unit is active are considered part of that run unit.

A run unit is a subset of a Multics process; it is stopped when the process is ended. Also, when all programs contained in a run unit are cancelled, the run unit is stopped (refer to the cancel_cobol_program command). Only one run unit can be active at any given time in a process. Therefore, the run_cobol command cannot be invoked recursively. Additionally, if a run unit has been started implicitly (as described above), the run_cobol command cannot be used until that run unit has been stopped; i.e., the run_cobol command does not terminate a currently active run unit.

The explicit creation of a run unit with the run_cobol command performs the following functions:

1. Establishment of a "main program", i.e., a program from which control does not return to the caller. The EXIT PROGRAM statements, when encountered in such a program, have no effect, as required in the COBOL definition. An implicitly started run unit has no "main program". The EXIT PROGRAM statement in all programs contained in such a run unit always causes control to be returned to the caller, even if the caller is a system program, e.g., the command processor.

2. Setting of the COBOL external switches. These switches are set to off unless otherwise specified by the -cobol_switch control argument.

3. User control of the action taken when a STOP RUN statement is executed in a COBOL object program. The action normally taken for STOP RUN is cancellation of all programs in the run unit, closing any files left open. After this has been done, the data associated with any of the programs is no longer available. Thus in a debugging environment, it may be useful to redefine the action taken for STOP RUN. When the run unit is explicitly initiated with the run_cobol command, the STOP RUN statement causes the signalling of the stop_run condition for which a handler is established that performs the normal action described above. If the -no_stop_run control argument is specified, this handler is not established, thus allowing the user to handle the signal using other Multics commands. If the user has not explicitly provided a handler for stop_run and specifies the -no_stop_run control argument, an unclaimed signal results.

The name specified in the run_cobol command line need not be a COBOL object program. It can be a program produced by any language compiler that provides a meaningful interface with COBOL programs (e.g., PL/I, FORTRAN).

Refer to the following related commands:

display_cobol_run_unit, dcr
stop_cobol_run, scr
cancel_cobol_program, ccp

Name: runoff, rf

SYNTAX AS A COMMAND

```
rf paths {-control_args}
```

FUNCTION

is used to type out text segments in manuscript form.

ARGUMENTS

paths
are the pathnames of input segments or multisegment files. The runoff suffix is assumed if not supplied. If two or more pathnames are specified, they are treated as if runoff had been invoked separately for each one, in the order in which they occur in the command line.

CONTROL ARGUMENTS
can be intermixed arbitrarily with paths.

-character, -ch
flags certain key characters in the output by putting the line containing the key character in a segment named entryname.chars. The normal output is not affected. Page and line numbers referring to the normal output appear with each flagged line, and reminder characters, enclosed by color-shift characters, are substituted for the key characters. The default set of key and reminder characters corresponds to those unavailable with a 963 typeball, as follows:

Key	*Reminder*
left square bracket	<
right square bracket	>
left brace	(
right brace	)
tilde	t
grave accent	'

The key and reminder characters can be changed by use of the .ch control line; specifying a blank reminder character removes the associated key character from the set of key characters. If a key character would print normally in the output, it should also appear in a .tr control line to turn it into a blank in the output.

-from N, -fm N
starts printing at the page numbered N. If the -page control argument is used, printing starts at the renumbered page N.

-hyphenate, -hph
When this control argument is used, a procedure named hyphenate_word_, that the user supplies, is invoked to perform hyphenation when the next word to be output does not fit in the space remaining in a line (see "Hyphenation Procedure Calling Sequence" at the end of this description). Otherwise, no attempt is made to hyphenate words.

-indent N, -in N
indents output N spaces from the left margin (default indentation is 0 except for "-device 202," which is the default for -segment and has a default indentation of 20; see also -number below). This space is in addition to whatever indentation is established by use of the .in control word.

-no_pagination, -npgn
suppresses page breaks in the output.

-number, -nb
prints source line numbers in the left margin of the output; minimum indentation of 10 is forced.

-page N, -pg N
changes the initial page number to N. All subsequent pages are similarly renumbered. If the control line .pa is used within the segment, the -page control argument is overridden and the page is numbered according to the .pa control line.

-parameter arg, -pm arg
assigns the argument arg as a string to the internal variable "Parameter".

-pass N
processes the source segments N times to permit proper evaluation of expressions containing symbols that are defined at a subsequent point in the input. No output is produced until the last pass.

-segment, -sm
directs output to the segment or multisegment file named entryname.runout. This control argument assumes by default that the material is to be dprinted, so the segment is prepared compatible with device 202 unless another device is specified; thus, unless overridden by the -indent control argument, each printed line in the output segment is preceded by 20 leading spaces so that the text is approximately centered on the page when dprinted.

-stop, -sp
waits for a carriage return from the user before beginning typing and after each page of output (including after the last page of output).

-to N
ends printing after the page numbered N.

-wait, -wt
waits for a carriage return from the user before starting output, but not between pages.

NOTES

Output lines are built from the left margin by adding text words until no more words fit on the line; the line is then justified by inserting extra blanks to make an even right margin. Up to 20 lines each of headers and footers can be printed on each page. The pages can be numbered, lines can be centered, and equations can be formatted. Space can be allowed for diagrams. Detailed control over margins, spacing, headers, justification, numbering, and other aspects of format is provided by control lines that begin with a period. Although the control lines are interspersed within the text, they do not appear in the output segment. The output can be printed page by page to allow positioning of paper, or it can be directed into a segment. Characters not available on the device to which output is directed are replaced by blanks. If special symbols must be hand drawn, a separate segment can be created that indicates where each symbol should be placed. The user can define variables and cause expressions to be evaluated; he also has the ability to refer to (and sometimes modify) variables connected with the workings of the runoff command.

A runoff input segment contains two types of lines: control lines and text lines. A control line begins with a period; all other lines are considered text lines. A two-character control word appears in the second and third character positions of each control line. The control word can take a parameter that is separated from the control word by one or more spaces. Lines that are entirely blank are treated as if they contained a .sp 1 control line.

Text lines contain the material to be printed. If an input line is too short or too long to fill an output line, material is taken from or deferred to the next text line. A line beginning with a space is interpreted as a break in the text (e.g., the beginning of a new paragraph) and the previous line is printed as is.

Tab characters (ASCII HT) encountered in the input stream are converted to the number of spaces required to get to the next tab position (11, 21, ...). Nonprinting control characters in the input segment are discarded in the output segment. The .tr control word can be used to print these control characters in the output segment.

When an input text line ends with any of the characters ".", "?", "!", ";", or "", or with ".", "?", or "!" followed by a double quote or ")", two blanks precede the following word (if it is placed on the same output line), instead of the normal single blank.

The translate character (!) is used both to "count" spaces and to prevent an unattractive line split. See the sample runoff segment at the end of this description.

The maximum number of characters per input or output line is 361; this permits 120 underlined characters plus the newline character.

TERMINOLOGY

The following paragraphs describe various terms that are used throughout the runoff description.

FILL AND ADJUST MODES

Two separate concepts are relevant to understanding how runoff formats output: fill mode and adjust mode. In fill mode, text is moved from line to line when the input either exceeds or cannot fill an output line. Adjust mode right justifies the text by inserting extra spaces in the output line, with successive lines being padded alternately from the right and from the left. Initial spaces on a line are not subject to adjustment. Fill mode can be used without adjust, but in order for adjust to work, fill mode must be in effect.

LINE LENGTH

The line length is the maximum number of print positions in an output line, including all spaces and indentations, but not including margins set or implied by the -device, -indent, or -number control arguments.

BREAK

A break ensures that the text that follows is not run together with the text before the break. The previous line is printed out as is, without padding.

SPACING BETWEEN LINES

Vertical spacing within the body of the text is controlled by the three control words: .ss, .ds, and .ms (for single, double, and multiple spacing respectively). Single spacing is the default. Multiple spacing is set by the control line .ms N where N-1 is the number of blank lines between text lines.

PAGE EJECT

A page eject ensures that no text after the control line causing the page eject (e.g., .bp for "begin page") is printed on the current page. The current page is finished with only footers and footnotes at the bottom, and the next text line begins the following page.

MARGINS

There are four margins on the page vertically. The first margin on the page is the number of blank lines between the top of the page and the first header; this margin is set by the .m1 control word. The second, set by .m2, specifies the number of lines between the last header and the first line of text. The third (.m3) is between the last line of text and the first footer. The fourth (.m4) is between the last footer and the bottom of the page. The default for the first and fourth margins is four lines; for the second and third, two lines.

PAGE NUMBERS

As the output is being prepared, a page number counter is kept. This counter can be incremented or set by the user. The current value of the counter can be used in a header or footer through the use of the symbol "%". A page is called odd (even) if the current value of the counter is an odd (even) number. The page numbers can be output as either arabic (the default) or roman (using the .ro control word).

HEADERS AND FOOTERS

A header is a line printed at the top of each page. A footer is a line printed at the bottom of each page. A page can have up to 20 headers and 20 footers. Headers are numbered from the top down, footers from the bottom up. The two groups are completely independent of each other. Provision is made for different headers and footers for odd and even numbered pages. Both odd and even headers (footers) can be set together by using the .he (.fo) control words. They are set separately by using the .eh, .oh, .ef, and .of control words.

A header/footer control line has two arguments, the line number (denoted in the control line descriptions as "#"), and the title.

The line number parameter of the control line determines which header or footer line is being set. If the number is omitted, it is assumed to be 1, and all previously defined headers or footers of the type specified (odd or even) are cancelled. Once set, a line is printed on each page until reset or cancelled.

.nf
No fill: fill mode is suppressed, so that a break is caused after each text line. Text is printed exactly as it is in the input segment. This control line causes a break.

.of # 'part1'part2'part3'
Odd footer: this defines odd page footer line number #. If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all footers defined by any .of control line are cancelled. For more information, see the discussion entitled "Headers and Footers."

.oh # 'part1'part2'part3'
Odd header: this defines odd page header line number #. If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all headers defined by any .oh control line are cancelled. For more information, see the discussion entitled "Headers and Footers."

.op
Odd page: the next page number is forced to be odd by adding 1 to the page number counter if necessary. A break is caused and the current page is ejected. No blank even page is made; the even page number is merely skipped.

.pa +/-N
the current line is finished as is (i.e., a break occurs) and the current page is ejected. The page number counter is set to N, or is changed by N if N was signed. If N is omitted, the page number counter is incremented by 1.

.pi N
Picture: if N lines remain on the present page, N lines are spaced over; otherwise, the text continues as before until the bottom of the page is reached, N lines are skipped on the next page before any text is printed. Headers are printed normally; the space resolved is below the headers. This option can be used to allow for pictures and diagrams. If several .pi control lines occur consecutively, each N is added to the number of lines pending and the total is checked against the space remaining on the page. All pending space is allotted together. If the total is greater than the usable space on a page, the next page contains only headers and footers and the rest of the space is left on the following page. If N is not specified, 1 is assumed.

.pl +/-N
Page length: the page length is set to N lines. If N is not specified, 66 is assumed. If N is preceded by a plus or a minus sign, the page length is changed by N rather than reset.

.rd
Read: one line of input is read from the user_input I/O switch; this input line is then processed as if it had been encountered instead of the .rd control line. Thus it can be either a text line or a control line; a break occurs only if the replacement line is one that would cause a break.

.ro

Roman numerals: when page numbers (% variable) are substituted into text or control lines as a result of a .ur control line or into a title or equation as it is printed, they are in lowercase roman notation. This can be reset to arabic numerals (the default) by use of the .ar control line.

.rt

Return: cease processing characters from the current input segment. If the current input segment was entered by a .if control line in another segment, return to the line following the .if control line.

.sk N

Skip: N page numbers are skipped before the next new page by adding N to the current page number counter. No break in text occurs. This control line can be used to leave out a page number for a figure. If N is not specified, 1 is assumed.

.sp N

Space N lines: If N is not specified, 1 is assumed. If not enough lines remain on the current page, footers are printed and the page ejected, but the remaining space is not carried over to the next page. The N blank lines are produced in addition to any that may occur automatically due to a .ds or .ms control line. For example, if .sp 4 is used with .ss or .ms 1, in effect four blank lines appear between two text lines, with .ds or .ms 2, five lines appear, with .ms 3, six lines.

After skipping the space, the equivalent of a .ne 2 is performed in an attempt to avoid separating the first line of a paragraph at the bottom of a page from the rest of the paragraph on the next page. The .ne feature can be avoided, if the user so desires, by using a blank line rather than .sp. Otherwise, a blank line is treated as if it were a .sp 1 control line.

This control line causes a break.

Note: A series of .sp control lines such as:

```
.sp a
.sp b
```

is not always equivalent to a single .sp control line whose argument is the sum of the individual arguments:

```
.sp a+b
```

If the .sp a finishes a page, causing a page ejection, b blank lines are produced at the top of the new page. If .sp a+b is used, the space does not appear at the top of the next page.

.sr name <expression>

Set reference: associates value of <expression> with the identifier name. The type of name is set to the type of <expression> (either numeric or string); if the expression is not provided or cannot be properly evaluated, a diagnostic message is printed. The name identifier can be either a user-defined identifier or one of the built-in symbols that the user can set (see "Built-In Symbols" below).

.ss

Single space: begin single spacing text. This is the default condition. This control line causes a break.

.tr cd..

Translate: the nonblank character c is translated to d in the output. An arbitrary number of cd pairs can follow the initial pair on the same line without intervening spaces. An unpaired c character at the end of a line translates to a blank character. (Translation of a graphic character to a blank only in the output is useful for preserving the identity of a particular string of characters, so that the string is neither split across a line, nor has padding inserted within it.) If several .tr control lines are used in a segment, the cd pairs are "added together." Also a particular c character can be translated to a different d character by using a new .tr control line to override the previous translation. To cancel a cd pair (i.e., have the _c character print out as itself), use another .tr control line of the form ".tr cc". A .tr control line with no cd pair is ignored.

.ts N

Test: process the next input line if the value of N does not equal zero (false). If N is not specified, 1 is assumed.

.ty STR

Type: write STR (i.e., the rest of the control line) onto the error_output I/O switch. Substitution of variables can occur if the first or second character of STR is "%". If STR is omitted, a blank line is written onto the I/O switch.

.un N

Undent: the next output line is indented N spaces less than the current indentation. Adjustment, if in effect, occurs only on that part of the line between the normal left indentation and the right margin. If N is not specified, its value is the current indentation value (i.e., the next output line begins at the current left margin). This control line causes a break.

.ur text

Use reference: the remainder of the .ur control line (text) is scanned, with variables of the form "%name%" replaced by their corresponding values (converted back to character string form if they were numeric). The line thus constructed is then processed as if it had been encountered in the original input stream (e.g., it can be another control line, including possibly another .ur).

.wt

Wait: read one line from the user_input I/O switch and discard it (see the .rd control word description).

.*

This line is treated as a comment and ignored. No break occurs.

.~

This line is treated as a comment and ignored with respect to the output segment; however, the line is printed in the appropriate place in the chars output segment.

SUMMARY OF CONTROL WORDS

The following conventions are used to specify arguments of control words:

#	integer constant.
c	character.
cd	character pair.
exp	expression (either numeric or string).
N	integer expression.
+/-N	indicates update by N; if sign not present, set to N.
f	segment name.
t	title of the form 'part1'part2'part3'.

SUMMARY OF REQUESTS

Request	Break	Default	Meaning
.ad	yes	on	Right justify text.
.ar	no	arabic	Arabic page numbers.
.bp	yes		Begin new page.
.br	yes		Break, begin new line.
.cc c	no	%	Change special character from % to c.
.ce N	yes	N=1	Center next N lines.
.ch cd....	no		Note "c" in chars segment as "d".
.ds	yes	off	Double space.
.ef # t	no		Defines even footer line #.
.eh # t	no		Defines even header line #.
.eq N	yes	N=1	Next N lines are equations.
.ex text	no		Call command processor with "text".
.fh t	no	line of underscores	Format of footnote demarcation line.

Request	Break	Default	Meaning
.fi	yes	on	Fill output lines.
.fo # t	no		Equivalent to: .ef # t .of # t
.fr c	no	t	Control footnote numbering: "t" reset each page. "f" continuous. "u" numbering suppressed for next footnote.
.ft	no		Delimits footnotes.
.gb STR	no		"go back" to label ST.
.gf STR	no		"go forward" to label STR.
.he # t	no		Equivalent to: .eh # t .oh # t
.if f exp	no		Segment f.runoff inserted at point of request; value of "exp" assigned to "Parameter".
.in +/-N	yes	N=0	Indent left margin N spaces.

.li N	no	N=1	Next N lines treated as text.
.li +/-N	no	N=65	Line length is N.
.ma +/-N	no	N=4	Equivalent to: .m1 N .m4 N
.mp +/-N	no	N=1	Print only every N-th page.
.ms +/-N	yes	N=1	Multiple space N lines.
.m1 +/-N	no	N=4	Margin above headers set to N.
.m2 +/-N	no	N=2	Margin between headers and text set to N.
.m3 +/-N	no	N=2	Margin between text and footers set to N.
.m4 +/-N	no	N=4	Margin below footers set to N.
.na	yes	off	Do not right justify.
.ne N	no	N=1	Need N lines; begin new page if not enough remain.
.nf	yes	off	Do not fill output lines; print them exactly as entered.
.of # t	no		Defines odd footer line #.
.oh # t	no		Defines odd header line #.
.op	yes		Next page number is odd.
.pa +/-N	yes		Begin page N.
.pi N	no	N=1	Skip N lines if N remain; otherwise skip N lines on next page before any text.
.pl +/-N	no	N=66	Page length is N.
.rd	no		Read one line of text from the user_input I/O switch and process it in place of .rd line.
.ro	no	arabic	Roman numeral page numbers.
.rt	no		"Return" from this input segment.
.sk N	no	N=1	Skip N page numbers before next new page.
.sp N	yes	N=1	Space N lines.
.sr sym exp	no		Assign value of "exp" to variable named "sym".
.ss	yes	no	Single space.
.tr cd....	no		Translate nonblank character c into d on output.
.ts N	no	N=1	Process the next input line only if N is not zero.
.ty STR	no		Write "STR" onto the error_output I/O switch.
.un N	yes	left margin	Indent next text line N spaces less.
.ur text	no		Substitute values of variables in "text", and scan the line again.
.wt	no		Read one line of text from the user_input I/O switch and discard it (for synchronization with terminal).
.*	no		Comment line; ignored.
.~	no		Comment line; ignored.

BUILT-IN SYMBOLS

Only those symbols marked "yes" in the Set column can have values assigned by the user.

All symbols are of type Number unless they are specified to be of type String.

Control words and control arguments that affect the values of the variables are indicated in parentheses: (x/y) indicates that x sets the switch to true (-1), and y sets it false (0); (a) or (a, b, c) indicates that it is affected by a or by a, b, and c.

Symbol	Set	Value
Ad		Adjust (.ad/.na).
Ce		Number of lines remaining to be centered (.ce).
CharsTable	yes	Translation table for chars segment output (String) (.ch).
Charsw	yes	A chars segment is being created (-character).
ConvTable	yes	Translation table for output. Product of DeviceTable and TrTable (String) (.tr, -device).
Date		Date of this invocation of runoff; format is mm/dd/yy (String).
Device	yes	Type of device output is to be formatted for (-device, -ball, -segment).
DeviceTable	yes	Translation table for physical device (String) (-device).
Eq		Equation line counter (.eq).
Eqcnt	yes	Equation reference counter (incremented each reference).
ExtraMargin	yes	Indent entire text this many spaces (-segment, -device, -indent).
Fi		Fill switch (.fi/.nf).
FileName		Name of current primary input segment (String).
Filesw		True if output is going to a segment (-segment).
Foot	yes	Footnote counter (.ft, .fr).
FootRef	yes	Footnote reference string in footnote body (String).
Fp	yes	First page to print (set at the beginning of each pass to the value of From).
Fr		Footnote counter reset switch.
From	yes	First page to print (-from).
Ft		Footnote processing switch (.ft).
Hyphenating	yes	True if an attempt to break a word should be made (-hyphenate).
In		Indent to here (.in).

Symbol	Set	Value
InputFileName		Name of current input segment (String) (.if).
InputLines		Current line number in current source file.
LinesLeft		Number of usable text lines left on this page.
Ll		Line length (.ll).
Lp	yes	Last page to print (initialized each pass from To).
Ma1		Space above header (.ma, .m1).
Ma2		Space below header (.m2).
Ma3		Space above foot (.m3).
Ma4		Space below foot (.ma, .m4).
Ms		Spacing between lines (ss = 1, ds = 2, etc.) (.ms, .ss, .ds).
MultiplePagecount		Form feeds between pages to printer (.mp).
NestingDepth		Index into stack of input files (.if).
Nl		Last used line number.
NNp	yes	Next page number (-page, .pa).
NoFtNo		True to suppress number on next footnote reference (.fr).
NoPaging	yes	True if no pagination is desired (-no_pagination).
Np	yes	Current page number (.pa, -page, initialized each pass from Start).
PadLeft		Alternate left/right padding switch (.un, .ad).
Parameter	yes	Argument passed during insert processing (-parameter, .if).
Passes	yes	Number of passes left to make (= 1 when printing is being performed) (-pass).
Pi		Space needed for pictures (.pi).
Pl		Page length (.pl).
Print	yes	Whether or not to print: ((Fp <= Np <= Lp) & (Passes <= 1))
Printersw		Output is intended for bulk printer (-device, -segment).
PrintLineNumbers	yes	True if source line numbers are to be printed in output (-number).
Roman		Roman numeral pagination (.ro/.ar).
Selsw		True if typeball other than 963 is being used (-ball).
Start	yes	Initial page number (-page).
Stopsw	yes	Stop between pages of output (-stop).

Symbol	Set	Value
TextRef	yes	Footnote reference string in main text (String).
Time		Local time, in seconds, since January 1, 1901.
To	yes	Last page to be printed (-to).
TrTable	yes	Translation table for user-supplied substitutions (String) (.tr).
Un		Undent to here (.un).
Waitsw	yes	Wait for input before printing first page (-wait).

NOTES ON HYPHENATION PROCEDURE CALLING SEQUENCE

The runoff command provides a means whereby a user-supplied program can be called whenever the space available on a line is less than the length of the next word (including attached punctuation, if any). The mechanism is activated by use of the -hyphenate control argument, and the PL/I calling sequence is provided below.

```
declare hyphenate_word_ entry(char(*) unaligned, fixed bin, fixed bin);

call hyphenate_word_(string, space, break);
```

where:

string
is the text word that is to be split. (Input)

space
is the number of print positions remaining in the line. (Input)

break
is the number of characters from the word that should be placed on the current line; it should be at least one less than the value of space (to allow for the hyphen), and can be 0 to specify that the word is not to be broken. Thus if the word "calling" is to be split, and 6 spaces remain in the line, the procedure should return the value 4 (adjustment is performed after hyphenation). (Output)

Name: runoff_abs, rfa

SYNTAX AS A COMMAND

```
rfa paths {-rf_args} {-dp_args} {-control_args}
```

FUNCTION

submits an absentee request to process text segments using the runoff command.

ARGUMENTS

paths
are the pathnames of segments to be processed by the runoff command.

rf_args
can be one or more control arguments accepted by the runoff command.

dp_args
can be one or more control arguments accepted by the dprint command, except -brief (-bf) and -truncate (-tc).

CONTROL ARGUMENTS

Control arguments and paths can be mixed freely and can appear anywhere on the command line.

-hold, -hd
specifies that runoff_abs should not dprint or delete the output file.

-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 th upper limit for the current shift are deferred to a shift with a higher limit.

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

-queue N, -q N
is the priority queue of the request. The default queue is defined by the site. See the Notes for a description of the interaction with the dprinting of output files.

NOTES

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

The -indent control argument is given to this command, it is interpreted as the runoff control argument; not as the dprint control argument.

If the -queue control argument is not specified, the request is submitted into the default absentee priority queue defined by the site and, if requested, the output files will be dprinted in the default queue of the request type specified on the command line. (If no request type is specified, the "printer" request type is used.)

If the -queue control argument is specified, the output files will be dprinted in the same number queue as the absentee request unless the -hold control argument is also specified. If the request type specified for dprinting does not have that queue, the highest numbered queue available for the request type is used and a warning is issued.

Name: sample_refs

SYNTAX AS A COMMAND

```
sample_refs -control_args
```

FUNCTION

periodically samples the machine registers to determine which segments a process is referencing. Three output segments are produced that are interpretable by the print_sample_refs command.

CONTROL ARGUMENTS

-segment name, -sm name
specifies the names to be given the three output segments; name can be either an absolute or relative pathname. If name does not end with the suffix srf, it is assumed. The output segments are named as follows:

(entry portion of) name.srf1
(entry portion of) name.srf2
(entry portion of) name.srf3

The default places the output segments in your working directory, with entrynames as follows:

mm/dd/yy__hhmm.m_zzz_www.srf1
mm/dd/yy__hhmm.m_zzz_www.srf2
mm/dd/yy__hhmm.m_zzz_www.srf3

-time N, -tm N
specifies the rate in milliseconds at which the process is sampled. N must be a positive integer. (Default: N = 1000; i.e., the process is sampled once every second)

NOTES

You are allowed one active invocation per process: a second invocation terminates the first, whereupon the new invocation proceeds normally..spb

You can sample the machine registers only when the process is running in a ring other than 0. For example, a process using a total of 100 seconds of processor time and sample_refs running at the default sample rate and recording 23 samples indicate that 77 seconds of processor time were spent in ring 0.

Under certain conditions, the contents of one of the machine registers sampled--the Temporary Segment Register (TSR)--can be invalid. This invalidity is noted, but does not necessarily indicate that the process is in error.

At the maximum sample rate (one millisecond) execution time can be increased by as much as 50 percent. Using a one-second sample rate, the increase in execution time is negligible.

The accuracy of sample rates less than 1000 milliseconds is not guaranteed due to load factors. The accuracy of such sample rates increases with load.

If the process being sampled is terminated without an invocation of sample_refs with the -reset option, interpretable output segments are still produced; however, both the off-time and the last recorded sample may be invalid.

Name: save_dir_info

SYNTAX AS A COMMAND

```
save_dir_info dir_path {seg_path}
```

FUNCTION

creates a segment containing all information available from the storage system about a directory and its contents.

ARGUMENTS

dir_path
is the pathname of the directory to be scanned.

seg_path
is the pathname of the directory information segment to be created. If you omit seg_path, the entryname portion of dir_path is assumed. If seg_path does not end with the dir_info suffix, it is assumed.

NOTES

This command is not recursive; the entire subtree inferior to the selected directory is not scanned, just the immediately inferior branches and links. The saved information segment can be processed by the comp_dir_info and list_dir_info commands.

Name: save_history_registers

SYNTAX AS A COMMAND

```
save_history_registers {state} {-control_args}
```

FUNCTION

allows a user to save processor history registers upon each occurrence of a signalable fault in the signalers stack frame. By default, the history registers are not saved, and the history register block in the signalers stack frame is set to all zeros.

ARGUMENTS

state
can be either "on" or "off." The default is off.

CONTROL ARGUMENTS

-print, -pr
displays the current state of the history register save switch if it is present without the state argument; with this argument, the state of the switch is displayed before the new state is applied.

-priv
specifies manipulation of the per-system state by directing the state and -print arguments to operate on the per-system history register save switch, wired_hardcore_data$global_hregs. When set, this switch causes all processes to save their history registers upon each occurrence of a signalable fault in the signalers stack frame. If -priv is not specified, then the state and -print arguments operate on pds$save_history_regs, the per-process history register save switch of the user's process executing this command.

ACCESS REQUIRED

When the -priv control argument is used, access to the hphcs_ gate is required.

Name: save_on_disconnect

SYNTAX AS A COMMAND

```
save_on_disconnect
```

FUNCTION

reverses the effect of the no_save_on_disconnect command, re-enabling process preservation across hangups in your process.

NOTES

This command is meaningful only if process preservation was in effect for the process at login time, either by default or because you gave -save_on_disconnect on the login command line.

Name: search, srh

SYNTAX AS A COMMAND

```
srh STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[srh STRA STRB]
```

FUNCTION

returns the integer representing the character position in strA of the leftmost occurrence of any character contained in strB. If no character of strB occurs in strA, 0 is returned.

EXAMPLES

The following lines from an exec_com segment demonstrate how you can use the search active function to check that argument 1 does not contain either of the special characters used by the star convention. If the argument does not contain special characters, execution continues; if it does, a message is printed and execution stops.

```
&if &[nequal [srh &1 *?] 0]
&then &goto continue
&print Star name not permitted:  &1
&quit
&label continue
```

The following interactions also illustrate the search active function:

```
string [srh "Paul, Mary;" ",;"]
5
string [srh "Harry" ",;"]
0
```

Name: segments, segs

SYNTAX AS A COMMAND

```
segs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[segs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of segments that match one or more star names.

ARGUMENTS

star_names
are star names to be used in selecting the names to be returned.

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

This page intentionally left blank.

Name: select

SYNTAX AS A COMMAND

```
select test_string {args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[select test_string {args}]
```

FUNCTION

tests a set of arguments and returns those arguments that pass the test. The test is given as the first argument and is used to select the second through last arguments. A string consisting of the concatenation of all the arguments which pass the test, separated by spaces, is returned. Each argument is requoted.

ARGUMENTS

test_string
: is the test to apply to each argument. See the "Notes" section below.

args
: are the arguments to be tested. Any number of arguments, including zero, may be supplied.

NOTES

Each argument is requoted, and the following active string is constructed:

```
[test_string argi]
```

The active string is evaluated. The result must be either "true" or "false". If the result is "true", argi is selected and the requoted argument appears in the result string. If the result is "false", the argument is not selected.

The test_string argument can be the name of an active function which takes one argument, or the name of an active function which expects more than one argument, followed by all but its last argument. In the latter case, test_string should be enclosed in quotes. The "do" and "if" active functions can be used in this manner to construct arbitrarily complex tests.

When used as a command, select prints out the string it would have returned as an active function.

EXAMPLES

Assume the working directory contains three segments, named a, b, and d:

```
! select "exists segment" a b c d e f
  a b d
```

In this example, the test_string is "exists segment". The following active strings are constructed:

```
[exists segment a]
[exists segment b]
[exists segment c]
[exists segment d]
[exists segment e]
[exists segment f]
```

These active strings are evaluated, producing the following results:

```
true true false true false false
```

Thus, the first, second, and fourth arguments are selected.

The following command line asks the user about each segment in his working directory, and renames each one for which he answers "yes" to "old." concatenated with its old name:

```
! rename ([select query [segs **]]) old.===
```

In the next example, the "do" active function is used to construct an active function which indicates whether its argument is a segment modified before 12/29/82 or not:

```
do "[date_time_before [status &r1 -dtcm] 12/29/82]"
```

This whole string serves as an active function. The following command line will delete any segments modified before 12/29/82:

```
! delete [select "do ""[date_time_before [status &r1 -dtcm]
                 12/29/82]""" [segs **]]
```

In the next example, an abbrev for an active function is defined which takes two or more dates as arguments. The first two dates specify limits. The active function returns all dates after the first two dates that are between the first two dates.

```
! .a DTRANGE do "[select ""date_time_before &r1""
      [select ""date_time_after &r2"" &rf3]]"

! string [DTRANGE 1jan1980 5may1981 2may1979 6may1982
      3march1981 2feb1981]

   3march1981 2feb1981
```

The inner select selects dates that are before the second argument, i.e. those which the second argument is after. Of those selected, the outer select selects dates after the first argument, i.e. those which the first argument is before.

Name: send_mail, sdm

SYNTAX AS A COMMAND

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

FUNCTION

sends a message to one or more recipients.

ARGUMENTS

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

CONTROL ARGUMENTS

-abbrev, -ab
: enables abbreviation expansion of request lines.

-abort
: prints an error message and returns to its caller immediately upon detecting an invalid address. An invalid address is either a sequence of arguments that cannot be converted into an address by send_mail (e.g., missing arguments, bad pathname syntax) or a nonexistent address (e.g., a nonexistent mailbox, a foreign address on a host that cannot be reached from the local system). (Default)

-acknowledge, -ack
: requests an acknowledgement from the recipients when they read the message.

-auto_write
: specifies that the qedx request automatically updates the message when you quit the editor.

-bcc addresses
specifies a list of "blind" recipients of the message. "Blind" recipients are listed in the bcc field for the message header. When the message is transmitted, this field is not included in the copy of the message sent to the primary and secondary recipients; it is, however, included in the copy of the message sent to the actual "blind" recipients. By default, the message has no "blind" recipients.

-brief, -bf
shortens some informative messages and suppresses others.

-cc {addresses}
specifies the secondary recipients of the message. By default, the message has no secondary recipients.

-debug, -db
enables send_mail's debugging facilities. It is not recommended for normal users of the command.

-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)

-from {addresses}
specifies the authors of the message. By default, the user issuing send_mail is the sole author of the message.

-input_file path, -if path
takes the message text from the specified file rather than from the terminal.

-line_length N, -ll N
specifies the line length to use for filling the message text. (Default: 72)

-long, -lg
prints the long form of all informative messages. (Default)

-no_abbrev, -nab
does not enable abbreviation expansion of request lines. (Default)

-no_abort
prints an error message for any invalid addresses that are encountered on the command line but then proceeds to prompt for a subject and message text. After you type the message text, send_mail enters its request loop to allow you to correct the lists of recipients before sending the message.

-no_acknowledge, -nack
does not request an acknowledgement. (Default)

-no_auto_write
specifies that the qedx request requires you to use the write request to update the

message before quitting the editor. Any attempt to exit without writing results in a query. (Default)

-no_debug, -ndb
disables sdm's debugging facilities. (Default)

-no_fill, -nfi
does not reformat the message text unless you use the fill request or the -fill control argument of the qedx and apply requests. (Default for file input)

-no_notify, -nnt
does not send notification messages.

-no_prompt
suppresses the prompt for request lines in the request loop.

-no_request_loop, -nrql
sends the message immediately upon completion of input unless input was from the terminal and was terminated by "\f" or "\q". (Default for terminal input)

-no_subject, -nsj
specifies that the message has no subject.

-notify, -nt
sends a "You have mail." notification to each recipient of the message. (Default)

-profile path, -pf path
specifies the pathname of the profile to use for abbreviation expansion. The suffix profile is added if necessary. This control argument implies -abbrev.

-prompt STR
sets the request loop prompt to STR. (Default: ^/send_mail^[(^d)^]:^2x)

-reply_to {addresses}, -rpt {addresses}
specifies the list of recipients who are to receive replies to the message instead of the message's authors. By default, the authors of the message receive the replies.

-request STR, -rq STR
executes STR as an sdm request line after reading the message text but before entering the request loop. This control argument implies -request_loop.

-request_loop, -rql
enters sdm's request loop after reading the message text. (Default for file input)

-subject STR, -sj STR
specifies the subject of the message. By default, sdm prompts you for the subject.

-terminal_input, -ti
accepts the message text from the terminal (see "Notes on Terminal Input" below). (Default)

-to {addresses}
specifies additional primary recipients of the message.

LIST OF ADDRESSES

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

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

It is included as a "blind" recipient of the message.

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

-mailing_list path, -mls path
specifies the pathname of a mailing list. The suffix mls is added if necessary. You can use the archive component pathname convention. A mailing list is a list of addresses contained in a segment or archive component.

-meeting path, -mtg path
specifies the pathname of a Forum meeting. The suffix control or forum is added if necessary. If the pathname given is just an entryname (i.e., no < or > characters appear in the pathname), the forum search path is used to find the meeting.

-save path, -sv path
specifies the pathname of a savebox. The suffix sv.mbx is added if necessary. It is included as a "blind" recipient of the message.

-user STR
specifies either a user's default mailbox or an entry in the system mail table (see "Notes on the -user Address Control Argument" below).

STR
is any noncontrol argument. If STR contains either < or >, it is interpreted as -mailbox path; otherwise it is interpreted as -user STR.

STR -at FSystem {-via RelayN...-via Relay1}
specifies an address on another computer system (see "Notes on Foreign Address").

LIST OF ADDRESS QUALIFIERS

-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 the user's full name. It is considered obsolete.

-name STR, -nm STR
must appear immediately following one of the above forms of an address and specifies the name of the address. An address name is usually the full name of the person who receives mail at that address or, for mailing lists, a description of the addresses comprising the mailing list (e.g., site administrators).

LIST OF REQUESTS

In the following summary of sdm requests, "-ca" is used as shorthand for "-control_args". For a complete description of any request, issue the sdm request:

```
! help request_name
```

.
prints a line describing the current invocation of sdm.

?
prints a list of requests available in sdm.

abbrev {-ca}, ab {-ca}
controls abbreviation processing of request lines.

answer STR -ca request_line
provides preset answers to questions asked by another request.

append path, app path
writes the ASCII representation of the message to the end of a segment.

apply {-ca} cmd_line, ap {-ca} cmd_line
passes the message text and header to a Multics command line for possible editing.

bcc {addresses}
prints or updates the list of "blind" recipients of the message. Blind recipients are listed in the bcc field for the message header. When the message is transmitted, this field is not included in the copy of the message sent to the primary and secondary recipients; it is, however, included in the copy of the message sent to the actual blind recipients.

cc {addresses}
prints or updates the list of secondary recipients of the message.

copy path, cp path
    copies the message into the specified mailbox.

debug_mode {-ca}
    enables/disables sends_mail's debugging facilities.

do rq_str {args}, [do rq_str args]
    executes/returns a request line with argument substitution.

exec_com ec_path {ec_args},
ec ec_path {ec_args},
[exec_com ec_path {ec_args}],
[ec ec_path {ec_args}]
    executes a file of read_mail requests that can return a value.

execute cmd_line,
e cmd_line,
[execute active_str],
[e active_str]
    executes a Multics command line or evaluates a Multics active string.

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 sdm requests and other topics.

if expr -then line1 {-else line2},
[if expr -then STR1 {-else STR2}]
    conditionally executes/returns one of two request lines.

list_help {topics}, lh {topics}
    displays the name of all sdm info segs on given topics.

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

log
    places a copy of the message into the user's logbox.

message_id, mid
    prints the unique identifier of the message and includes a Message ID field in the message.

preface path, prf path
    writes the ASCII representation of the message to the beginning of a segment.

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 Multics Qedx editor.

quit {-ca}, q {-ca}
exits sdm.

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.

You can use the following requests only within an invocation of sdm that is created using the read_mail reply request. In this summary, "specs" is short for "message_specifiers" and "-c/sa" is short for "-control_args -selection_args".

in_reply_to {specs}, irt {specs}
prints or changes the content of the message's In Reply To field.

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_original {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.

NOTES ON THE -user ADDRESS CONTROL ARGUMENT

A user's default mailbox is specified in the form Person_id.Project_id. For an entry in the mail table, STR is usually in the form of Person_id (the mail table is fully described in the *Extended Mail System User's Guide*, CH23).

If STR contains one period and no white space, it is interpreted as a User_id that specifies the user's default mailbox; otherwise it is interpreted as the name of an entry in the mail table.

For example:

```
-user DBuxtehude.SiteSA
```

is interpreted as a User_id that identifies a default mailbox. On the other hand,

```
-user "George G. Byron"
-user L.v.Beethoven
-user Burns
```

are all interpreted as the names of entries in the mail table: the first because it contains white space; the second because it contains more than one period; the third because it contains no period.

When interpreted as a User_id, the STR cannot contain any angle brackets (<>) and must have the form Person_id.Project_id, where "Person_id" cannot exceed 28 characters in length and "Project_id" is limited to 32 characters. In this case, "-user STR" is equivalent to the address -mailbox >udd>Project_id>Person_id>Person_id.mbx.

When interpreted as the name of a mail table entry, STR cannot contain any commas, colons, semicolons, backslashes (\), parentheses, angle brackets, braces ({}), quotes, commercial at-signs (@), or white space other than spaces. The query of the mail table is performed in a case-insensitive manner. The display_mailing_address command can be used to determine the actual address corresponding to the STR.

NOTES ON FOREIGN ADDRESS

STR identifies the user (or group of users) to receive the message and is not interpreted in any way by the local system. FSystem is the name of the foreign system where the address is located.

If the -via control arguments are not given, FSystem must be one of the names of a foreign system in the local system's network information table (NIT); if they are given, however, the foreign system name need not be known to the local system.

The -via control arguments identify an explicit route to be used to reach the foreign system. Relay1 must be one of the names of a foreign system in the local system's NIT. Mail destined for this foreign address is forwarded to the system identified as Relay1, then to the system identified as Relay2, and so on until it reaches the system identified as RelayN where it is delivered to the system on which the foreign address actually resides.

When the NIT is queried for either FSystem or Relay1, the query is performed in a case-insensitive manner.

For example, the address

```
HDT -at OZ -via MC -via mit-multics
```

identifies the address HDT on a system named OZ. The local system relays mail sent to this address to the system mit-multics, which then forwards the mail to a system named MC, which actually delivers the mail to its final destination.

NOTES ON TERMINAL INPUT

By default or if you give -terminal_input, send_mail issues the prompt "Message:" and reads the message text from the terminal.

If you terminate the text with a line containing just a period, send_mail reformats the message (unless you provide -no_fill on the command line) and sends it to the specified recipients (unless you also give -request or -request_loop on the command line). If any errors occur while sending the message, send_mail enters its request loop to allow you to correct the problem.

If you terminate the text with a line containing "\f" anywhere on the line, the command enters the qedx editor on the message text. Any characters on the line after the "\f" are treated as qedx requests. (See "Notes on the qedx Editor" below.)

If you terminate the text with a line containing "\q" anywhere on the line, send_mail reformats the message, unless you supply -no_fill on the command line, and enters its request loop. Any characters on the line after the "\q" are ignored with a warning message.

NOTES ON THE Qedx EDITOR

You can invoke the qedx editor either by the "qedx" request in the send_mail request loop or by terminating a message being input with "\f". Any requests typed are processed as qedx requests until you enter the qedx "q" (quit) request and you are returned to send_mail's request loop.

Use the "w" (write) request to reflect any changes made to the message text. If you issue the "q" request and you have modified the message since it was last written, qedx queries for permission to exit; if permission is given, any changes made since the last write are lost. You can use the "qf" (quit-force) request to abort unwanted editing of the message without being queried.

The request line

```
1,$dr
```

only restores the original message text to the buffer if you have not yet used the write request, but it restores the message text as saved by the last write request in the buffer if you give it after a write request.

Type "help qedx" within sdm for more information on the qedx request.

NOTES ON ADDRESSES ON THE send_mail COMMAND LINE

Successive uses of -from, -cc, -reply_to, and -to do not override previous uses; instead, the addresses specified in the multiple uses are merged to form the actual list.

For example,

```
! sdm DErasmus.Multics -from JJRousseau.PDO -to BShields.Multics
```

sends the message from JJRousseau.PDO to DErasmus.Multics and BShields.Multics.

Name: send_message, sm

SYNTAX AS A COMMAND

```
sm {-control_args} address {message}
```

FUNCTION

sends a message(s), one line at a time, to a given user on a given project or to a specified mailbox.

ARGUMENTS

address
: can be of the form Person_id.Project_id to specify a mailbox belonging to that person; a string containing at least one > or < to specify the pathname of a mailbox; one of the arguments -log, -mailbox (-mbx), or -save (-sv), immediately followed by a string giving the pathname of a logbox, mailbox, or save box, respectively; -last_message_destination (-lmds) if you have used send_message in this process; or -last_message_sender (-lms) if a message has been received in the user's default mailbox. All arguments beginning with the first noncontrol argument after a destination are considered to be message text.

message
is anything that appears up to the end of the command. It can be one or more words. If you omit it, sm enters an input loop; you can then send a multiline message (see "Notes on Input Loop" below).

CONTROL ARGUMENTS

-access_class STR, -acc STR
sends messages at the specified AIM access class. The ring 1 privilege must be turned on in the sending process.

-acknowledge, -ack
requests that the recipient's process return an acknowledgment message when the message is read. It implies -brief.

-acknowledge_if_deferred, -ackid
requests that the recipient's process return an acknowledgment message when the message is read only if the recipient is not accepting messages or has deferred them. Requesting acknowledgments while in the input loop is not affected by changes in the recipient's wakeup state.

-brief, -bf
does not print an error message if the message cannot be sent or if the recipient is not accepting messages.

-comment STR, -com STR
adds a comment of the form (STR) after the user's person and in the message's header. The default is to use the value of the "full_name._" variable in the user's default value segment as a comment.

-escape, -esc
turns on the ".." escape convention to execute Multics commands from within input mode for the current message. (Default)

-express, -xps
sends the message only if the recipient is likely to see it immediately, that is, is currently accepting messages.

-long, -lg
prints error messages. (Default)

-no_acknowledge, -nack
requests that the recipient's process do not return an acknowledgment message when the message is read. (Default)

-no_comment, -ncom
suppresses adding of a comment.

-no_escape, -nesc
    turns off the ".." escape convention.

-no_express, -nxps
    always sends the message. (Default)

-no_print_destination, -nprds
    does not print the destination to which the message is being sent if you supply the message on the command line. (Default, if you use -lmds and -lms)

-no_update_destination, -nupds
    does not set the last message destination.

-print_destination, -prds
    prints the destination to which the message is being sent if you supply the message on the command line. (Default, if you don't use -lmds and -lms)

-silent, -sil
    suppresses all error messages.

-update_destination, -upds
    sets the last message destination. (Default)

NOTES

If the message has a parenthesis, bracket, or semicolon character, enclose it in quotes because that character is not treated specially by sm. You can type a quotation mark as """".

For a description of the mailbox, see accept_messages and print_mail.

NOTES ON INPUT LOOP

When sm enters the input loop it types "Input to <destination>" and accepts lines that are sent one at a time. Input loop is terminated by a line consisting solely of a period. When in input loop, you can execute Multics commands if typed on a line beginning with two periods. You can receive messages while in the input loop, so this is a way to hold conversations.

If the user whom you are sending messages to changes message status (e.g., defers messages, logs out), sm prints a message to that effect unless you supplied -bf or -sil.

EXAMPLES

If SHolmes on the Alpha project sends the following to AChristie on the Beta project:

```
sm AChristie.Beta ! need access to your lsg command.
```

If AChristie is accepting messages, the message prints as follows:

```
Message from SHolmes.Alpha 07/11/86 1200.6 mst Fri:
! need access to your lsg command.
```

If an acknowlegment is requested, the acknowledgment says

```
From SJohnson <time>: Acknowledged.
```

if the message is read right away, or

```
From SJohnson <time>
Acknowledge message of <sent_time>
```

if the message is read later.

If the recipient has insufficient access to return an acknowledgment, none is sent. No error message is printed.

The command line

```
sm AChristie.Beta Testing complete; please install this week.
```

sends

```
Message from SHolmes.Alpha 07/11/86 1505.7 mst Fri:
Testing complete
```

and then prints the error message "Segment please not found." because the characters typed after the semicolon are interpreted as another command line.

The command line

```
sm AChristie.Beta So long (for now).
```

sends two lines:

```
So long for.
So long now.
```

The sender's intended message would have been sent if it had been enclosed in quotes: "So long (for now)."

set_acl

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 nonlink entries in a directory (see the Programmer's Reference Manual for a discussion of ACLs).

ARGUMENTS

path
: If it is -working_directory (-wd), the user's working directory is assumed. You can use the star convention, which applies to types of entries, depending on the type of mode specified in modeN.

This page intentionally left blank.

modeN
is a valid access mode. For segments or multisegment files it can consist of any or all the letters rew; for directories, of any or all the letters sma except that if you give m, you must supply s also. Use null (n, "") to specify null access. To obtain a list of modes for extended types, see the describe_entry_type command.

User_idN
is an access control name of the form Person_id.Project_id.tag. All ACL entries with matching names receive modeN. (For a description of the matching strategy, see "Examples" below.) If no match is found and you give the three components, an entry is added to the ACL. If you omit the last User_id, your Person_id and 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 omits components.

-chase
chases links matching a star name. Links are always chased when path is not a star name.

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

-no_sysdaemon, -nsd
does not add "rw *.SysDaemon.*" when using -replace.

-replace, -rp
deletes all ACL terms--with the exception of the default *.SysDaemon.* term unless you supplied -no_sysdaemon--before adding the terms specified on the command line. (Default: to add to and modify the existing ACL)

-sysdaemon, -sd
adds, with -replace, a "rw *.SysDaemon.*" ACL term before adding the terms specified on the command line. (Default)

Select either of the following control arguments to avoid the ambiguity that occurs only when modeN is null and you use the star convention in path:

-directory, -dr
affects directories only.

-interpret_as_extended_entry, -inaee
interpret the selected entry as an extended entry type.

-interpret_as_standard_entry, -inase
interpret the selected entry as a standard entry type.

-segment, -sm
affects segments and multisegment files only. (Default)

-select_entry_type STR, -slet STR
affects only entries of the entry type selected by STR, which is a comma-delimited list of file system entry types. Use the list_entry_types command to obtain a list of valid entry type values.

ACCESS REQUIRED

You require 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. When you use the star convention to specify the last component of an entryname, extended entries are excluded from any matches.

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 an entry with mode rew to *.*.* (everyone) to the ACL of every segment in the working directory that has a two-component name with a second component of pl1 if that entry does not exist; otherwise, it changes the mode of the *.*.* entry to rew.

The command line

```
! sa -wd sm Keats.Faculty
```

adds an entry with mode sm for Keats.Faculty.* to the ACL of the working directory if that entry does not exist; otherwise, it changes the mode of the Keats.Faculty.* entry to sm.

The command line

```
! sa alpha.basic rew .Faculty. r Keats.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 Keats.Faculty to r.

The command line

```
! sa foo.mbx adrosw Degas
```

setts the ACL of the mailbox named foo.mbx to adrosw for the Person_id Degas.

Name: set__bit__count, sbc

SYNTAX AS A COMMAND

```
sbc path1 count1 {...pathN countN}
```

FUNCTION

sets a specified bit count on a specified segment, multisegment file (MSF), data management (DM) file, or extended entry and changes the bit count author for that entry to be the user who invoked the command.

ARGUMENTS

pathi
: is the pathname of the entry. If pathi is a link, the bit count of the entry linked to is set.

counti
is the bit count, in decimal, desired for pathi.

ACCESS REQUIRED

You must have write access on the entry whose bit count is to be set.

NOTES

Setting the bit count on a directory is permitted, but several system modules then regard the directory as a MSF.

See Section 2 of the Programmer's Reference Manual for a description of the bit count and bit count author.

Name: set_cc

SYNTAX AS A COMMAND

```
set_cc fileNN {-control_arg}
```

FUNCTION

sets the carriage control transformation for a specified FORTRAN formatted file either on or off.

ARGUMENTS

fileNN
is the name of a FORTRAN file in the range of file01 to file99. If fileNN is out of range, an error message is printed.

CONTROL ARGUMENTS

-off
turns the carriage control transformation off for the specified FORTRAN file.

-on
turns the carriage control transformation on for the specified FORTRAN file.

NOTES

When the transformation is on, the first character of each line written to the file is changed to a control character in accordance with the following table:

Character	Resulting Control Character
0	Newline 012 (double space)
1	Newpage 014 (page eject)
blank	None (single space)
+	The previous and current lines are written as a single line split by a carriage return character, which causes the second line to overprint the first. If the file is attached to a terminal, the + is ignored; the result is a single space between lines.

When the transformation is off, the first character is not changed. The default is off for all files except for file06 and file42, for which the default is on.

EXAMPLES

To turn off the carriage control transformation for file06, type:

```
set_cc file06 -off
```

Name: set_dir_ring_brackets, sdrb

SYNTAX AS A COMMAND

```
sdrb path {rb1 {rb2}}
```

FUNCTION

allows a user to modify the ring brackets of a specified directory.

ARGUMENTS

path
: is the relative or absolute pathname of the directory whose ring brackets are to be modified.

rb1, rb2
: are the numbers that represent the directory ring brackets: rb1 is the number to be used for the first ring bracket of the directory, and rb2 is the one to be used for the second. The ring brackets must be in the allowable range v through 7 (where v depends on your current validation level) and must have the ordering

```
rb1 <= rb2
```

If rb1 and rb2 are omitted, they are set to your current validation level. If rb1 is omitted, rb2 cannot be given and rb1 and rb2 are set to your current validation level.

NOTES

Your process must have a validation level less than or equal to rb1. See the Programmer's Reference Manual for a discussion of ring brackets and validation levels.

Name: set_epilogue_command

SYNTAX AS A COMMAND

```
set_epilogue_command command_line
```

FUNCTION

establishes the command line as an epilogue of the process. Just before a process is destroyed the command line is executed.

ARGUMENTS

command_line
is a single string containing the command line to be executed. Quote it if it contains spaces, special characters, abbreviations, etc.

NOTES

This command can only store a single command line. If you invoke it several times within a single process, only the final command line is executed when the process terminates.

EXAMPLES

The command lines

```
set_epilogue_command "string a"
set_epilogue_command "string b"
logout
b
```

establish "string b" to be executed when the process terminates.

The command line

```
set_epilogue_command "application$close_date_bases"
```

establishes "application$close_date_bases" to be executed when the process terminates.

Name: set_fortran_common, sfc

SYNTAX AS A COMMAND

```
sfc paths {-control_arg}
```

FUNCTION

initializes common storage for a FORTRAN run. Supply as an argument every object file that is part of the FORTRAN run to ensure that the common blocks are properly initialized. This command allows you to specify the files containing the block data subprograms prior to the run.

ARGUMENTS

paths
is a list of pathnames of files containing block data subprograms that initialize common.

CONTROL ARGUMENTS

-long, -lg
prints a message if a referenced common block has already been allocated.

NOTES

This command is useful in the run exec_com, which initializes the environment for a FORTRAN run.

Due to dynamic linking in Multics, if the first program to reference a common block is not compiled or bound with the block data subprogram that initializes the common block, this block may not be successfully initialized.

Any common blocks referenced in the specified files are allocated (if necessary) and initialized. If no initialization information is associated with the referenced common block, it is initialized to binary zeroes. If a common block was previously allocated, it is effectively deleted and reinitialized.

Name: set_iacl_dir, sid

SYNTAX AS A COMMAND

```
sid path model {User_id1...modeN User_idN} {-control_args}
```

FUNCTION

manipulates the directory initial access control lists (initial ACLs) of directories.

ARGUMENTS

path
: specifies the directory whose directory initial ACL is to be changed. If it is -working_directory (-wd), the directory initial ACL for the user's working directory is changed. You can use the star convention.

modes
: is the mode associated with User_ids. It can consist of any or all the letters sma except that if you give m, you must also give s. The strings null, n, and "" specifically deny access to User_ids.

User_ids
: is an access control name of the form Person_id.Project_id.tag. If one or more of the components is missing, all entries that match User_ids are changed to modes (see set_acl for a description of the matching strategy). If the three components are present, the directory initial ACL entry with that name is changed to modes or one is added if none exists. If the last modes has no User_ids following it, your name and project are assumed.

CONTROL ARGUMENTS

-no_sysdaemon, -nsd
: does not add "sm *.SysDaemon.*" when using -replace.

-replace, -rp
: deletes all directory initial ACL terms--with the exception of the default *.SysDaemon.* term unless you supplied -no_sysdaemon--before adding the terms specified on the command line. (Default: to add to, and modify, the existing initial ACL)

-ring N, -rg N
: identifies the ring number whose directory initial ACL is to be set. It can appear anywhere on the line, except between a mode and its associated User_id, and affects the whole line. If present, follow it by N (where 0 <= N <= 7). If omitted, your ring is assumed.

-sysdaemon, -sd
: adds, with -replace, an "sm *.SysDaemon.*" initial ACL term before adding the terms specified on the command line. (Default)

NOTES

A directory initial ACL contains the ACL entries to be placed on directories created in the specified directory (see "Access Control" in the Programmer's Reference Manual).

EXAMPLES

The command line

```
! sid listings sm * -ring 5
```

adds an entry, if it does not exist, with the mode sm for everyone (*.*.*) to the ring 5 directory initial ACL of the listings directory; otherwise it changes the mode of the *.*.* entry to sm.

The command line

```
! sid -wd sa BShields..
```

changes the mode of all entries with Person_id BShields in the directory initial ACL of the working directory to sa. If no such entries exist, an error message is printed.

Name: set_iacl_seg, sis

SYNTAX AS A COMMAND

```
sis path model {User_id1...modeN User_idN} {-control_args}
```

FUNCTION

manipulates the segment initial access control lists (initial ACLs) of directories.

ARGUMENTS

path
: specifies the directory whose segment initial ACL is to be changed. If it is -working_directory (-wd), the segment initial ACL for the user's working directory is changed. You can use the star convention.

modes
: is the mode associated with User_ids. It can consist of any or all the letters rew. The strings null, n, and "" specifically deny access to User_ids.

User_ids
is an access control name of the form Person_id.Project_id.tag. If one or more of the components is missing, all entries that match User_ids are changed to modes (see set_acl for a description of the matching strategy). If the three components are present, the segment initial ACL entry with that name is changed to modes or one is added if none exists.

CONTROL ARGUMENTS

-no_sysdaemon, -nsd
does not add "rw *.SysDaemon.*" when using -replace.

-replace, -rp
deletes all segment initial ACL terms--with the exception of the default *.SysDaemon.* term unless you supplied -no_sysdaemon--before adding the terms specified on the command line. (Default: to add to, and modify, the existing initial ACL)

-ring N, -rg N
identifies the ring number whose segment initial ACL should be set. It can appear anywhere on the line, except between a mode and its associated User_id, and affects the whole line. If present, follow it by N (where 0 <= N <= 7). If omitted, your ring is assumed.

-sysdaemon, -sd
adds, with -replace, an "rw *.SysDaemon.*" initial ACL term before adding the terms specified on the command line. (Default)

NOTES

A segment initial ACL contains the ACL entries to be placed on segments created in the specified directory (see "Access Control" in the Programmer's Reference Manual).

EXAMPLES

The command line

```
! sis test rew *
```

adds an entry with mode rew for everyone (*.*.*) to the segment initial ACL in the test directory if that entry does not exist; otherwise it changes the mode of the *.*.* entry to rew.

The command line

```
! sis -wd re Socrates.. -rg 5
```

changes the mode of all entries with Person_id Socrates in the ring 5 segment initial ACL of the working directory to re. If no such entries exist, an error message is printed.

Name: set_ips_mask

SYNTAX AS A COMMAND

```
set_ips_mask {signal_names} {-control_args}
```

FUNCTION

sets the IPS mask for the current process to mask some or all IPS signals.

ARGUMENTS

signal_names
are the names of one or more IPS signals to be masked. The signal names must be defined in sys_info$ips_mask_data. Presently the defined signal names are quit, alrm, neti, cput, trm_, sus_, wkp_, pgt_, system_shutdown_scheduled_, and dm_shutdown_scheduled_. Supply either signal_names or -all.

CONTROL ARGUMENTS

-all, -a
sets the IPS mask to unmask all IPS signals.

-brief, -bf
suppresses printing of the previous state of the IPS mask after setting it.

-long, -lg
prints the previous state of the IPS mask after setting it. (Default)

NOTES

If all undefined IPS signals are either masked or unmasked and you give -lg, they are not mentioned; if, however, some are masked and others are not, an octal list is printed. This can only happen when you specify an invalid (probably uninitialized) value in a call to set that mask.

Name: set_mailing_address, smla

SYNTAX AS A COMMAND

```
smla {address} {-control_args}
```

FUNCTION

sets the user's preferred mailing address, which is used by the mail system when mail is addressed to him by Person_id or alias alone (i.e., "sdm Opus", instead of "sdm Opus.Bloom"). The user can also indicate that his mailing address be reset to the default (Person_id.default_Project_id). For example, mail addressed to "Milo" is sent to Milo.DProject, where "DProject" is Milo's default project at the time the mail is sent. Maintainers of other mail table entries can also use this command to update those entries.

ARGUMENTS

address
: can be any recipient address accepted by send_mail. You can specify only one address. It is incompatible with -dp.

CONTROL ARGUMENTS

-default_project, -dp
: resets the mailing address using the default project.

-user name
: specifies the entry whose mailing address is to be updated. Enclose the name in quotes if it contains white space. If name is an alias, its associated regular entry is updated. You can use -user only if you have rw access to the ACS segment associated with the entry. (Default: your own entry)

NOTES

Don't use -dp if the entry is not associated with a registered user, since only users have default projects. If you attempt this, an error is reported.

Name: set_max_length, sml

SYNTAX AS A COMMAND

```
sml path length {-control_args}
```

FUNCTION

allows the maximum length of a nondirectory segment to be set.

ARGUMENTS

path
is the pathname of the segment whose maximum length is to be set. If path is a link, the maximum length of the target segment of the link is set. You can use the star convention.

length
is the new maximum length expressed in words. If this length is not a multiple of 1024 words, it is converted to the next higher multiple of 1024 words.

CONTROL ARGUMENTS

-brief, -bf
suppresses the warning message that the length argument has been converted to the next multiple of 1024 words.

-decimal, -dc
specifies that length is a decimal number. (Default)

-interpret_as_extended_entry, -inaee
interpret the selected entry as an extended entry type.

-interpret_as_standard_entry, -inase
interpret the selected entry as a standard entry type.

-octal, -oc
specifies that length is an octal number.

ACCESS REQUIRED

You need m permission on the directory containing the segment.

NOTES

The maximum length is the maximum size the segment can attain. Currently maximum length must be a multiple of 1024 words (one page).

If the new maximum length is less than the current length, you are asked if the segment should be truncated to the maximum length. If you answer "yes," the truncation takes place and the maximum length of the segment is set; if "no," no action is taken.

You can't set the maximum length of a mailbox or message segment unless the segment is empty.

set_max_length

EXAMPLES

The command line

```
! sml report -oc 10000
```

sets the maximum length of the segment report in your working directory to four pages.

The command line

```
! sml *.archive 16384
```

sets the maximum length of all two-component segments with a second component of archive in your working directory to 16 pages.

Name: set_mdir_account, smda

SYNTAX AS A COMMAND

```
smda path {User_id}
```

FUNCTION

sets the quota account of a master directory; used by the volume executive (the owner or manager of logical volumes).

ARGUMENTS

path
: is the pathname of the master directory whose quota account is to be changed.

User_id
: is the name (Person_id.Project_id) of the new quota account of the master directory. If omitted, your User_id is assumed.

ACCESS REQUIRED

You need e access on the logical volume containing the master directory. The volume need not be mounted.

NOTES

The quota for the master directory is returned to the old quota account and withdrawn from the new one, which must have sufficient quota to allow this.

Name: set_mdir_owner, smdo

SYNTAX AS A COMMAND

```
smdo path {User_id}
```

FUNCTION

sets the owner of a master directory (see the Programmer's Reference Manual).

ARGUMENTS

path
is the pathname of the master directory to be changed.

User_id
is the Person_id.Project_id of the new owner of the master directory. If omitted, your User_id is assumed.

ACCESS REQUIRED

You need e access on the logical volume containing the master directory. The volume need not be mounted.

Name: set_mdir_quota, smdq

SYNTAX AS A COMMAND

```
smdq path1 change1...{pathN changeN}
```

FUNCTION

sets the quota on a master directory (see the Programmer's Reference Manual).

ARGUMENTS

pathi
is the pathname of a master directory whose quota is to be changed.

changei
is the amount of quota, or the amount of quota change; you can specify it as follows:

```
+n  add n records of quota to pathi
-n  subtract n records of quota from pathi
 n  set the quota on pathi to n records
```

ACCESS REQUIRED

You must have m permission on the master directory and must be the owner of the master directory, be a volume administrator, or have the same quota account as the master directory.

NOTES

If the quota is being increased, the master directory's quota account must have sufficient volume quota to satisfy the request.

The quota of a master directory can never be zero, and it can never be set less than the current number of records being charged against the master directory.

Name: set_resource, setr

SYNTAX AS A COMMAND

```
setr type STR1...STRN {-control_args}
```

FUNCTION

modifies parameters of a resource.

ARGUMENTS

type
is a resource type defined in the Resource Type Description Table (RTDT).

STRi
is the unique identifying name of the particular resource being modified. If STR looks like a control argument (with a preceding hyphen), then use -name (-nm) before it.

CONTROL ARGUMENTS

-access_class accr, -acc accr
sets the initial AIM access class parameters, where accr is the access class range; you must supply -priv with it. If your authorization is within the access class range inclusive, you are allowed to read and write to the resource (provided you also meet other access requirements).

-acs_path path
specifies the pathname of the access control segment (ACS) for this resource. The ACS and the desired access control list set are not created by set_resource but by the accounting owner. If you give no -acs_path, the accounting owner of the resource is given rew access by default.

-alloc STR
sets the allocation state of the resource to free or allocated, where STR must be either "on" or "off"--on sets the allocation state to allocated, off to free. If you don't supply -alloc, the allocation state is free. (The allocation state flag is a convenience to you and is largely ignored by resource management.)

-attributes STR, -attr STR
specifies the desired values for the attributes of this resource.

-charge_type name, -crgtp name
specifies the name of the billing algorithm used to account for the use of this resource.

-comment STR, -com STR
specifies the desired value of the comment string for this resource.

-location STR, -loc STR
specifies a descriptive location for the resource, to aid the operator in locating the resource when it is stored in a special place (e.g., a vault, a different room, etc.); you must supply -priv with -location.

-lock STR
locks or unlocks the resource, where STR must be either "on" or "off"--on prevents any use of the resource, off allows its use. If you don't give -lock, the lock is off. You must supply -priv with -lock.

-priv
makes a privileged call to obtain the status of this resource (see "Access Required" below). If you are a privileged user (RCP Administrator), it allows you to set and change certain fields (including the acs_pathname) for this resource in the registry, thus effectively allowing yourself enough access to mount a tape for a write.

-release_lock STR, -rll STR
specifies whether the resource can be released by the owner or only by a privileged process (see "Access Required" below)--on resources can only be released by a privileged process, off resources by the owner. If you don't supply -release_lock, the resource can be released by the owner. You must give -priv with -release_lock.

ACCESS REQUIRED

You need write effective access to the resource named to modify its status; execute effective access to the resource named to modify protected attributes (only the accounting owner can modify the ACS path); and execute access to the rcp_admin_ gate to use -access_class, -lock, -location, -priv, and -release_lock.

NOTES

If you specify multiple resources and an error occurs in the modification of one of them, none are modified.

set_ring_brackets

Name: set_ring_brackets, srb

SYNTAX AS A COMMAND

```
srb path {ring_numbers}
```

FUNCTION

allows you to modify the ring brackets of a specified segment, multisegment file (MSF), data management (DM) file, or extended entry.

ARGUMENTS

path
: is the relative or absolute pathname of the segment, MSF, DM file, or extended entry whose ring brackets are to be modified.

ring_numbers
: are the numbers that represent the ring brackets of the segment, MSF, DM file, or extended entry. For a segment or MSF there are three ring brackets (rb1 rb2 rb3). The ring brackets must be in the allowable range 0 through 7 and must have the ordering

```
rb1 <= rb2 <= rb3
```

If you omit rb1, rb2, and rb3, they are set to your current validation level. The rb1 ring bracket is the number to be used as the first ring bracket of the segment; if omitted, you can't give rb2 and rb3 and rb1, rb2, and rb3 are set to your current validation level. The rb2 ring bracket is the number to be used as the second ring bracket of the segment; if omitted, you can't give rb3, and it is set, by default, to rb1. The rb3 ring bracket is the number to be used as the third ring bracket of the segment; if omitted, it is set to rb2.

For a DM file there are only two ring brackets (rb1 rb2). They have the same properties as rb1 and rb2 for segments.

For an extended entry the ring brackets you can give depend on the entry type (see describe_entry_type).

NOTES

Your process must have a validation level less than or equal to rb1. Ring brackets and validation levels are discussed in "Intraprocess Access Control" of the Programmer's Reference Manual.

set_search_paths

Name: set_search_paths, ssp

SYNTAX AS A COMMAND

```
ssp search_list {search_paths} {-control_arg}
```

FUNCTION

allows you 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. (See add_search_paths for a list of acceptable keywords.) If no search paths are specified, then the specified search list is set as if it were being initialized for the first time in your 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, with only minor restrictions, your dynamic linking search rules to suit your needs.

ARGUMENTS

path
is the pathname of a segment containing the ASCII representation of search rules. Search rules are absolute pathnames and any keyword in "List of Keywords," one search rule per line. If you supply no path, use -default.

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.) You can use the default keyword to obtain the site-defined default rules.

NOTES

You can give up to 21 rules. You can leave leading and trailing blanks, but not embedded blanks.

If you don't include the system libraries in the search rules, many standard commands cannot be found.

See also add_search_rules, delete_search_rules, get_system_search_rules, and print_search_rules.

Name: set__severity__indicator, ssi

SYNTAX AS A COMMAND

```
ssi name value
```

FUNCTION

allows you to set severity indicators from command level (see the severity command).

ARGUMENTS

name
is the name of the severity indicator to be set.

value
is the decimal integer to be used as the new value of the severity indicator.

Name: set__system__storage

SYNTAX AS A COMMAND

```
set_system_storage {virtual_pointer} {-control_args}
```

FUNCTION

establishes an area as the storage region in which normal system allocations are performed.

ARGUMENTS

virtual_pointer
is a virtual pointer to an initialized area (see Section 1).

CONTROL ARGUMENTS

-create
creates and initializes a system-free segment in your process directory.

-system
specifies the area used for linkage sections.

NOTES

Specify either virtual_pointer or the control arguments.

To initialize or create an area, see the create_area command. Set up the area as either zero_on_free or zero_on_alloc. Make the area specified extensible.

EXAMPLES

The command line

```
! set_system_storage free_$free_
```

places objects in the segment whose reference name is free_ at the offset whose entry point name is free_.

The command line

```
! set_system_storage my_seg$
```

uses the segment whose reference name is my_seg. The area is assumed to be at an offset of 0 in the segment. The segment must already exist with the reference name my_seg and must be initialized as an area.

The command line

```
! set_system_storage my_seg
```

uses the segment whose relative pathname is my_seg. The segment must already exist.

Name: set__time__default, std

SYNTAX AS A COMMAND

```
std key value {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[std key value {-control_arg}]
```

FUNCTION

sets a default date/time value for the process. As an active function, it returns "true" if the action requested was successful, "false" otherwise.

ARGUMENTS

key
is a keyword representing the default to be set.

value
is a value to become the new default. If the value is "-system" (-sys), the system default is used; if it is -pop, a remembered value is used, saved by an earlier setting with -push. It is an error if you have used no -push before.

CONTROL ARGUMENTS

-push
saves the current value of the default before setting to the new value.

LIST OF KEYS

date
sets the process default date. The value must be acceptable to date_time_$format (see "Notes").

date_time
sets the process default date_time. The value must be acceptable to date_time_$format (see "Notes").

debug, db
sets the process date/time debugging switch. The value can be "on" or "true," or "off" or "false." When debugging is enabled, convert_date_to_binary_ displays a description of time strings as they are parsed, and identifies the exact location of any error in the time string. The system default value is off.

language, lang
sets the process default language. The language name can be in any of the languages known to the date/time system. To print a list of acceptable language values, type "display_time_info -language".

time
sets the process default time. The value must be acceptable to date_time_$format (see "Notes").

zone
sets the process default zone. The zone abbreviation can be in any of the languages known to the date/time system. To print a list of acceptable zone values, type "display_time_info -zone" or "display_time_info -map".

NOTES

The named format strings acceptable to date_time_$format are described in Section 1, under "Time Format." The names "date", "date_time", and "time" are not allowed in this context.

Name: set_ttt_path

SYNTAX AS A COMMAND

```
set_ttt_path {path} {-control_arg}
```

FUNCTION

changes the pathname of the terminal type table (TTT) associated with your process.

ARGUMENTS

path
is the pathname of the TTT. If you don't give it, you must supply the control argument.

CONTROL ARGUMENTS

-reset, -rs
resets the TTT pathname to its default value: >system_control_1>ttt.

Name: set_tty, stty

SYNTAX AS A COMMAND

```
stty {-control_args}
```

FUNCTION

modifies the terminal type associated with your 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. 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.

-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.

-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. 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. 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.

-print_frame, -pr_fr
prints the framing 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 your terminal type 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.

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. 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 your 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 your 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 you are 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 when you are 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`
can be used to prevent unimportant modes from causing errors if they cannot be set. Force is positional: any mode specified after it is affected by it. For instance, in the command line

```
! stty -modes fulldpx,force,replay,^force,polite
```

if the fulldpx mode cannot be set, the command fails; no error message is returned if replay cannot be set; and an error occurs if polite cannot 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 your 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 you type 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 you are typing input until the carriage is at the left margin, unless you allow 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, you may wish to disable the erase character when using replacement canonicalization. This may be accomplished by the command line

```
set_tty -edit \400
```

Name: set_user_storage

SYNTAX AS A COMMAND

```
set_user_storage {virtual_pointer} {-control_arg}
```

FUNCTION

establishes an area as the storage region in which normal user allocations are performed. These allocations include FORTRAN common blocks and PL/I external variables whose names do not contain dollar signs.

ARGUMENTS

virtual_pointer
: is a virtual pointer to an initialized area (see Section 1 for a description of virtual pointers).

CONTROL ARGUMENTS

-create
creates (and initializes) a system-free segment in your process directory.

-system
specifies the area used for linkage sections.

NOTES

The control arguments must be given only if virtual_ptr is not specified and vice versa.

To initialize or create an area, refer to the description of the create_area command. The area must be set up as either zero_on_free or zero_on_alloc. It is recommended that the area specified be extensible.

EXAMPLES

The command line

```
! set_user_storage free_$free_
```

places objects in the segment whose reference name is free_ at the offset whose entry point name is free_.

The command line

```
! set_user_storage my_seg$
```

uses the segment whose reference name is my_seg. The area is assumed to be at an offset of 0 in the segment. The segment must already exist with the reference name my_seg and must be initialized as an area. The command line

```
! set_user_storage my_seg
```

uses the segment whose relative pathname is my_seg. The segment must already exist.

Name: set_volume_quota, svq

SYNTAX AS A COMMAND

```
svq logical_volume change {account}
```

FUNCTION

sets a quota account's volume quota on a logical volume; used by the volume executive (the owner or manager of logical volumes).

ARGUMENTS

logical_volume
is the name of the logical volume for which quota is to be set.

change
is the amount of quota, or the amount of quota change; you can specify it as follows:

```
+n  add n records to the quota
-n  subtract n records from the quota
 n  set the quota to n records
```

account
is the name of the quota account (in the form Person_id.Project_id) to be set. If omitted, your User_id is used.

ACCESS REQUIRED

To use this command you must have e access to the logical volume. It is not necessary that the volume be mounted.

NOTES

If you set the volume quota less than the quota account's current quota used, the quota is changed as directed, but a warning message is printed.

Name: severity

SYNTAX AS A COMMAND

```
severity command_name {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[severity command_name {-control_arg}]
```

FUNCTION

returns a number representing the severity of the most recent translation or invocation of the specified command.

ARGUMENTS

command_name
: is the name of any command that provides a severity indicator (see "Notes on Severity Indicators").

CONTROL ARGUMENTS

-default XX, -dft XX
: specifies the default value XX to be returned if referencing a nonexistent or uninitialized severity indicator.

NOTES

If the command specified has not defined a severity indicator or you haven't invoked it yet, an error is indicated unless you give -default.

The fortran command only supports the severity active function if your site is using the new FORTRAN compiler as its standard FORTRAN compiler.

NOTES ON SEVERITY INDICATORS

Severity indicators are set by system commands, user commands, or from command level by the set_severity_indicator command. They are single-precision (one word) binary values. The meaning of the value depends on the particular command. System commands supporting severity indicators explain their values under "Severity" in the description of the command. Under program control you can define and set severity indicators by assigning an integer value to an external variable.

EXAMPLES

Examples for PL/I and FORTRAN follow.

PL/I:

```
x: proc;
    .
    .
    .
  dcl x_severity_ fixed bin (35) ext static;
    .
    .
    .
  end;
```

FORTRAN:

```
common /y_severity_/ idum
   idum = 5
```

Name: shortest_path

SYNTAX AS A COMMAND

```
shortest_path path {entry {component}}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[shortest_path path {entry {component}}]
```

FUNCTION

returns the shortest absolute pathname represented by the argument if you give one argument, or the shortest absolute pathname of the archive component or the entry in the directory specified by path if you give two or three arguments. The shortest name is determined by using the shortest of each of the names on each component in the path.

ARGUMENTS

path
: is the pathname to be expanded and returned if you don't use entry; otherwise this is the pathname of the directory to be used in the returned pathname.

entry
: is the entryname to be used in the returned pathname.

component
: is the archive component name to be used in the returned pathname.

NOTES

Since the pathname returned by path is in quotes, the command processor treats it as a single argument regardless of special characters in the name.

When more than one name qualifies as the shortest name for a directory, shorthest_path tries to select the name containing all lowercase characters. If several names still qualify, they are compared to the primary name of the directory: the first name found with the same first character as the primary name is chosen; this comparison is case independent.

EXAMPLES

Assume the working directory is >udd>Proj>JKeats; then,

```
! shortest_path start_up.ec
  >udd>Proj>JKeats>start_up.ec

! shortest_path >user_dir_dir>Multics>Library>Source
    >bound_command_demos_.s::program.pl1
  >udd>m>lib>s>bound_command_demos_.s::program.pl1

! shortest_path <s>bound_expand_path_.s.archive
  >udd>Proj>JKeats>s>bound_expand_path_.s.archive
```

*

Name: signal

SYNTAX AS A COMMAND

```
signal CONDITION_NAME {-control_args}
```

FUNCTION

signals Multics conditions, allowing you to specify some information to be associated with the condition. The result of a condition signal depends on your program or the system program handling the condition signal.

The descriptions that follow assume that the signal is handled by the default unclaimed signal handler (default_error_handler_$wall). Any messages described are sent over the error_output switch.

ARGUMENTS

CONDITION_NAME
is the name of the condition to signal. It can't contain embedded white space because condition names are only significant to the first space character. It can't be longer than 256 characters.

CONTROL ARGUMENTS

-cant_restart
sets the cant_restart flag for this signal. The default handler establishes a new listener level after printing a message and refuses to accept the start command. (See "Notes.")

-code ET_CODE_NAME
associates the error table code name ET_CODE_NAME with this signal. It must be a virtual pointer to an error table acceptabe to cv_ptr_. If you omit the segment name portion of the virtual pointer, error_table_ is assumed. The text message defined for this error table code is printed if an error message is printed; thus an ET_CODE_NAME of noentry is interpreted as error_table$noentry, not as a pointer to noentry|0.

-default_restart
sets the default_restart flag for this signal. The default handler prints a message and restarts execution.

-info_string INFO_MESSAGE
associates the string INFO_MESSAGE with this signal. If an error message is printed, this string is also printed. Enclose it in quotes if it contains white space or special characters. The string can't be longer than 256 characters.

-quiet_restart
sets the quiet_restart flag for this signal. The default handler restarts execution without printing a message.

-support_signal
sets the support_signal flag for this signal. This indicates that the error is being signaled on behalf of another procedure and should only be used when your handler is present on the stack that expects it.

NOTES

Don't use this command with any of the system conditions defined in the Programmer's Reference Manual or with PL/I language conditions because they require other associated information that you can't specify with signal and unpredictable results may occur.

You can use the on command to handle signals produced with this command.

The default handler deals with all condition signals that are otherwise unhandled by your programs or system programs on the stack. If you select none of -cant_restart, -default_restart, or -quiet_restart, the default handler prints the error message described below, and establishes a new listener level. If you type "start" at this point, execution continues. If the command is executed in an exec_com and you type start, execution continues with the next command in the exec_com.

The default message printed for a condition signaled is of the form

```
Error: CONDITION_NAME condition by signal$signal|octalnumber
ERROR_TABLE_MESSAGE
INFO_STRING_MESSAGE
```

If you give no -info_string, the INFO_STRING_MESSAGE line is omitted. If you don't chose -code, the ERROR_TABLE_MESSAGE line is omitted.

Name: sort_seg, ss

SYNTAX AS A COMMAND

```
ss path {-control_args}
```

FUNCTION

orders the contents of a segment according to the ASCII collating sequence.

ARGUMENTS

path
specifies the pathname of an input segment. The star convention is not allowed.

CONTROL ARGUMENTS

The control arguments accepted by the command are described below, organized by function.

SORT UNITS
-block, -bk
-delimiter, -dm

HANDLING DUPLICATES
-duplicates, -dup
-only_duplicates, -odup
-only_duplicate_keys, -odupk
-only_unique, -ouq
-only_unique_keys, -ouqk
-unique, -uq
-unique_keys, -uqk

OUTPUT FILES
-output_file, -of
-replace, -rp

SORT ORDER
-all, -a
-ascending, -asc
-case sensitive, -cs
-character, -ch
-descending, -dsc
-field, -fl
-integer, -int
-non_case_sensitive, -ncs
-numeric, -num

Sort Units

-block N, -bk N
makes the sort unit a block of N strings where N must be a positive integer. The default for N is 1 (see "Examples" below).

-delimiter /REGEXP/, -dm /REGEXP/
uses REGEXP as a regular expression as the string delimiter. Strings to be sorted are delimited by the characters that match the regular expression. (See the qedx command.)

-delimiter L, -dm L
makes each L characters of the input segment a delimited string, where L is a positive integer. This essentially divides the input into character strings of length L.

-delimiter {-string} STR, -dm {-str} STR
uses STR concatenated with a newline character as the string delimiter. The character STR can be any sequence of ASCII characters. It can be preceded by -string (-str) to distinguish it from an integer or a regular expression. The default is a single newliine character (see "Examples").

Handling Duplicates

-duplicates, -dup
retains duplicate sort units in the sorted results. (Default)

-only_duplicates, -odup
only sort units that occur more than once in the segment appear in the sorted results. One unit from each set of duplicate sort units is placed in the output segment, sorted.

-only_duplicate_keys, -odupk
only sort units that have duplicate sort fields appear in the sorted results. All such units having duplicate sort fields are placed in the output segment since the nonsort field portions of the units may differ.

-only_unique, -ouq
only sort units that are unique appear in the sorted results. Whenever a set of duplicate units are found, they are removed from the output segment.

-only_unique_keys, -ouqk
only sort units that have unique sort fields appear in the sorted results. All units having duplicate sort fields are removed from the output segment.

-unique, -uq
deletes duplicate sort units from the sorted results. For each set of duplicate sort units only the first appears in the sorted results, along with nonduplicate sort units.

-unique_keys, -uqk
deletes sort units having duplicate sort fields from the sorted results. For each set of sort units having duplicate fields only the first appears in the sorted results, along with nonduplicate sort units.

Output Files

-output_file path, -of path
places the sorted units in a segment whose pathname is path. You can use the equal convention.

-replace, -rp
replaces the original contents of the input segment with the sorted units. The default is to ask you if the input segment should be replaced with its sorted contents.

Sort Order

-all, -a
makes the primary (and only) sort field be the entire sort unit; i.e., the entire sort unit is considered when sorting. (Default)

-ascending, -asc
makes the sort in ascending order, according to the ASCII collating sequence. (Default)

-case_sensitive, -cs
makes the sort by comparing sort fields without translating letters to lowercase. (Default)

-character, -ch
makes the sort based on the character representation of the sort field. (Default)

-descending, -dsc
makes the sort in descending order, according to the ASCII collating sequence.

-field field_specs, -fl field_specs
specifies the field(s) to be used when comparing two sort units. This allows units to be sorted based upon comparison of only a part of each sort unit. You can use multiple -fields to specify more than one field. (See "Syntax of Field Specifications" below.)

-integer, -int
makes the sort by converting the sort field to fixed binary (71,0) integers when comparing one sort unit with another (see "Notes" below").

-non_case_sensitive, -ncs
makes the sort by translating letters in the sort fields to lowercase when comparing one sort unit with another. The actual sorted results remain unchanged.

-numeric, -num
makes the sort by converting the sort field to float decimal (59) numbers when comparing one sort unit with another (see "Notes").

SYNTAX OF FIELD SPECIFICATIONS

The field_spec operands of -field define the fields within each sort unit by which the unit is sorted. The first field_spec defines the primary sort field, the second, a secondary sort field, and so forth.

Each field_spec consists of a field start location, field length, and optional sorting controls, which must appear in the following order:

```
field_start field_length {sort_controls}
```

LIST OF field_start FORMATS

You can give the field start location in one of the following formats:

S
a positive integer, giving the character position of the start of the field in the sort unit (e.g., 1 if the field begins at the first character). If the sort unit contains fewer than S characters, then the unit is sorted as if space characters appeared in the sort field.

-from S, -fm S
where S is a positive integer giving the character position of the start of the field in the sort unit.

-from STR, -fm STR
where STR is a character string that identifies the beginning of the sort field. The field begins with the first character of the sort unit that follows STR. If STR does not appear in the sort unit, then the unit is sorted as if the sort field contains space characters.

-from /REGEXP/, -fm /REGEXP/
where REGEXP is a regular expression that identifies the beginning of the sort field. The field begins with the first character of the sort unit that follows the part of the sort unit matching REGEXP (see the qedx command). If no match for REGEXP is found in the sort unit, then the unit is sorted as if the sort field contains space characters.

-from -string STR, -fm -str STR
treats STR as a character string that identifies the beginning of the sort field, even though STR may look like an integer or a regular expression. For example,

```
-from -string 25
```

identifies a sort field that begins with the character following "25" in the sort unit.

LIST OF field_length FORMATS

The sort field length can be specified in one of the following ways:

L
a positive integer, giving the length of the sort field in characters. If the sort unit is too short to hold a sort field of L characters (i.e., if the number of characters from the first character of the sort field to the end of the sort unit is less than L), then the unit is sorted as if the field were extended on the right with space characters to a length of L characters. Alternately, L can be -1 to indicate that the remainder of the sort unit is to be used as the sort field.

-for L
where L is a positive integer giving the length of the sort field in characters, or -1 to use the remainder of the sort unit as the sort field.

-to E
where E is a positive integer giving the character position of the end of the sort field in the sort unit (e.g., 5 if the field stops after the fifth character of the sort unit). If the sort unit contains fewer then E characters, then the unit is sorted as if space characters were added on the right to extend the unit to E characters.

-to STR
where STR is a character string that identifies the end of the sort field. The field ends with the first character of the sort unit preceding STR. If STR does not appear in the sort unit after the starting position of the sort field, then the unit is sorted as if space characters appeared in the sort field.

-to /REGEXP/
where REGEXP is a regular expression that identifies the end of the sort field. The field ends with the first character of the sort unit that precedes the part of the sort unit matching REGEXP (see the qedx command). If no match for REGEXP is found in the sort unit after the starting position of the sort field, then the unit is sorted as if space characters appeared in the sort field.

-to -string STR
treats STR as a character string that identifies the end of the sort field, even though STR may look like an integer or a regular expression.

Note that when -to is used to indicate the end of the field, then sort_seg examines all sort units to determine the length of the longest instance of this sort field in any sort unit. It then sort units as if the sort field in each unit were extended on the right with space characters to the length of the longest sort field instance.

LIST OF sort_controls

The sort controls may be one from each of the following sets of arguments. If no sort control is given, then the default is specified by the corresponding control argument (-ascending or -descending, -case_sensitive or -non_case_sensitive -character or -integer or -numeric).

ascending, asc
sorts units with this field in ascending order. This sort control is incompatible with descending.

descending, dsc
sorts units with this field in descending order. This sort control and ascending are mutually exclusive.

case_sensitive, cs
sorts units by treating uppercase letters in this field as being different from lowercase letters. This sort control is incompatible with non_case_sensitive.

non_case_sensitive, ncs
sorts units by translating this field to lowercase. This sort control is incompatible with case_sensitive.

character, ch
sorts units with this field by the character representation. It is incompatible with integer or numeric.

integer, int
sorts unit with this field by converting the character representation to its integer value (fixed binary (71,0)). This sort control is incompatible with character or numeric.

numeric, num
sorts units with this field by converting the character representaion to its numeric value (float decimal (59)). It is incompatible with character or integer.

NOTES

Using the control arguments, the segment is broken down into separate sort units, which are strings or blocks of strings. A string can comprise one or more lines. These sort units are then sorted, and the ordered units either replace the original segment or are placed in a new segment.

If the command is invoked without any control arguments, -replace, -ascending, -all, -character and -delimiter are assumed, and the default delimiter of a newline character is used; that is, sort_seg, when invoked with path as the only argument, sorts the lines of that segment as character strings in ascending ASCII collating sequence, replacing the original segment with the sorted result. As a safety measure, the following question is asked when -replace is not supplied:

Do you really want to sort the contents of PATH?

This helps avoid accidental sorting of segments.

The start position of a sort field is calculated relative to the beginning of a sort unit. If the blocking factor is N = 1, the start position is calculated corresponding to the beginning of a string. If the blocking factor is N > 1, the start position is calculated relative to the beginning of the first string of a block. When calculating field specifications within a sort unit of N > 1 strings (blocking factor N > 1), string delimiters internal to the sort unit should not be considered (see "Examples").

Sort fields/units of unequal length are compared by assuming the shorter field/unit to be padded on the right with space characters, immediately following the rightmost character. If a field/unit contains nonprinting graphic characters (such as BS, HT, NL, VT, FF, CR, etc.), which precede the space character in the ASCII collating sequence, they are sorted accordingly, with sometimes unexpected results. The string delimiter is never considered when padding (see "Examples").

The numeric sort mode converts the sort field character string to a float decimal (59) value for sorting purposes. Similarly, the integer sort mode converts the sort field character string to a fixed bin (71,0) value. The character string representation must be acceptable to the PL/I or FORTRAN language conversion rules. The actual sort field remains unchanged in the sorted results.

If characters are detected in the input segment following the final delimited sort unit, they are ignored for the purposes of sorting but appear in the sorted output immediately following the final delimited sort unit. An error message specifies the location of the first nondelimited character.

A maximum of 261,119 units can be sorted. The sort is stable; i.e., duplicate units appear in the same order in the sorted segment as in the original segment.

The input segment is sorted using temporary segments in the process directory. If -output_file is given and path is the pathname of an already-existing segment, its contents are destroyed upon beginning the sort. If the sorted results are to replace the original contents of the input file, that replacement does not occur until the last possible moment.

The determination of whether or not a sort unit is to be deleted (see -unique) is independent of sort field specifications; i.e., given a number of nonidentical sort units that contain identical sort fields, all the units do appear in the sorted results.

The following groups of control arguments are mutually exclusive with other control arguments in the same group. If more than one from a group is given in a single command, the last one given in the command overrides the others.

-all, -field

-ascending, -descending

-case_sensitive, -non_case_sensitive

-character, -integer, -numeric

-duplicates, -only_duplicates, -only_duplicate_keys, -unique, -unique_keys

-replace, -output_file

In addition, if -delimiter is used several times, the final specification overrides the previous ones.

EXAMPLES

Suppose a segment contains the following lines (nl stands for the ASCII newline character and # stands for the ASCII space character):

```
ABCDEFGHXYnl
ABCDEFXYnl
ABCDEFGHIJXYnl
ABCXYnl
```

The display below shows how sort_seg sorts the contents of this segment, according to the arguments specified in the first column.

these arguments	define these sort units	sorted on these fields	giving these results
-dm XY	ABCDEFGH	ABCDEFGH##	ABCXYnl
	ABCDEF	ABCDEF####	ABCDEFXYnl
	ABCDEFGHIJ	ABCDEFGHIJ	ABCDEFGHXYnl
	ABC	ABC#######	ABCDEFGHIJXYnl
-bk 2	ABCDEFGHABCDEF	ABCDEFGHABCDEF	ABCDEFGHXYnl
-dm XY	ABCDEFGHIJABC	ABCDEFGHIJABC#	ABCDEFXYnl
			ABCDEFGHIJXYnl
			ABCXYnl
-fl 6 4	ABCDEFGHXY	FGHX	ABCXYnl
	ABCDEFXY	FXY#	ABCDEFGHIJXYnl
	ABCDEFGHIJXY	FGHI	ABCDEFGHXYnl
	ABCXY	####	ABCDEFXYnl
		first second	
-fl 1 4 7 2	ABCDEFGHXY	ABCD GH	ABCDEFGHXYnl
	ABCDEFXY	ABCD XY	ABCDEFGHIJXYnl
	ABCDEFGHIJXY	ABCD GH	ABCDEFXYnl
	ABCXY	ABCX ##	ABCXYnl
-dm Y	ABCDEFGHXABCDEFX	FGHX DE	ABCDEFGHIJXYnl
-bk 2	ABCDEFGHIJXABCX	FGHI DE	ABCXYnl
-fl 6 4 4 2			
			ABCDEFGHIJXABCX
			ABCDEFXYnl
		first second	
-fl 6 4 dsc	ABCDEFGHXY	FGHX CDE	ABCDEFXYnl
3 3 asc	ABCDEFXY	FXY# CDE	ABCDEFGHXYnl
	ABCDEFGHIJXY	FGHI CDE	ABCDEFGHIJXYnl
	ABCXY	#### CXY	ABCXYnl
-fl 1 3	ABCDEFGH	ABC	ABCDEFGHXYnl
-unique_key	ABCDEF	ABC	
-dm XY	ABCDEFGHIJ	ABC	
	ABC	ABC	
		first second	
-fl 1 3 5 2	ABCDEFGH	ABC EF	ABCDEFGHXYnl
-odupk	ABCDEF	ABC EF	ABCDEFXYnl
-dm XY	ABCDEFGHIJ	ABC EF	ABCDEFGHIJXYnl
	ABC	ABC ##	

Name: sort_strings, sstr

SYNTAX AS A COMMAND

```
sstr {-control_args} strings
```

SYNTAX AS AN ACTIVE FUNCTION

```
[sstr {-control_args} strings]
```

FUNCTION

orders the argument strings according to the ASCII collating sequence.

ARGUMENTS

strings
are the strings to be sorted. All arguments following the first strings are treated as strings. You can use -string to identify a first string that looks like a control argument or to separate a numeric string from operands of -field.

CONTROL ARGUMENTS

The control arguments accepted by the command are described below, organized by function.

SORT UNITS
-block, -bk

HANDLING DUPLICATES
-duplicates, -dup
-only_duplicates, -odup
-only_duplicate_keys, -odupk
-only_unique, -ouq
-only_unique_keys, -ouqk
-unique, -uq
-unique_keys, -uqk

INPUT STRINGS
-string, -str

SORT ORDER
-all, -a
-ascending, -asc
-case_sensitive, -cs
-character, -ch
-descending, -dsc
-field, -fl
-integer, -int
-non_case_sensitive, -ncs
-numeric, -num

Sort Units

-block N, -bk N
makes the sort unit a block of N strings, where N must be a positive integer (see "Examples" below). (Default: one string)

Handling Duplicates

-duplicates, -dup
retains duplicate sort units in the sorted results. (Default)

-only_duplicates, -odup
only sort units that occur more than once in the input appear in the sorted results. One unit from each set of duplicate sort units is placed in the return value, sorted.

-only_duplicate_keys, -odupk
only sort units that have duplicate sort fields appear in the sorted results. All such units having duplicate sort fields are placed in the return value since the nonsort field portions of the units may differ.

-only_unique, -ouq
only sort units that are unique appear in the sorted results. Whenever a set of duplicate units are found, they are removed from the return value.

-only_unique_keys, -ouqk
only sort units that have unique sort fields appear in the sorted results. All units having duplicate sort fields are removed from the return value.

-unique, -uq
deletes duplicate sort units from the sorted results. For each set of duplicate sort units only the first appears in the sorted results, along with nonduplicate sort units.

-unique_keys, -uqk
deletes sort units having duplicate sort fields from the sorted results. For each set of sort units having duplicate fields only the first appears in the sorted results, along with nonduplicate sort units.

Input Strings

-string strings, -str strings
identifies the strings that follow as the strings to be sorted. All remaining arguments are treated as input strings.

Sort Order

-all, -a
makes the primary (and only) sort field be the entire sort unit; i.e., each string is sorted based upon its entire value, rather than being split into one or more sort fields. (Default)

-ascending, -asc
returns the sorted results in ascending order. (Default)

-case_sensitive, -cs
makes the sort by comparing sort fields without translating letters to lowercase. (Default)

-character, -ch
makes the sort based on the character representation of the sort field. (Default)

-descending, -dsc
returns the sorted results in descending order.

-field field_specs, -fl field_specs
specifies the field(s) to be used when comparing two sort units. This allows units to be sorted based upon comparison of only a part of each sort unit. You can use multiple -fields to specify more than one field. (See "Syntax of Field Specifications" below.)

-integer, -int
makes the sort by converting the sort field to fixed binary (71,0) integers when comparing one sort unit with another (see "Notes" below).

-non_case_sensitive, -ncs
makes the sort by translating letters in the sort fields to lowercase when comparing one sort unit with another. The actual sorted results remain unchanged.

-numeric, -num
makes the sort by converting the sort field to float decimal (59) numbers when comparing one sort unit with another (see "Notes").

SYNTAX OF FIELD SPECIFICATIONS

The field_spec operands of -field define the fields within each sort unit by which the unit is sorted. The first field_spec defines the primary sort field, the second, a secondary sort field, and so forth.

Each field_spec consists of a field start location, field length, and optional sorting controls, which must appear in the following order:

```
field_start field_length {sort_controls}
```

LIST OF field_start FORMATS

You can give the field start location in one of the following formats:

S
a positive integer, giving the character position of the start of the field in the sort unit (e.g., 1 if the field begins at the first character). If the sort unit contains fewer than S characters, then the unit is sorted as if space characters appeared in the sort field.

-from S, -fm S
where S is a positive integer giving the character position of the start of the field in the sort unit.

-from STR, -fm STR
where STR is a character string that identifies the beginning of the sort field. The field begins with the first character of the sort unit that follows STR. If STR does not appear in the sort unit, then the unit is sorted as if the sort field contained space characters.

-from /REGEXP/, -fm /REGEXP/
where REGEXP is a regular expression that identifies the beginning of the sort field. The field begins with the first character of the sort unit that follows the part of the sort unit matching REGEXP (see the qedx command). If no match for REGEXP is found in the sort unit, then the unit is sorted as if the sort field contained space characters.

-from -string STR, -fm -str STR
treats STR as a character string that identifies the beginning of the sort field, even though STR may look like an integer or a regular expression. For example,

```
-from -string 25
```

identifies a sort field that begins with the character following 25 in the sort unit.

LIST OF FIELD_LENGTH FORMATS

You can specify the sort field length in one of the following ways:

L
a positive integer, giving the length of the sort field in characters. If the sort unit is too short to hold a sort field of L characters (i.e., if the number of characters from the first character of the sort field to the end of the sort unit is less than L), then the unit is sorted as if the field were extended on the right with space characters to a length of L characters. Alternately, L can be -1 to indicate that the remainder of the sort unit is to be used as the sort field.

-for L
where L is a positive integer giving the length of the sort field in characters, or -1 to use the remainder of the sort unit as the sort field.

-to E
where E is a positive integer giving the character position of the end of the sort field in the sort unit (e.g., 5 if the field stops after the fifth character of the sort unit). If the sort unit contains fewer then E characters, then the unit is sorted as if space characters were added on the right to extend the unit to E characters.

-to STR
where STR is a character string that identifies the end of the sort field. The field ends with the first character of the sort unit preceding STR. If STR does not appear in the sort unit after the starting position of the sort field, then the unit is sorted as if space characters appeared in the sort field.

-to /REGEXP/
where REGEXP is a regular expression that identifies the end of the sort field. The field ends with the first character of the sort unit that precedes the part of the sort unit matching REGEXP (see the qedx command). If no match for REGEXP is found in the sort unit after the starting position of the sort field, then the unit is sorted as if space characters appeared in the sort field.

-to -string STR
treats STR as a character string that identifies the end of the sort field, even though STR may look like an integer or a regular expression.

Note that when you use -to to indicate the end of the field, then sort_strings examines all sort units to determine the length of the longest instance of this sort field in any sort unit; it then sort units as if the sort field in each unit were extended on the right with space characters to the length of the longest sort field instance.

LIST OF SORT_CONTROLS

The sort controls may be one from each of the following three sets of arguments; the arguments within each set are incompatible with each other. If you give none, then the default is specified by the corresponding control argument.

ascending, asc
sorts units with this field in ascending order.

descending, dsc
sorts units with this field in descending order.

case_sensitive, cs
sorts units by treating uppercase letters in this field as being different from lowercase letters.

non_case_sensitive, ncs
sorts units by translating this field to lowercase.

character, ch
sorts units with this field by the character representation.

integer, int
sorts unit with this field by converting the character representation to its integer value (fixed binary (71,0)).

numeric, num
sorts units with this field by converting the character representaion to its numeric value (float decimal (59)).

NOTES

Using the control arguments, each string (or group of strings if you supply -block) is treated as a separate sort unit. These sort units are then sorted, and the ordered units are printed or returned as the active function return value.

If you invoke sort_strings without any control arguments, -ascending, -all, and -character are assumed.

The start position of a sort field is calculated relative to the beginning of a sort unit. If the blocking factor is N = 1, the start position is calculated corresponding to the beginning of a string. If the blocking factor is N > 1, the start position is calculated relative to the beginning of the first string of a block. When calculating field specifications within a sort unit of N > 1 strings (blocking factor N > 1), you should not consider string delimiters internal to the sort unit (see "Examples"). Each group of N strings is concatenated without intervening spaces to form the sort unit.

Sort fields/units of unequal length are compared by assuming the shorter field/unit to be padded on the right with space characters, immediately following the rightmost character. If a field/unit contains nonprinting graphic characters (such as BS, HT, NL, VT, FF, CR, etc.), which precede the space character in the ASCII collating sequence, they are sorted accordingly, with unexpected results sometimes.

The numeric sort mode converts the sort field character string to a float decimal (59) value for sorting purposes. Similarly, the integer sort mode converts the sort field character string to a fixed binary (71,0) value. The character string representation must be acceptable to the PL/I or FORTRAN language conversion rules. The actual sort field remains unchanged in the sorted results.

You can sort a maximum of 261,119 units. The sort is stable; i.e., duplicate units appear in the same order in the sorted results as in the original input.

The input strings are sorted using temporary segments in the process directory.

The determination of whether or not a sort unit is to be deleted (see -unique) is independent of sort field specifications; i.e., given a number of nonidentical sort units that contain identical sort fields. all the units do appear in the sorted results.

The following groups have control arguments that are mutually exclusive with each other. If you provide more than one from a group in a single command, the last one given in the command overrides the others.

-all, -field

-ascending, -descending

-case_sensitive, -non_case_sensitive

-character, -integer, -numeric

-duplicates, -only_duplicates, -only_duplicate_keys,
-unique, -unique_keys

EXAMPLES

```
! sstr -str [segs **.pl1]
```

sorts the list of pl1 segments in the current directory into alphabetical order.

```
! sstr [index_set 1 20 2]
  1 11 13 15 17 19 3 5 7 9
```

sorts the numeric strings in ASCII collating sequence. Sorted in integer sequence, the strings are:

```
! sstr -integer [index_set 1 20 2]
  1 3 5 7 9 11 13 15 17 19
```

Sorting on the second character of each string gives

```
! sstr -integer -fl 2 1 -str [index_set 1 20 2]
  1 3 5 7 9 11 13 15 17 19
```

-str is included to separate the numeric operands of -field from the strings to be sorted. Reversing the sort order gives:

```
! sstr -dsc -integer -fl 2 1 -str [index_set 1 20 2]
  19 17 15 13 11 1 3 5 7 9
```

Values from 11 through 19 are sorted on their second digit. Values from 1 through 9 have no second digit and are extended on the right with a space that is converted to the integer 0. Since they all have the same sort field value, their original order is maintained.

You can block strings together for sorting purposes

```
! sstr -block (1 2 3) xyz uvw def bcd abc def ghi
  abc bcd def def ghi uvw xyz
  abc def def bcd ghi xyz uvw
  bcd abc def ghi xyz uvw def
```

Name: start, sr

SYNTAX AS A COMMAND

```
sr {-control_arg}
```

FUNCTION

is used after you have issued the quit signal to resume execution of your process from the point of interruption.

CONTROL ARGUMENTS

-no_restore, -nr
    does not restore the standard I/O attachments (see "Notes").

NOTES

You can also use start to resume execution after an unclaimed signal, provided that the condition that caused that signal either is innocuous or has been corrected. This command 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 you give -no_restore.

You can issue start at any time after a quit signal as long as you haven't given a release command.

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, multisegment files (MSFs), data management (DM) files, directories, and links for which you want status information. The default pathname is your working directory (-wd). This argument can be "-nonstandard_names (-nsn) STR" to specify a nonstandard segment name, such as one beginning with a minus sign or containing any of the nonstandard characters "<, *, ?, =, %, $, ., :, , ::". In this case STR must be in the working directory. The star convention is allowed but does not apply to STR. You can't use the star convention in the active function.

CONTROL ARGUMENTS

You can use the following control arguments with any type of entry; they can appear anywhere on the line after the command name and are in effect for the whole line. Give at least one control argument in the active function.

-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 about the ultimate link in the chain for null links and links to null links. It 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.

-entry_type, -ettp, -type, -tp
prints the type of entry, which can be one of the standard types (segment, MSF, DM file, directory, or link) or one of the available extended types (e.g., mailbox).

-interpret_as_extended_entry, -inaee
interprets the selected entries as extended entry types. (Default)

-interpret_as_standard_entry, -inase
interpret the selected entries as standard entry types.

-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)

-nonstandard_names, -nsn
specifies a nonstandard segment name, such as one beginning with a minus sign or containing any of the nonstandard characters "<, *, ?, =, %, $, ., :, ,::". In this case STR must be in the working directory.

-primary, -pri
prints the primary name on the entry.

-segment, -sm
selects segments when using the star convention.

-select_entry_type STR, -slet STR
selects entries of the types specified by STR, which is a comma-delimited list of file system entry types. Use the list_entry_types command to obtain a list of valid entry type values.

-switch SW_NAME
prints whether the specified SW_NAME is on or off, where SW_NAME is the name of a switch. Valid switch names are copy, complete_volume_dump, damaged, incremental_volume_dump, safety, synchronized or any valid extended entry type switch name.

LIST OF TYPE-SPECIFIC CONTROL ARGUMENTS

You can use the following control arguments only for segments, MSFs, and directories.

-access, -ac
prints your effective mode, ring brackets, access class (if different from the default), and safety switch (if on).

-access_class, -acc
prints the access class.

-all, -a, -long, -lg
rints all relevant information about the object or link: the type of entry, the pathname of the entry being linked to, names, the unique identifier, the date-used, the date-modified, the date-branch-modified, the date-link-modified, the date-dumped by hierarchy and volume dumpers, the author, the bit count author (if different from the author), the device, the bit count, records used, current blocks (for segments, if different from records used), the maximum length in words (if type is segment), the safety switch (if on), the damaged switch (if on), your mode, ring brackets, the access class (if not null), the copy switch (if on), and the volume dumper control switches (if off). Bit count author, bit count, safety switch, and copy switch are not printed for DM files. Synchronized switch is not printed for directories.

-bc_author, -bca
prints the bit count author of the entry. Not valid for DM files.

-bit_count, -bc
prints the bit count. Not valid if the files selected are not DM files.

-copy_switch, -csw
prints whether the copy switch is on or off. Not valid for DM files.

-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 (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, -logical_volume, -lv
prints the logical volume on which the entry resides.

-length, -ln
1. When used as a status active function with -length, then for segments or MSRs or DM files: returns the current length; for directories: returns the bit count.

2. When used as a status command with -length, then

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. The length is in records and is based on the bit count;

for MSFs: prints the number of records used by the whole file, the sum of the bit counts of all components, and the number of components. The length is in records and is based on the bit count;

for DM files: prints the number of records used and the number of the highest control interval. The length is in records and is based on the bit count.

for directories: prints the number of records used and the bit count.

-max_length, -ml
    prints the maximum length of a segment.

-mode, -md
    prints your 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. Not valid for DM files.

-synchronized_switch, -synch
prints whether the synchronized switch is on or off. Not valid for directories.

-unique_id, -uid
prints the entry's unique identifier. For DM files, prints the file manager (fm) unique id.

LIST OF CONTROL ARGUMENTS FOR SEGMENTS

You can use the following control arguments only 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 DM FILES

You can use the following control arguments only for DM files.

-concurrency_sw, -concsw
prints whether the concurrency switch is on or off.

-highest_control_interval, -hci
prints the number of the highest control interval allocated to the file.

-protected_sw, -psw
prints whether the protected switch is on or off.

-rollback_sw, -rlbsw
prints whether the rollback switch is on or off.

LIST OF CONTROL ARGUMENTS FOR LINKS

You can use the following control argument only for links.

-link_path, -lp
prints the target pathname.

NOTES

If you supply no control arguments, the following information is printed for segments, MSFs, DM files, and directories: names, the type, the date-used, the date-modified, the date-branch-modified, the bit count, records used, your mode, and the access class; for links: the pathname of the entry linked to, names, the date-link-modified, the date-dumped.

The -device, -length, -logical_volume, and -mode control arguments are ignored for links.

Zero-valued dates (i.e., dates that have never been set) and 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 MSFs are labeled as such.

For a description of the attributes listed, see "Entry Attributes" in the Programmer's Reference Manual.

EXAMPLES

In the example below, you request all the status information on the segment named >user_dir_dir>Demo>MTwain>working_file.

```
! st >user_dir_dir>Demo>MTwain>working_file -long

   names:     test_segment
              working_file
   type:                          segment
   unique id:                     764576046673
   date used:                     01/09/84 1459.0 est Mon
   date modified:                 01/09/84 1459.0 est Mon
   branch modified:               11/19/82 1542.6 est Fri
   date branch dumped:            01/27/84 0305.4 est Fri
   date volume dumped:            01/31/84 0305.4 est Tue
   author:                        LNTolstoy.Demo.a
   bit count author:              MTwain.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, you ask 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>OFOWWilde>newtest.pl1

   type:                        segment
   date used:                   01/26/84 2145.0 est Thu
   date modified:               01/13/84 1630.0 est Fri
   branch modified:             01/13/84 1626.7 est Fri
   date branch dumped:          01/14/84 0305.4 est Sat
   date volume dumped:          01/16/84 0305.4 est Mon
   mode:                        rew
   ring brackets:               4, 4, 4

     >user_dir_dir>Demo>OFOWWilde>newtest.list

   names:                       newtest.list
   type:                        link
   links to:                    user_dir_dir>Demo>OFOWWilde>sub_dir>
                                   newtest.list
   date link modified:          01/26/81 2139.3 est Fri
```

In the following example, you ask for status information about the directory named >user_dir_dir>Demo>JWVGoethe>test.

```
! status >user_dir_dir>Demo>JWVGoethe>test

   names:                       test
   type:                        directory
   date used:                   12/05/84    606.6 est Wed
   date modified:               12/05/84    606.6 est Wed
   branch modified:             11/29/84    957.2 est Thu
   bit count:                   0
   records used:                1
   mode:                        sma
   access class:                Sensitive,Research
```

Name: stop_cobol_run, scr

SYNTAX AS A COMMAND

```
scr {-control_arg}
```

FUNCTION

terminates the current COBOL run unit.

CONTROL ARGUMENTS

-retain_data, -retd
leaves the data segments associated with the programs of the run unit intact for debugging purposes (see "Notes").

NOTES

The results of stop_cobol_run and the execution of the STOP RUN statement from within a COBOL program are identical. Stopping the run unit consists of cleaning up all files that have been opened during the execution of the current run unit and ensuring that the next time a program that was a component of this run unit is invoked its data is in its initial state.

To maintain the value of all data referenced in the run unit in its last used state, use -retain_data.

Refer to the run_cobol command for information concerning the run unit and the COBOL runtime environment. See also the cancel_cobol_program and display_cobol_run_unit commands.

Name: stop_run

SYNTAX AS A COMMAND

```
stop_run
```

FUNCTION

is used in conjunction with the run command to effect an abnormal termination of the run-unit created by the run command.

NOTES

The stop_run command signals the finish condition, executes the epilogue handlers, and forces a return from the run command to its caller. It is normally called internally by any fortran main program, or any of "call exit", "stop", or the "end" of a main program. For a description of run units see the run command.

Name: string

SYNTAX AS A COMMAND

```
string {STRs}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[string {STRs}]
```

FUNCTION

returns a single character string formed by concatenating all of the strings together, separated by single spaces. If no strings are specified, a null character string is returned. If one or more strings are specified, any quotes in these are returned as single quotes.

EXAMPLES

The following interactions illustrate the string command.

```
! string He said, "Hi."
  He said, Hi.
! string He said, """Hi."""
  He said, "Hi."
```

The following interaction illustrates the active function.

```
! string [string This is "food".]
  This is food.
```

Name: strip

SYNTAX AS A COMMAND

```
strip path {STR}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[strip path {STR}]
```

FUNCTION

returns the absolute pathname of the specified entry, with the last component removed if the entryname portion has more than one component. If an archive component pathname is specified, the components are stripped off the archive component name if it has more than one component.

ARGUMENTS

path
: is the pathname from which the trailing name component is removed.

STR
: is the character string to be stripped. If STR is not specified, the last component is removed. If STR is specified, the last components are removed only if they match STR and STR does not equal path.

NOTES

Since the pathname is returned in quotes, the command processor treats it as a single argument regardless of special characters in the name.

EXAMPLES

Assume the working directory is >udd>Proj>Myname.

```
! strip "foo bar.compin" compin
  >udd>Proj>Myname>foo bar

! strip start_up.ec.old
  >udd>Proj>Myname>start_up.ec

! strip start_up.ec.old ec.old
  >udd>Proj>Myname>start_up

! strip start_up.ec.old xyz
  >udd>Proj>Myname>start_up.ec.old

! strip >udd>Multics>Library>Source>bound_command_demos_.s::program.pl1
  >udd>Multics>Library>Source>bound_command_demos_.s::program
```

Name: strip_component, spc

SYNTAX AS A COMMAND

```
spc path {STR}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[spc path {STR}]
```

FUNCTION

returns the archive component name portion of the absolute pathname of the specified entry with the last component removed. If an archive component pathname is not supplied, then this is equivalent to strip_entry.

ARGUMENTS

path
: is the pathname from which the trailing name component is removed.

STR
: is the character string to be stripped. If STR is not specified, the last component of the entryname portion of path is removed. If STR is specified, the last components are removed only if they match STR and STR does not equal path.

NOTES

Since the pathname is returned in quotes, the command processor treats it as a single argument regardless of special characters in the name.

EXAMPLES

```
! spc >udd>Proj>Myname>start_up.ec.old ec.old
  start_up

! spc >udd>Multics>Library>Source>bound_command_demos_.s::program.pl1
  program
```

Name: strip_entry, spe

SYNTAX AS A COMMAND

```
spe path {STR}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[spe path {STR}]
```

FUNCTION

returns the entryname portion of the specified pathname with the last component name removed, if the entryname portion has more than one component.

ARGUMENTS

path
: is the pathname from which the trailing name component is removed.

STR
: is the character string to be stripped. If STR is not specified, the last component of the entryname portion of path is removed. If STR is specified, the last components are removed only if they match STR and STR does not equal path.

NOTES

Since the pathname is returned in quotes, the command processor treats it as a single argument regardless of special characters in the name.

EXAMPLES

```
! spe start_up.ec.old
  start_up.ec

! spe [hd]>start_up.ec.old ec.old
  start_up

! spe >udd>Multics>Library>Source>bound_command_demos_.s::program.pl1
  bound_command_demos_.s
```

(The ".s" is not stripped because the actual entryname is "bound_command_demos_.s.archive".)

Name: substitute_arguments, sbag

SYNTAX AS A COMMAND

```
sbag {-control_args} {control_string {args}}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[sbag control_string {args}]
```

FUNCTION

substitutes arguments into a control string and prints the result on user_output. As an active function, the result is returned.

ARGUMENTS

control_string
: is a character string that can contain substitution constructs (see "List of Substitutions" below).

args
: are zero or more character string arguments. Any argument supplied but not referenced by an argument substitution designator is ignored.

CONTROL ARGUMENTS

If you give control arguments with no control string, subsequent sbag invocations in the process are affected; with a control string and its arguments, subsequent sbag invocations are not affected. Give the control arguments first. (See "Notes on modes" below.)

-brief, -bf
: does not print the expanded control string. (Default)

-control_string, -cs
: permits a control string to look like a control argument.

-long, -lg
: prints the expanded control string on error_output before it is printed or returned.

LIST OF SUBSTITUTIONS

The following expansion designators appearing in the control string are replaced by their expansion value, as described below. Any other use of the ampersand (&) produces an error.

&0, &1,...&9
expands to the zeroth through ninth arguments. &0 is the control string, &1 is the first argument following the control string, and so on. If the corresponding argument is missing, the designator expands to a null string.

&(0), &(1),...
expands to any argument, including arguments after the ninth. Use parenthesis when the argument number is two or more digits. If the corresponding argument is missing, the designator expands to a null string.

&q0,...&q9, &q(0), &q(1),...
expands to the corresponding argument following the control string. Quotes within the argument are doubled, according to the quote depth of the surrounding context within the control string (see "Notes on Quote Doubling" below).

&r0,...&r9, &r(0), &r(1),...
expands to the corresponding argument following the control string, enclosed in an added layer of quotes with internal quotes with the argument doubled accordingly (see "Notes on Requoting" below). This designator keeps the argument as a single unit after one layer of quote stripping by the command processor.

&f1,...&f9, &f(1),...
expands to the Nth through last arguments following the control string, with arguments separated by one space. If N is greater than &n, expands to a null string.

&qf1,...&qf9, &qf(1),...
expands to the Nth through last arguments following the control string, with quotes doubled within arguments, and arguments separated by one space. If N is greater than &n, expands to a null string.

&rf1,...&rf9, &rf(1),...
expands to the Nth through last arguments following the control string, with each argument individually requoted, and arguments separated by one space. If N is greater than &n, expands to a null string.

&n
expands to the number of arguments you give following the control string.

&f&n, &qf&n, &rf&n
expands to the last argument following the control string, with quotes doubled (&qf&n) or with requoting (&rf&n).

&control_string
expands to the control string (without expansions), with quotes doubled. It is equivalent to &q0.

&!
expands to a unique name. Each use of &! is replaced by a 15-character identifier. Every use within a single invocation is replaced by the same string, but the string is different for every invocation of exs.

&&
expands to a single ampersand, to allow ampersands to be literally inserted into the expanded control string.

NOTES

The substitute_arguments active function is similar to the do active function. The do command is an older interface that acts like exs as a command and like sbag as an active function.

NOTES ON MODES

This command has a long/brief mode. This mode is kept in internal static storage and is thus remembered from one invocation of sbag to the next in a single process. Set the mode for the life of the process by invoking sbag with control arguments and no control string; set the modes for a single invocation by giving control arguments, a control string, and its arguments.

The modes of the sbag command are separate from the modes of the do and execute_string commands, although they provide similar functions.

NOTES ON QUOTE DOUBLING

Each parameter designator to be expanded is found nested a certain level deep in quotes. If it is found to be outside quotes, its quote level is zero; if found between a single pair of quotes, its quote level is one; and so on. If an "&q" construct is found nested to quote-level L, then, as the argument is substituted into the expanded control string, each quote character found in the argument 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 control string. If the "&q" construct is not between quotes, or if the corresponding argument contains no quotes, quote doubling has no effect.

NOTES ON REQUOTING

If an "&r" construct is found, the substituted argument is placed between an additional level of quotes before having its quotes doubled. For example, if &r1 is found nested to quote level L, 2**L quotes are inserted into the expanded control string; then, the first argument is substituted, with each of its quotes replaced by 2**(L+1) quotes; and, finally, 2**L more quotes are placed following it. If you give no argument, nothing is placed in the expanded control string; so, you can distinguish between arguments that are not supplied and arguments that are supplied but are null. If you give an argument, the expansion of an "&r" construct is identical to the expansion of an "&q" construct surrounded by an extra level of quotes.

Name: substr

SYNTAX AS A COMMAND

```
substr STR J {N}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[substr STR J {N}]
```

FUNCTION

returns the portion of STR starting with the character in position J and continuing for N characters (where J and N are decimal integers; J must be greater than zero and N must be greater than or equal to zero). If you omit N, the remainder of STR is returned. If J is greater than the length of STR, the null string is returned; if N is greater than the remainder of STR, the remainder is returned.

EXAMPLES

The following interaction illustrates the substr active function:

```
string [substr programmers 4 4]
gram

string [substr trounce 3]
ounce
```

Name: suffix

SYNTAX AS A COMMAND

```
suffix path
```

SYNTAX AS AN ACTIVE FUNCTION

```
[suffix path]
```

FUNCTION

returns the last component of the entryname--or archive component name, if you supply an archive component pathname--portion of the specified segment. If that entryname has only one component, the null string is returned.

ARGUMENTS

path
is the pathname from which the trailing name component is removed.

NOTES

Since the pathname is returned in quotes, the command processor treats it as a single argument regardless of special characters in the name.

EXAMPLES

```
suffix >udd>Proj>Myname>start_up.ec
ec

suffix >udd>Multics>Library>Source>bound_command_demos_.s::program.pl1
pl1
```

Name: switch_off, swf

SYNTAX AS A COMMAND

```
swf keyword paths {-control_args}
```

FUNCTION

turns off a specified switch for one or more entries--directory, segment, multisegment file (MSF), data management (DM) file, and extended entry. 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 entries for which it is possible to set the specified switch. You can use the star convention, which includes links only if you give -chase. You can specify by "-name STR" a pathname that looks like a control argument or contains starname special characters not meant to be matched.

CONTROL ARGUMENTS

-chase
includes links and chases them when you use the star convention.

-interpret_as_extended_entry, -inaee
interprets the selected entry as an extended entry type.

-interpret_as_standard_entry, -inase
interprets the selected entry as a standard entry type.

-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 you use the star convention. (Default)

LIST OF KEYWORDS

copy_switch, csw
if ON, allows processes lacking write access to modify a copy of the segment in the process directory. (Segments)

complete_volume_dump_switch, cvds
if ON, the entry is dumped during a complete volume dump of the physical volume on which it resides.

damaged_switch, dsw
if ON, the segment is assumed to have been damaged by a device error or system crash.

incremental_volume_dump_switch, ivds
if ON, the entry is dumped during an incremental dump cycle of the volume dumper.

perprocess_static_switch, ppsw
if ON, the segment's internal static storage is not initialized when a run unit is created. (Object segment)

safety_switch, ssw
if ON, the delete command and delete_ subroutine query you before deleting the entry.

synchronized_switch
if ON, writes out to disk the segment's pages only after corresponding pages in an associated before journal are written out. Only authorized users can set this switch. (Segments)

ACCESS REQUIRED

You require modify permission on the parent directory.

department
is the computer center department name.

ds_company
is the company name, with the characters of the name double spaced.

ds_department
is the computer center department name, with the characters of the name double spaced.

installation_id
is the installation identification.

last_down_reason
is the reason for the last system service interruption, if known. The reason can be:

shutdown	normal system shutdown
crash	system crash (no number assigned)
N	number of system crash

max_rate_structure_number
returns the largest valid rate structure number. If it is zero, there are no rate structures defined at this site other than the default one in installation_parms.

max_units
is the current maximum number of load units, in the form "nnn.n".

max_users
is the current maximum number of users.

n_units
is the current number of logged-in load units including daemon and absentee, in the form "nnn.n".

n_users
is the current number of logged-in users including daemon and absentee.

next_shift
is the next shift number.

rate_structure_name {rs_number}
returns the name of the rate structure corresponding to rs_number. If you give no number, the names of all rate structures defined at the site are returned in ascending order by rate structure number, separated by blanks, in a single string.

rate_structure_number {rs_name}
returns the number corresponding to rs_name.

reason_down
is the reason for next shutdown, if specified by the operator.

LIST OF KEYWORDS

copy_switch, csw
if ON, allows processes lacking write access to modify a copy of the segment in the process directory. (Segments)

complete_volume_dump_switch, cvds
if ON, the entry is dumped during a complete volume dump of the physical volume on which it resides.

damaged_switch, dsw
if ON, the segment is assumed to have been damaged by a device error or system crash.

incremental_volume_dump_switch, ivds
if ON, the entry is dumped during an incremental dump cycle of the volume dumper.

perprocess_static_switch, ppsw
if ON, the segment's internal static storage is not initialized when a run unit is created. (Object segment)

safety_switch, ssw
if ON, the delete command and delete_ subroutine query you before deleting the entry.

synchronized_switch
if ON, writes out to disk the segment's pages only after corresponding pages in an associated before journal are written out. Only authorized users can set this switch. (Segments)

ACCESS REQUIRED

You require modify permission on the parent directory.

NOTES

The keywords can also include switches defined for particular extended entry types (see describe_entry_type).

```
&if [equal [system_type] [system_type 6180]] &then logout -brief
```

Name: tape_archive, ta

SYNTAX AS A COMMAND

```
ta key table_path {args}
```

FUNCTION

performs a variety of operations to create and maintain a set of files on magnetic tape.

ARGUMENTS

key
is one of the key functions described below.

table_path
is the pathname of a segment created and maintained by tape_archive to serve as a table of contents for the archive. If the table segment does not exist, it is created by the append operation or the direct interactive mode.

args
are additional arguments or control arguments as required by the particular key chosen (see below).

LIST OF EXTRACT OPERATIONS

x

```
Usage:  ta x table_path {components} {-control_arg}
```

extracts from the archive those components named by the path arguments, placing them in segments in the storage system. The star convention is allowed for components. The directory where you place a segment is the directory portion of the component argument. The ACL, names, and other settable segment attributes that were in effect when you archived the component are placed onto the new segment. If a segment of the same name already exists, it observes the duplicate name convention like that of the copy command. If you supply no component names, all components are extracted and placed in your working directory.

xd

```
Usage:  ta xd table_path {components} {-control_arg}
```

extracts and deletes; operates like x, but deletes the component from the archive if the extraction is successful.

LIST OF KEYS

all
prints all the information available in alphabetical order sorted by keyword name. You can't use it in the active function.

company
is the company name.

date_time_last_down
is the date and time that service was last interrupted by shutdown or crash.

date_time_last_up
is the date and time that the system was brought up.

date_time_next_down
is the date and time that service will next be shut down if specified by the operator.

date_time_next_up
is the date and time that the system will next be brought up if specified by the operator.

date_time_shift_change
is the date and time at which the current shift number will change to next_shift.

default_absentee_queue
is the default absentee queue.

department
is the computer center department name.

ds_company
is the company name, with the characters of the name double spaced.

ds_department
is the computer center department name, with the characters of the name double spaced.

installation_id
is the installation identification.

last_down_reason
is the reason for the last system service interruption if known. The reason can be:

shutdown	normal system shutdown
crash	system crash (no number assigned)
N	number of system crash

max_rate_structure_number
returns the largest valid rate structure number. If it is zero, there are no rate structures defined at this site other than the default one in installation_parms.

max_units
is the current maximum number of load units, in the form "nnn.n".

max_users
is the current maximum number of users.

n_units
is the current number of logged-in load units including daemon and absentee, in the form "nnn.n".

n_users
is the current number of logged-in users including daemon and absentee.

next_shift
is the next shift number.

rate_structure_name {rs_number}
returns the name of the rate structure corresponding to rs_number. If you give no number, the names of all rate structures defined at the site are returned in ascending order by rate structure number, separated by blanks, in a single string.

rate_structure_number {rs_name}
returns the number corresponding to rs_name.

reason_down
is the reason for next shutdown if specified by the operator.

session_type
returns the type of Multics session currently in force. This will be "init" (during answering service initialization), "special" (during special session), "normal" (during normal service), and "shut" (during shutdown).

shift
is the current shift number.

sysid
is the system identifier as written on the hardcore system tape currently running. Normally this is the Multics release number (e.g., MR10.2). This information is different from the one obtained with version_id.

trusted_path_login
returns "true" if logout -hd and new_proc -auth are disabled, "false" otherwise.

version_id
is the version identifier as written on the MULT tape that was used to bring up the current system. You might set this to "37-19.3", which is an internal version number. This information is different from the one obtained with sysid.

Name: system_type

SYNTAX AS A COMMAND

```
system_type {SystemName}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[system_type {SystemName}]
```

FUNCTION

prints the canonical system type name either for the running system or for a user-specified system type. As an active function, it returns the system type name, rather than printing it.

ARGUMENTS

SystemName
is a system type name acceptable to the system_type_ subroutine. Its canonical name is printed. If you supply no SystemName, the canonical name for the system type of the running system is printed.

NOTES

To avoid embedding knowledge of the canonical names for system types in exec_coms, always use the active function first to canonicalize any name being compared with. For instance, in this example, system_type is used to compare the canonical name for "6180" against the type of the running system:

```
&if [equal [system_type] [system_type 6180]] &then logout -brief
```

Name: tape_archive, ta

SYNTAX AS A COMMAND

```
ta key table_path {args}
```

FUNCTION

performs a variety of operations to create and maintain a set of files on magnetic tape.

ARGUMENTS

key
is one of the operations described below.

table_path
is the pathname of a segment created and maintained by tape_archive to serve as a table of contents for the archive. If the table segment does not exist, it is created by the append operation or the direct interactive mode.

args
are additional arguments or control arguments as required by the particular key chosen.

LIST OF EXTRACT OPERATIONS

x

```
Usage:  ta x table_path {components} {-control_arg}
```

extracts from the archive those components named by the path arguments, placing them in segments in the storage system. You can use the star convention for components. The directory where you place a segment is the directory portion of the component argument. The ACL, names, and other settable segment attributes that were in effect when you archived the component are placed onto the new segment. If a segment of the same name already exists, it observes the duplicate name convention as that of the copy command. If you supply no component names, all components are extracted and placed in your working directory.

xd

```
Usage:  ta xd table_path {components} {-control_arg}
```

extracts and deletes; operates like x, but deletes the component from the archive if the extraction is successful.

xdf

```
Usage:  ta xdf table_path {components} {-control_arg}
```

extracts forcibly and deletes; operates like xd, but forcibly deletes any existing segments in the storage system if all their names conflict with names of components being extracted. This request also disregards the safety switch when deleting components from the archive.

xf

```
Usage:  ta xf table_path {components} {-control_arg}
```

extracts forcibly; operates like x, but forcibly deletes existing segments in the storage system if all their names conflict with names of components being extracted.

The extract operation has this control argument:

-single_name, -snm
: places the name of the component, as it appears in the table, as the only name on the newly created file in the storage system. (Default: place all the names)

LIST OF APPEND OPERATIONS

a

```
Usage:  ta a table_path {paths} {-control_args}
```

appends named files to the archive. The star convention is allowed for paths. The files that are appended to the archive are not otherwise affected. If the named file is already in the archive, a diagnostic is issued and the component is not replaced. At least one file must be explicitly named by the path arguments. If the tape archive does not exist, it is created.

ad

```
Usage:  ta ad table_path {paths} {-control_args}
```

appends and deletes; operates like a, but then deletes each file that was appended to the archive. Deletion takes place after the tape is processed and the file has been successfully appended to the tape. If the safety switch is on for any named file, you are queried whether the file should be deleted.

adf

```
Usage:  ta adf table_path {paths} {-control_args}
```

appends and deletes forcibly; operates like ad, but the safety switch is disregarded.

The append operation has these control arguments:

-mode ascii
-mode binary
-mode ebcdic
specifies that the file is to be replaced on or appended to the tape archive using the supplied encoding mode. If you give the ascii or ebcdic encoding mode, the file is verified to ensure that it can be encoded in the specified mode without loss of information; if it can't, a warning message is printed and the encoding mode for that file is changed to binary. If you don't give explicit mode specifications, the file is encoded in the mode determined by the criteria described under "Notes on Default Encoding Modes" below.

-single_name, -snm
records the name of the component, as specified in the command line, as the only name for the file on the volume set.

LIST OF REPLACE OPERATIONS

r

```
Usage:  ta r table_path {paths} {-control_args}
```

replaces components in or adds components to the tape archive. The star convention is allowed for paths. If you name no files in the command line, all files of the archive for which files by the same name are found in your working directory are replaced. If a file is explicitly named and does not already exist in the tape archive, it is appended and an advisory is printed. If the tape archive does not exist, then it is created.

rd

```
Usage:  ta rd table_path {paths} {-control_args}
```

replaces and deletes; operates like r, but deletes each file that was replaced in or appended to the archive. Deletion takes place after the tape is processed and the file has been successfully replaced on or appended to the tape. If the safety switch is on for any named file, you are queried whether the file should be deleted.

rdf

```
Usage:  ta rdf table_path {paths} {-control_args}
```

replaces and deletes forcibly; operates like rd, but the safety switch is disregarded. The replace operation has these control arguments:

-mode ascii
-mode binary
-mode ebcdic
specifies that the file is to be replaced on or appended to the tape archive using the supplied encoding mode. If you give the ascii or ebcdic encoding mode, the file is verified to ensure that it can be encoded in the specified mode without loss of information; if it can't, a warning message is printed and the encoding mode for that file is changed to binary. If you don't give explicit mode specifications, the file is encoded in the mode determined by the criteria described under "Notes on Default Encoding Modes" below.

-single_name, -snm
records the name of the component, as specified in the command line, as the only name for the file on the volume set.

LIST OF UPDATE OPERATIONS

u
```
Usage:  ta u table_path {paths}
```

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

ud
```
Usage:  ta ud table_path {paths}
```

updates and deletes; operates like u, but deletes each file that was updated in the archive. Deletion takes place after the tape is processed and the file has been successfully updated on the tape. If the safety switch is on for any named file, you are queried whether the file should be deleted.

udf
```
Usage:  ta udf table_path {paths}
```

updates and deletes forcibly; operates like ud, but the safety switch is disregarded.

LIST OF DELETE OPERATIONS

d
```
Usage:  ta d table_path components
```

deletes named components from the archive. The star convention is allowed for components.

df
```
Usage:  ta df table_path components
```

deletes forcibly; operates like d, but the safety switch is disregarded.

LIST OF CANCEL OPERATIONS

cancel

```
Usage:  ta cancel table_path {components}
```

cancels any pending requests for the components named. The star convention is allowed for components. This operation removes any requests scheduled to be performed on the named components. If you name no components, you are queried whether all pending requests are to be canceled. Use cancel to reinstate dead components (components that have been logically deleted or replaced from a tape archive but still exist on the volume set).

LIST OF TABLE OF CONTENTS OPERATIONS

t

```
Usage:  ta t table_path {components} {-control_args}
```

prints table of contents and associated information for each named component of the archive (including files scheduled to be placed into the archive), as well as information about the archive itself. The star convention is allowed for components.

The table-of-contents operation has these control arguments:

-all, -a
prints dead components (see the cancel operation).

-brief, -bf
prints the component name only.

-header, -he
prints the header information.

-long, -lg
prints all the information shown below, in the absence of -header.

-no_header, -nhe
suppresses the header information, even if you select -long.

-pending
prints only those components for which requests are pending.

If you give no control arguments, a short header, pending operations for the named components, and the component names are printed.

The information printed in the table of contents for each component includes

1. Any type of request pending for the component
2. The entryname of the component

3. The filename of the file on the tape

4. A one-letter indication of the recording mode of the file (b for binary, a for ascii, and e for ebcdic)

5. The length of the file in storage system records

6. The bit count author of the file

7. The date the file was archived to tape

8. The date the file was last modified while still in the storage system previous to its archival

9. The date the file was dumped by the Multics backup facility

10. The pathname of the directory in which the file exists or is to be created if a request is pending.

LIST OF PROCESSING OPERATIONS

go

```
Usage:  ta go table_path {-control_args}
```

performs the actual tape mounting and processing of the queued file transfer requests. First, the current volume set is mounted. If the current volume set is empty, ta asks you which volume is to be used. This volume then becomes the current volume set and is remembered in the table. Those components scheduled for extraction are processed. Next, additions and replacements are performed. When all tape processing has been completed, requests to delete files in the storage system that have been appended or replaced are processed. Finally, if the processing involves writing to tape, the table is modified, to reflect the new state of the tape archive, and appended to the tape.

The go operation has these control arguments:

-long, -lg
prints a message for all file searches, extractions, or appendings as they are perfomed on the volume set.

-retain all
specifies that the volume set is to remain mounted after processing is complete. In cases where several successive tape-processing operations are planned, -retain speeds up the processing of requests by reducing the physical handling of the tapes. The volume set remains mounted until you invoke go with -retain none.

-retain none
reverts the effect of -retain all. (Default)

LIST OF COMPACTION OPERATIONS

compact
```
Usage:  ta compact table_path
```

schedules the tape archive for compaction. The compaction process copies the active components on the current volume set onto the alternate volume set. This process removes cumulative tape waste attributable to inactive tape files (components that have been logically deleted, updated, or replaced, but never physically removed). Having the same volume for both primary and alternate volume sets is not allowed. You can process other file transfer requests at the same time that the archive is being compacted. After the compaction operation, the alternate volume set becomes the current volume set and vice versa.

LIST OF PARAMETER ALTERATION OPERATIONS

alter
```
Usage:  ta alter table_path alter_spec
```

changes global attributes of the tape archive that can be set by you. The specific attribute modified depends on the alter_spec arguments, which can be:

auto_limit floatnum
: automatically schedules the tape archive for compaction at the next mounting whenever the number of wasted tape records on the volume set exceed a certain fraction of the total tape records used. When compaction is automatically scheduled in this manner, an advisory message is printed. The floatnum argument must be between 0.0 and 1.0. (Initial default: 1.0; never automatically compact)

compaction off
: unschedules any pending compaction of the tape archive.

density N {-alternate}
: selects the recording density (BPI) to be used on the volume set. To select a density other than the default, enter alter for the primary volume set prior to any tape operations. To change the density of an existing volume set, give -alternate. This schedules a compaction of the primary volume set onto the alternate volume set at the selected density. (Initial defaul: 1600)

volume -number N new_volume_spec {-alternate}
: makes the volume with label new_volume_spec supersede the Nth volume in the primary volume set (the alternate volume set if you give -alternate.) If new_volume_spec is the null string, the volume is deleted. If N is greater than the number of volumes currently contained in the volume set, the volume is appended to the volume set. You can't have the same volume for both primary and alternate volume sets.

volume old_volume_spec new_volume_spec {-alternate}
makes the volume (reel) with label new_volume_spec supersede the volume old_volume_spec. If old_volume_spec is the null string and you supply -alternate, new_volume_spec is appended to the alternate volume set; otherwise it is appended to the primary volume set. If new_volume_spec is the null string, old_volume_spec is deleted from the appropriate volume set. You can't have the same volume for both primary and alternate volume sets.

volume_type STR
selects the tape standard to be used, where STR is ansi or ibm. You can't change this parameter unless the volume set is empty. (Initial default: ansi)

warning_limit floatnum
prints an advisory message whenever the number of wasted tape records on the volume set reaches or exceeds a certain fraction of the total tape records. The floatnum argument must be from 0.0 to 1.0. (Initial default: 0.5)

LIST OF LOAD TABLE OPERATIONS

load_table
Usage: `ta load_table {table_path} {-control_args} {volume_ids}`

loads the copy of the online table kept on a volume set into the segment specified by table_path. If the segment already exists, you are asked whether it should be overwritten. If you don't give the tape volume name in load_table, ta queries you for a volume name. There is no way to specify the density or other characteristics of the volume, or multiple volume names, when responding to the query; use, therefore, the full load_table syntax unless the tape was recorded at 1600 BPI on a 9-track tape drive using the ANSI standard and ASCII recording mode.

Control arguments for this operation are:

-density N, -den N
specifies the density of the tape volume to be N. (Default: 1600)

-retain all
specifies that the volume set is to remain mounted after processing is complete. In cases where several successive tape processing operations are planned, -retain speeds up processing of requests by reducing the handling of the tapes. The volume set remains mounted until you invoke the processing operation (go) with -retain none.

-volume_type STR, -vt STR
specifies the per-format module originally used by mtape_ to generate the tapes. Acceptable volume tapes are ansi and ibm. (Default: ansi)

LIST OF RECONSTRUCT TABLE OPERATIONS

reconstruct

```
Usage:  ta reconstruct table_path {volume_ids} {-control_args}
```

scans the volume set and constructs a valid table into the segment specified by table_path. If the segment already exists, you are asked whether it should be overwritten. If you don't give the tape volume name in the command line, ta queries you for a volume name. There is no way to specify the density or other characteristics of the volume, or multiple volume names, when responding to the query; use, therefore, the full reconstruct syntax unless the tape was recorded at 1600 BPI on a 9-track tape drive using the ANSI standard and ASCII recording mode. Examine the table that is reconstructed for accuracy, as deleted or replaced files on the volume set may be also reconstructed.

Control arguments for this operation are:

-density N, -den N
: specifies the density of the tape volume to be N. (Default: 1600)

-force, -fc
: forces the overwriting of an already-existing tape_archive table. (Default: to query for overwriting)

-long, -lg
: displays on the terminal the names of the files it has added to the table and the tables that it has found on the volume set and processed. (Default: not to display anything except error messages)

-retain all
: specifies that the volume set is to remain mounted after processing is complete. In cases where several successive tape processing operations are planned, -retain speeds up processing of requests by reducing the handling of the tapes. The volume set remains mounted until you invoke the processing operation (go) with -retain none.

-volume_type STR, -vt STR
: specifies the per-format module originally used by mtape_ to generate the tapes. Acceptable volume tapes are ansi and ibm. (Default: ansi)

LIST OF INTERACTIVE MODES

direct

```
Usage:  ta direct table_path {-control_arg}
```

allows you to direct the actions of ta using an interactive mode in which each line typed is interpreted as a key followed by the arguments (except for table_path) accepted by that key. This mode of operation is exited by typing "go". If any requests are outstanding when the mode is exited, the tapes are automatically mounted and the requests performed except as noted below.

The direct operation has this control argument:

-retain all
specifies that the volume set is not to be dismounted when the "go" request is complete. If you give -retain, the "go" request does not terminate the command invocation, but returns you to the interactive mode of ta so that you can enter more requests. Use the "quit" request to exit this mode and dismount the volume sets.

In addition, the following special commands are accepted in this mode of operation:

causes ta to identify itself.

..{command_line}
passes the specified command line to the command processor.

go
specifies that all preceding requests are to be recorded into the table and that the volume set is to be mounted and processed. If you haven't set the volume name with the alter request, the go request queries you for a volume name. All other information about the tape volume set must have been previously set by alter requests; otherwise the defaults apply. Use a 1600 BPI minimum (or 6250 if available) for tape archives unless they are specifically intended for interchange with non-Multics systems. To set the first volume name, use a request of the following form:

```
tape_archive alter foo volume "" VOLUME_NAME
tape_archive alter foo density 1600
```

You can't alter the tape archive until at least one component has been added. Alter the volume and density after adding a component, but before using the go request for the first time.

quit
exits the interactive mode without performing the actual processing of the requests. Unless preceded by save, all requests made in this invocation of ta are discarded. If unsaved requests exist, you are asked to confirm the command.

save
permanently records in the table all requests made during this invocation of the command.

While in the interactive mode, all requests are maintained in a temporary copy of the online table, allowing you to abort processing if desired without recording any requests in the actual online table.

All keys are accepted in this mode of operation except for load_table.

NOTES

This command provides you with the ability to append components to the archive, replace its components with new versions, extract and delete its components, list its contents, and re-create the online table in the event of a catastrophe or for file transferring.

You can use a tape archive to temporarily hold files that will be needed at some future time, but that meanwhile take up large amounts of expensive storage space. Additionally you can use tape archives to transfer files between Multics systems and, in a limited fashion, from Multics to other operating systems.

A tape archive consists of one or more reels of tape, known as the "volume set," on which files are stored in ANSI or IBM standard tape format, one archive per volume. The constituent files that compose the tape archive are called components of the archive. Associated with each tape archive is a Multics segment known as the table. This segment is created and maintained by ta and contains information about each component in the archive.

You can request to move components between the tape and the storage system by invoking ta before any reels are actually mounted and processed. Once you have specified all desired transfer requests, invoke ta to mount and process the tape.

An interactive mode of operation is supplied that allows you to specify multiple requests to a single invocation of the command and that automatically performs the requests after you have satisfactorily entered them all.

NOTES ON DEFAULT ENCODING MODES

If you give no particular encoding mode for files being appended to, or replaced in, the archive, the following criteria are applied to determine the most appropriate mode: When performing file replacement, the default encoding mode remains unchanged if it is determined that the file has not been altered in any way that precludes encoding it in the same mode; otherwise a diagnostic is printed, and the replacement is performed in binary mode.

NOTES ON TAPE FILE NAMING CONVENTIONS

Tape files of a ta volume set follow certain conventions regarding ordering and naming.

Each user file is preceded by an attribute file, containing the information necessary to re-create the file faithfully in the storage system (e.g., names, ACL). Attribute files are named "ATTRIBUTEFILENNNN" for ANSI tapes, and "ATTRIBUT.FILENNNN" for IBM tapes, where NNNN is the physical file number (by order of occurrence on the tape) of the attribute file, e.g., "ATTRIBUTEFILE0023".

Each user file is named with a unique name constructed of all or part of the Multics entryname of the file, translated to uppercase, one or more reserved characters, and the physical file number of the file. For ANSI tapes, the reserved character is a slash (/); for IBM tapes, the commercial-at sign (@). The Multics file name is truncated or padded with reserved characters to 12 characters. In addition, characters appearing in the Multics file name that are not allowed as part of a tape file name under the applicable standard are translated to the reserved character. Due to IBM file-naming restrictions, the ninth character of all tape file names on IBM tapes is translated to a period, and if the character following the period is not alphabetic, that character is translated to an X.

For example, on an ANSI tape the name of the tape file containing the Multics file named "alpha" might be "ALPHA////////0024", and the name of the tape file containing the Multics file named "source.archive" might be "SOURCE.ARCHI/0037". On an IBM tape these files might appear as "ALPHA@@@.X@@@0024" and "SOURCE@A.CHI@0037", respectively.

Copies of the online table describing the tape are named "ONLINE-TABLE-NNNN" for ANSI tapes and "ONLINE#T.BLE#NNNN" for IBM tapes, where NNNN is a number representing the serial number of the online tables on this volume set. This number starts from 1 and increases by one each time a new table is written to the tape.

This page intentionally left blank.

Name: tape_in

SYNTAX AS A COMMAND

```
tape_in path {-control_args}
```

FUNCTION

allows the user to transfer files between magnetic tape and the storage system. To accomplish a file transfer, the tape_in command accesses either the tape_ansi_ or the tape_ibm_ I/O module for the tape interface, and the vfile_ I/O module for the storage system interface. Unstructured format storage system files (for stream I/O) and sequential format storage system files (for record I/O) may be specified; 9-track ANSI standard labeled tapes, 9-track IBM standard labeled tapes, and any 9-track unlabeled tape structured according to OS standards may be read.

ARGUMENTS

path
: is the pathname of the control file governing the file transfer. If path does not end with the tcl suffix, it is assumed.

CONTROL ARGUMENTS

-severityN, -svN
: causes the tape_in compiler's error messages with severity less than N (where N is 0, 1, 2, 3, or 4) not to be written into the error_output I/O switch. The default value for N is 0. See "Error Diagnostics" below for further information on error reporting.

-check, -ck
: performs only semantic checking on the Tape Control Language (TCL) control file. No tapes are mounted if this control argument is specified.

-ring
: mounts volumes of the volume-set with write permit rings.

BASIC TCL CONTROL FILE

The control file that governs file transfer is actually a program, written by the user, in the Tape Control Language (TCL). The contents of this control file describe the file transfer(s) to take place. When the user issues the tape_in or tape_out command, the control file named in the command line by the path argument is compiled and if the compilation is successful, the generated code is interpreted to accomplish the desired file transfer(s). The same control file may be used with both the tape_in command (to read a file from tape into the storage system) and with the tape_out command (to write a file from the storage system onto tape).

The TCL control file consists of a list of statements of the form:

```
     <keyword>:      <argument(s)>;
or:
     <keyword>;
```

These statements are combined to form file-groups and file-groups are combined to form volume-groups. A TCL control file consists of one or more volume-groups.

A file-group is a list of statements that define one tape to storage system file transfer. A file-group must begin with a File statement and must contain a path statement. In addition, it may contain one or more local statements. A file-group is terminated by a global statement, an End statement, or another File statement.

A volume-group is a series of statements that specify the file transfer(s) to be performed between the storage system and a particular tape volume-set. A volume-group must begin with a Volume statement, contain one or more file-groups, and terminate with an End statement. In addition, a volume-group may optionally contain one or more global statements, which apply to all the file-groups within the volume-group that follow the global statement.

All TCL control files must have at least four statements: a Volume statement, a File statement, a path statement, and an End statement; all other TCL statements are optional. The simplest control file has just these four statements, for example:

```
Volume:   012345;
File:   File1;
path:   >udd>Project_id>Person_id>demo;
End;
```

This example control file relies on TCL control file defaults, which are listed below under "Volume-Group Defaults." The file transfers possible with this sample control file are two: either writing tape file File1 from storage system file demo; or writing storage system file demo from tape file File1.

LIST OF TCL CONTROL FILE STATEMENTS

Volume statement

```
Volume:  <volid>;
```

specifies the tape volume to be used in file transfer. This statement causes a tape volume whose volume identifier is <volid> to be mounted on a 9-track drive. If <volid> contains any of the following characters, it must be enclosed in quotes.

1. any ASCII control character.
2. : ; , or blank
3. the sequence /* or */
4. If <volid> itself contains a quote character, the quote must be doubled and the entire <volid> string enclosed in quotes.

Some examples of Volume statements are:

```
Volume: 23;              (mounts volume 23)
Volume: 001234;          (mounts volume 001234)
Volume: XJ56;            (mounts volume XJ56)
Volume: "as"";56";       (mounts volume as";56)
Volume:  -00451;         (mounts volume -00451)
```

See the descriptions of the tape_ansi_ and tape_ibm_ I/O modules in the Subroutines manual for more details on volume specifications. Also, see "Multivolume Files" below for a discussion of multivolume volume-groups.

File statement

```
File:  <fileid>;
```

specifies the tape file to be read or written. For output, <fileid> must be from one to 17 characters for ANSI labeled tapes and must be a valid DSNAME for IBM labeled tapes. A valid DSNAME is from one to eight characters long. The first character must be an alphabetic or national (@,$,#) character; the remaining characters can be any alphanumeric or national characters, a hyphen (-), or a plus zero (12-0 punch). For input, <fileid> may be an asterisk () for labeled tapes, if a tape file sequence number is also specified. For output with labeled tapes, <fileid> may not be an asterisk. <fileid> for IBM unlabeled tapes, which are discussed below, must be an asterisk. The File statement marks the beginning of any local attributes for a given tape file transfer.

path statement

```
path:  <pathname>;
```

associated with every File statement must be one path statement. The path statement specifies the pathname of the storage system file to be read or written. <pathname> may be either a relative or absolute pathname.

End statement

```
End;
```

associated with every Volume statement must be an End statement, to mark the end of the TCL for that volume-group.

LIST OF GLOBAL STATEMENTS

A global statement changes a volume-group default. The Tape and the Density global statements may appear only once in a volume-group and must preceed all file-groups. The Block, Expiration, Format, Mode, Record, and Storage global statements may appear any number of times within a volume-group. These statements apply to all subsequent file-groups within the volume-group.

Block statement

```
Block: <blklen>;
```

specifies the tape file (maximum) physical block length, in bytes, to be used with subsequent file-groups. The <blklen> specification must be a decimal integer >= 18. For IBMSL, IBMNL, and IBMDOS formats, the maximum value is 32760 bytes. For ANSI formats, the maximum value is 99996 bytes. WARNING: <blklen> greater that 2048 does not comply with the ANSI standard for tapes.

Density statement

```
Density: <den>;
```

indicates the density in which the volume is (to be) recorded. <den> must be either 800, 1600, 6250, 2, 3, or 4 (for IBM compatibility) to indicate 800, 1600 or 6250 bpi respectively. WARNING: the use of 1600 or 6250 bpi for ANSI interchange tapes is nonstandard. This global statement may appear only once within a volume-group or an error is indicated.

Expiration statement

```
Expiration: <date>;
```

specifies the expiration date of files to be written (created). <date> is a string of a form acceptable to the convert_date_to_binary_ subroutine, for example "11/08/82". Because overwriting a file on a tape logically truncates the file set at the point of overwriting, the expiration date of a file must be earlier than or equal to the expiration date of the previous file (if any) on the tape; otherwise, an error is indicated. If an attempt is made to overwrite an unexpired file, the user is queried for explicit permission at the time of writing, unless the -force control argument is specified in the command line (only possible with tape_out).

Format statement

```
Format:  <form>;
```

specifies the tape record format to be used with subsequent file-groups. <form> must be either u, f, fb, d, db, s, or sb for ANSI tapes (using tape_ansi_ I/O module) and f, fb, u, v, vs, vb, or vbs for IBM tapes (using tape_ibm_ I/O module).

Mode statement

```
Mode:  <mode>;
```

specifies the tape mode and character code to be used with subsequent file-groups. <mode> may be either ascii or ebcdic for IBM tapes (using tape_ibm_ I/O module) and may be either ascii, ebcdic, or binary for ANSI tapes (using tape_ansi_ I/O module). WARNING: the use of ebcdic mode or binary mode is not standard for ANSI tapes. See "I/O Module Compatibility and Record Length Tables" below for a description of the interaction between a given combination of format, block, and record specification. Values must be carefully chosen to ensure desired results.

Record statement

```
Record:  <reclen>;
```

specifies the tape file (maximum) logical record length, in bytes, to be used with subsequent file-groups. <reclen> must be a decimal integer, such that 1 <= <reclen> <= maximum segment size in bytes.

Storage statement

```
Storage:  <structure>;
```

states the internal (logical) structure of the storage system file(s) to be specified by subsequent file-groups. An unstructured file is referenced as a series of 9-bit bytes, commonly called lines; a sequential file is referenced as a sequence of records, each record being a string of 9-bit bytes. <structure> must be either unstructured or sequential. When an unstructured file is written into the storage system from a tape the NL character is appended as each line is written, unless the record already ends in a NL character, in which case nothing further is appended. When an unstructured file is written from the storage system to tape, the NL character is stripped off before writing the tape record. If a line of an unstructured file consists of just a NL character, it is written to tape as a zero length record. If the Storage global statement is omitted from a control file volume-group, the assumed storage system file format is unstructured. If a sequential file is referenced within that volume-group, the results are undefined and an error is indicated. Processing is terminated on that file in which the error is indicated.

Tape statement

```
Tape: <tape-type>;
```

specifies the kind of tape that is processed. <tape-type> may be ibmsl for IBM standard labeled tape, ibmnl for IBM unlabeled tape, ibmdos for IBM DOS standard labeled tape, or ansi for ANSI standard labeled tape. The tape label processing is done automatically by the I/O module in use. This global statement may appear only once within a volume-group or an error is indicated.

LIST OF LOCAL STATEMENTS

A file-group may contain one or more local statements. A local statement overrides the volume-group defaults in effect at the time a file-group is evaluated. A local statement has no effect outside of the file-group in which it occurs and may appear anywhere within the file-group.

The block, expiration, format, mode, record and storage local statements operate exactly as do their global statement counterparts, except that they affect only the file-group in which they are contained.

generate statement

```
generate;
```

causes the entire contents of a file on an ANSI tape to be replaced while retaining the structure of the file itself and incrementing the file generation number. The file to be modified is identified by the File statement, or by a combination of the File statement and the number statement.

modify statement

```
modify;
```

causes the entire contents of a file on an ANSI or IBM labeled tape to be replaced while retaining the structure of the file itself. The file to be modified is identified by the File statement, or by a combination of the File statement and the number statement.

number statement

```
number: <number>;
```

specifies the file sequence number of the file to be used in the file transfer. <number> must be either an integer between 1 and 9999 inclusive, or the character "*". For input with labeled tapes, <number> = is ignored unless was specified for the <fileid> in the File statement. (In this case an error is indicated.) For output

with labeled tapes, <number> = appends the current file to the volume-set. If a tape volume has not yet been initialized, that is, if the first file to be written is the first file on that tape volume, <number> = is considered a fatal error. Until a volume has been initialized, files cannot be appended to it. In this situation, either the number statement should be omitted or, if used, <number> must be equal to 1.

If the control file is to be used with the tape_in command, <number> specified in a number statement must correspond with a file on the specified tape volume-set. If both the <fileid> in the File statement and the <number> in the number statement are specified in the file-group, they must identify the same tape file; otherwise an error is indicated.

When reading unlabeled tapes, the number statement is required to identify the file to be read. When writing unlabeled tapes, the number statement is required to locate the tape position at which to write the file.

When the control file is to be used with the tape_out command for writing labeled tapes, the number statement is optional. If the number statement is given in a control file for use with the tape_out command, the file location specified in the number statement is the location where the file is written on the tape. Otherwise, with no number statement, the first file to be written in a volume-group is the first file position on the tape (for labeled tapes only). Subsequent files on that volume are appended after the first file.

replace statement

```
replace: <fileid>;
```

if an existing tape file is to be replaced on an ANSI or IBM standard labeled tape and its name is known, the file to be overwritten is identified by <fileid> in the replace local statement and the new file to be written is identified by <fileid> in the File statement. If the file identified in the replace statement does not exist, an error is indicated.

storage_extend statement

```
storage_extend;
```

Normally when a user sets up a file-group to transfer a tape file to a storage system file, it is intended that a new file be created in the storage system. Should the user want to extend an already existing file in the storage system, the storage_extend local statement should be used in the TCL control file. If the storage system file to be extended does not exist, an error is indicated. If the storage_extend local statement exists in a control file used with tape_out, it is ignored.

tape_extend statement

```
tape_extend;
```

allows new data records to be appended to an existing file on an ANSI or IBM standard labeled tape without in any way altering the previous contents of the tape file. The tape file to be extended is identified by the File statement or by the File statement and number local statement in combination. If the tape file to be extended does not exist on the tape, an error is indicated. Recorded in the labels of an ANSI or IBM labeled tape file is the version number. Initially, it is zero when the file is created. Every time a file is extended, its version number is incremented. The version number field is two digits and is reset to zero when the one-hundredth revision is made.

CONTROL FILE COMMENTS

Comments may be inserted anywhere within the TCL program by surrounding the comment text with the comment delimiters. `/*` is the delimiter that begins a comment, and `*/` is the delimiter that terminates the comment.

VOLUME-GROUP DEFAULTS

Associated with a volume-group are a set of default characteristics. In the absence of overriding global statements or local statements, these defaults apply to all file-groups within the volume-group. If no tape-type is specified in the control file, ANSI standard labeled tape is assumed. If, however, a tape-type is specified (using a Tape statement), the volume-group defaults for that tape-type are in effect until overridden.

Tape-type ANSI or no Tape statement (this is the default):

1. density: 800 bpi
2. file expiration: immediate
3. storage system file format: unstructured
4. mode: ascii
5. tape file record format: variable length records, blocked
6. physical block length: 2048 characters (maximum)
7. logical record length: 2048 characters (maximum)

Tape-type ibmsl, ibmnl, or ibmdos:

1. density: 1600 bpi
2. file expiration: immediate
3. storage system file format: unstructured
4. mode: ebcdic
5. tape file record format: variable length records, blocked
6. physical block length: 8192 characters (maximum)
7. logical record length: 8188 characters (maximum)

I/O MODULE COMPATIBILITY AND RECORD LENGTH TABLES

tape_ansi_

```
mode: ascii (default) | binary | ebcdic
block length: 18 <= B <= 99996 bytes (2048 default)
   for output mode, block length must be divisible by 4
density:  D = 800 (default) | 1600 | 6250
file sequence number: 1 <= N <= 9999 or *
record length: 0 < R < 1044480
format: F = fb | f | db (default) | d | s | sb | u
```

tape_ibm_

```
mode: ascii | ebcdic (default)
block length: 20 <= B <= 32760 bytes (8192 default)
   for output mode, block length must be devisible by 4
density:  D = 800 | 1600 (default) | 6250
file sequence number: 1 <= N <= 9999 or *
record length: 0 <= R <= 1044480
format: F = fb | f | vb (default) | v | vbs | u
```

Format	Record Length in bytes (R)	Block Length in bytes (B)
u	R is undefined	amrl <= B <= 99996 (tape_ansi_) amrl <= B <= 32760 (tape_ibm_)
f	R = amrl	B = R
fb	R = amrl	B must satisfy mod(B,R) = 0
d	amrl+4 <= R <= 99996	B = R
db	amrl+4 <= R <= 99996	B >= R
s	amrl <= R <= 1044480	18 <= B <= 99996
sb	amrl <= R <= 1044480	18 <= B <= 99996
v	amrl+4 <= R <= 32756	B = R + 4
vb	amrl+4 <= R <= 32756	B >= R + 4
vs	amrl <= R <= 1044480	20 <= B <= 32760
vbs	amrl <= R <= 1044480	20 <= B < 32760

NOTES

amrl is the actual or maximum record length of a given record format, i.e., the actual or maximum number of characters that can be recorded in a logical record. The value of R is dependent on the choice of record format. For ANSI tapes, B must be an integer in the range of 18 <= B <= 99996. For IBM tapes, B must be an integer in the range of 20 <= B <= 32760. For ANSI tapes, in order to comply with the ANSI standard, B must be in the range of 18 <= B <= 2048. For IBM tapes, the condition mod(B,4) = 0 must be satisfied. The TCL record statement should not be used for U-format file transfer.

ADDITIONAL OPTIONS AVAILABLE FOR THE TCL USER

A number of options are available to the user who wants to do more than the simple file transfer between a tape volume-set and the storage system. These features need not be of concern to most users, but for the user with specialized needs, these additional options are explained below.

Multivolume Files

Multivolume files are specified in a control file by a slightly more complicated Volume statement than shown above. The multiple <volid>s of such a volume-set are separated from one another by commas and are listed either in the order in which they became members of the volume-set, for input, or in the order in which they are candidates for volume-set membership, for output. The entire volume-set membership need not be specified in a Volume statement referencing a volume-set, but the first (possibly only) member must be mentioned. Up to 64 <volid>s may be specified in a single control file Volume statement.

Volume switching for multivolume files is handled automatically by the I/O modules. If sufficient volume-set members are given in the TCL control file, the volume switching is transparent to the user. If insufficient members of a volume-set are given or the membership is being developed, the user is queried during execution for names of additional volume-set members.

Sending Messages to the Operator

If it is necessary for the user to have a message displayed on the operator's console, the comment phrase can be included in the Volume statement. The comment text consists of the keyword -comment followed by the text of the message. Whenever the volume with the <volid> immediately preceding the comment phrase is to be mounted, the specified message is displayed on the operator's console. The message may be from 1 to 64 characters and must be a contiguous string with no embedded spaces or a quoted string with embedded quotes doubled. For example:

Volume: 060082 -comment "tape is Smith's" 060083 -comment tape_also_Smith's;

370/DOS Tapes

The tape_ibm_ I/O Module processes tapes created by or destined for IBM/DOS installations as well as tapes for IBM/OS installations. The Tape: ibmdos; global statement is used in the TCL control file to specify that the tape files referenced by the given volume-group are destined for or have been produced by a IBM/DOS installation. The important difference between tape files created by OS and those created by DOS operating system is that the tape file structure attributes are not recorded in the tape labels under DOS. It is therefore necessary for all of the structure attributes of a DOS tape file, namely encoding mode, logical record format, logical record length, and block size to be specified in the TCL control file.

Unlabeled Tapes

The tape_ibm_ I/O Module supports processing of unlabeled tapes, provided that the tapes are structured according to the OS standard. DOS leading tape mark (LTM) unlabeled format tapes cannot be processed. The ibmnl specification in the Tape statement is mutually exclusive with any statement, global or local, which refers to labeled tapes: namely, the Expiration global statement and the expiration, generate, modify, replace, and tape_extend local statements. If any of these appear together within the same file-group, an error is indicated. When referencing unlabeled tape files in a given file-group, the argument of the File statement, <fileid>, must be specified by an asterisk, and the tape file desired must be specified by the number local statement.

ERROR DIAGNOSTICS

The error messages issued during tape_in and tape_out compilation are graded and have the form:

```
prefix errornumber, SEVERITY # IN STATEMENT M OF LINE N
<text of error message>
SOURCE:
<source statement in error>
```

where N is the line number on which the described statement begins and M is a number identifying which statement in line N is in error. If line N contains only one statement, "IN STATEMENT M OF" is omitted from the error message.

The severity numbers produce one of the following prefixes:

SEVERITY	PREFIX	EXPLANATION
0	COMMENT	The error message is a comment.
1	WARNING	The error message warns that a possible error has been detected. However, the translation still proceeds.
2	ERROR	The error message warns that a probable error has been detected. However, the error is nonfatal, and the translation still proceeds.
3	FATAL ERROR	The error message warns that a fatal error has been detected. Processing of the input still continues to diagnose further errors, but no translation is performed.
4	TRANSLATOR ERROR	The error message warns that an error has been detected in the operation of the translator. No translation is performed.

CONTROL FILE EXECUTION

When the TCL control file is being executed in response to the tape_in command, the volume named in each volume-group of the control file is mounted in turn without a write ring (unless the -ring control argument has been specified). If any output options appear in a control file being executed in response to the tape_in command, these statements are ignored. Then each file-group in that volume-group is processed resulting in one file transfer to the storage system per file-group.

FILE TRANSFER

File transfer is performed as follows. One logical record is read from the tape file, and as many characters as were read are written into the storage system file either as a line with newline (NL) character appended, if necessary, (unstructured case) or as one logical record in a sequential format file.

EXECUTION TIME DIAGNOSTICS

Any fatal error from an I/O module during execution of a control file causes the user to be queried as to whether or not he wishes to continue processing the other file-groups and volume-groups in the control file or whether to terminate processing of the control file. In the case of some correctable errors the user will be given the alternative of controlling the process. This alternative places the user at command level allowing resolution of the problem. When the user wishes to continue processing, the start command is used. Executing the release command will cause the tape_in command to be terminated.

CONTROL FILE EXAMPLES

Below are examples of typical control files. In the first example, the user wishes to load into the storage system, the contents of volume "2314dp" which contains a dump of a disk pack containing source and data.

The numbers at the left-hand side of the page in the examples below do not actually appear in the control file, but are included only for annotation reference.

Example 1:

```
! tape_in sample1.tcl -ring

1       Volume: 2314dp;
2       /* Source Pack being loaded */
3       Tape: ibmsl;
4       Storage: unstructured;
5       Density: 800;
6       Format: fb;
7       Record: 80;
8       Block: 800;
9       File: FILEX;
10      path: <setup>data_entry>FILEX;
11      File: FILEXX;
12      path: <setup>data_entry>FILEXX;
13      File: FILEY;
14      path: <setup>data_entry>FILEY;
15      File: FILEYY;
16      path: <setup>data_entry>FILEYY;
17      File: FILEZ;
18      path: <setup>data_entry>FILEZ;
 .
 .
 .
58      File:  FILEZZ;
59      path: <setup>data_entry>FILEZZ;
60      End;
```

Annotations for sample1.tcl:

1. mounts the volume 2314dp with a write ring.
2. comment.
3. specifies an IBM standard labeled tape.
4. files are created in unstructured format, ready for use in stream I/O. NL characters are appended as the file is written to disk. The mode is the default for the ibmsl tape-type, namely, ebcdic.
5. tape is recorded at 800 bpi.
6. all files on tape are in fixed block format unless stated otherwise. Possible record padding problems may be encountered.
7. all logical records are 80 characters unless stated otherwise (card image files).
8. all files blocked to 800 characters unless stated otherwise.

9. first file to be read from tape is named FILEX. It may be at any file location on the tape. The tape is automatically positioned to the file by name.

10. read tape file, FILEX, into storage system file named FILEX. The relative pathname, <setup>data_entry>FILEX, is expanded.

11. continue reading files off the tape volume, one by one, into files in the storage system with the same name.

.
.
.

60. end of volume-group and end of control file.

Example 2: Control File for Reading DOS tape

```
   !  tape_in sample2.tcl

1     Volume:  042281 -comment "Please send tape to accounting";
2     Tape: ibmdos;
3     Density: 800;
4     Storage: unstructured;
5     Mode: ebcdic;
6     File: abc;
7     record: 80;
8     block:  800;
9     format: fb;
10    path: >udd>Example>Foo>fargo.pl1;
11    End;
```

Annotations for sample2.tcl:

Note: Only selected statements in the control file are annotated here.

1. mount volume 042281 without a ring after printing comment message for operator.

2. read IBM DOS standard labeled tape.

4. read tape file into storage system as unstructured format files appending NL characters to each record from tape.

Example 3: Control File for Reading an Unlabeled Tape

```
      !  tape_in sample3.tcl

1     Volume: 042381;
2     Tape: ibmnl;
3     Storage: sequential
4     File: *;
5     format:  vbs
6     number: 3;
7     path: >udd>Example>Foo>foobar.data;
8     End;
```

Annotations for sample3.tcl:

Note: Only selected statements in the control file are annotated here.

2. unlabeled tape is to be read. Files are unnamed. This statement must appear when processing unlabeled tapes.

4. <fileid> is specified by "*" for unnamed files.

6. the number statement must be present when processing unlabeled tapes. The third file on the tape is read.

The tape file record format is VBS, the tape file record length for VBS format is 1044480 bytes, and the tape file block length is 8192 bytes.

Name: tape_out

SYNTAX AS A COMMAND

```
tape_out path {-control_args}
```

FUNCTION

allows you to transfer files between the storage system and magnetic tape. To accomplish a file transfer, the tape_out command accesses either the tape_ansi_ or the tape_ibm_ I/O module for the tape interface, and the vfile_ I/O module for the storage system interface. Unstructured format storage system files (for stream I/O) and sequential format storage system files (for record I/O) can be specified; 9-track ANSI standard labeled tapes, 9-track IBM standard labeled tapes, and any 9-track unlabeled tape structured according to OS standard can be written.

ARGUMENTS

path
is the pathname of the control file governing the file transfer. If pathname does not end with the tcl suffix, it is appended.

CONTROL ARGUMENTS

-severityN, -svN
causes the tape_out compiler's error messages with severity less than N (where N is 0, 1, 2, 3, or 4) not to be written into the error_output I/O switch. The default value for N is 0. See "Error Diagnostics" in the tape_in command for further information on error reporting.

-check, -ck
specifies that only semantic checking be done on the Tape Control Language (TCL) control file. No tapes are mounted if this control is specified.

-force, -fc
specifies that the expiration date of a tape file to be overwritten is to be ignored. This control argument extends unconditional permission to overwrite a tape file, regardless of the file's "unexpired" status. This unconditional permission supresses any query made by the I/O module to inquire about tape file's expiration date.

-ring
mounts volumes of the volume-set with write permit rings (default).

TCL CONTROL FILE

The control file that governs file transfer for the tape_out command is written in the control file language described in the tape_in command.

ADDITIONAL OPTIONS AVAILABLE FOR THE TCL USER

A number of options are available to you if you want to do more than the simple file transfer between storage and a tape volume-set. These features need not be of concern to most users, but for the user with specialized needs, these additional options are explained below.

Protecting Tape File From Accidental Overwriting

To protect tape files from being accidentally overwritten tape_ansi_ and tape_ibm_ include expiration dates in the tape labels they write. The expiration local statement or Expiration global statement can be used in the TCL source file. To overwrite or delete a tape file the current date must be later than the expiration date specified in the tape label. If this is not the case, the attempt to destroy the tape file will fail and an error will be indicated unless the -force control argument has been specified in the tape_out command line. In that case expiration date checking will not be done.

Special Outer Modes

Normally, when you set up a TCL control file file-group to write a storage system file onto a tape volume that is for use with tape_out, it is intended that a new file be created on the tape volume. The TCL default output mode is create. This is the only output mode available for unlabeled tapes. For labeled tapes, however, the TCL language offers four additional specialized output modes: generate, modify, replace, and tape_extend. The replace mode causes the tape file labels to be rewritten using specified and default file structure attributes. The tape_extend, modify, and generate local statements do not cause the tape file labels to be recomposed, so any file attributes specified in the file-group or volume-group that do not match those recorded in the tape labels cause an error.

CONTROL FILE EXECUTION

When the TCL control file is being executed in response to tape_out, the volume named in each volume-group of the control file is mounted in turn with a write ring. Then each file-group in that volume-group is processed resulting in one file transfer to the volume-set per file-group.

FILE TRANSFER

File transfer is performed as follows. Either a line or a record is read from the storage system file depending on whether the file is unstructured or structured. For unstructured format storage system files, a line read is a line from the file up to, and including, the first newline character (NL) encountered; for sequential format storage system files, a record read is one logical record of the file. The characters read from the storage system are then written on the tape as one logical record of the tape file.

Under certain circumstances, tape records being written must be padded in accordance with a set of per-format padding rules (see the descriptions of tape_ansi_ and tape_ibm_). Because of padding rules and treatment of newline characters when writing tape, a file that is written out to tape may not appear the same when read back in from tape. The following suggestions are offered:

1. Use the defaults to write character data (i.e., source files or text files); with tape_ansi_, use d, db, s, or sb format with the maximum block length and choose the record length so that the amrl (the actual or maximum record length of a given record format) is greater than the longest line in the storage system file. Don't use f or fb format to avoid unwanted pad characters resulting from block padding.
2. Use the defaults with mode of binary or use s or sb format, with the maximum permissible block and/or record lengths and mode of binary, to write binary data with tape_ansi_.
3. Use vbs format with the maximum block length and choose the record length so that the amrl is greater than the longest line in the storage system file to write character data with tape_ibm_ (vb may cause one to three blanks to be appended to lines).

4. When transfering sequential format files to tape, use a variable length record format (d, db, s, or sb with tape_ansi_ and v, vb, or vbs with tape_ibm_) to avoid unwanted padding characters being inserted into records. (vb may cause one to three blanks to be appended to lines.)

EXECUTION TIME DIAGNOSTICS

Any fatal error from an I/O module during execution of a control file causes you to be queried whether or not you wish to continue processing the other file-groups and volume-groups in the control file or whether to terminate processing of the control file. In the case of some correctable errors you are given the alternative of "controlling the process." This alternative places you at command level allowing resolution of the problem. When you wish to continue processing, the start command is used. Executing the release command causes the tape_out command to be terminated.

CONTROL FILE EXAMPLES

Below are examples of typical control files. In the first example, you wish to produce two tapes, one for the Multics system, the other for an Os installation. The Multics tape contains the source code of user subsystem SUBSYS, as well as its object code. The OS tape contains only the source code.

EXAMPLE: SAMPLE1.TCL

```
  !  tape_out samplel.tcl

1          Volume: 001234;
2          /* Dump source in DB and object in SB format */
3          File: FILE_1;
4          path: SUBSYS.pll;
5          File:  FILE_2;
6          mode: binary;
7          path: <object>SUBSYS;
8          format: SB;
9          End;
10         Volume: DFG054;
11         /* append source to tape */
12         Tape: ibmsl;
13         File:  TESTSAVE;
14         format: VBS;
15         block: 4096;
16         path: SUBSYS.pll;
17         number: 3;
18         End;
```

1. mounts volume 001234 with a ring. The volume defaults are set to ANSI standard labeled tape-type, 800 bpi density, ASCII encoding mode, DB record format, block length = 2048, and record length = 2048.

2. is a comment in the control file. Since the storage statement is missing, the default storage system file format is set to transfer unstructured files.

3. since there is no number statement, the default positions the tape so that FILE_1 is created as a new file at the first file position on the tape volume.

4. specifies the pathname of the storage system file to be written to tape. Since the file-group contains no local statements, the file is written according to the current volume defaults.

5. positions the tape so that the file to be written is appended at file position two on the tape volume.

6. specifies that the file is to be written in binary encoding mode.

7. specifies the pathname of the storage system file to be written to tape.

8. specifies that the file is to be written in SB format. Notice that the block length is the current volume default block length (2048) and the record length is the current volume default record length (2048).

9. signifies end of volume-group. The I/O switch is closed and detached. The volume-set is taken down and the drive is released.

10. mounts volume DFG054 with a ring.

11. is a comment. Storage format is still unstructured.

12. changes tape-type to IBM standard labeled; changes the volume-group defaults to those associated with ibmsl: 1600 bpi, ebcdic, VB format, block length = 8192, and record length = 8188.

13. specifies name of file to be written onto tape. Notice that the underscore (_) cannot appear in an IBM file name whereas it can appear in an ANSI file name.

14. changes the record format to VBS. A spanned record format transfering a sequential file is needed, so that unwanted block padding is not inserted into the file as it is transferred. The default record length for VBS format is 1044480 bytes.

15. changes the block length to 4096.

16. specifies the pathname of the storage system file to be written.

17. This number statement is required to make sure the file is appended to an already existing tape volume. Without this number statement, the file would be created as the first file on the tape volume, overwriting any existing files. If files one and two do not exist, an error is indicated, but if these files do exist, the file is written at file position three on the tape volume.

18. rewinds and takes down the volume since no more file-groups in the control file reference the current tape volume.

EXAMPLE: SAMPLE2.TCL

```
  !  tape_out sample2.tcl -fc

1      Volume: 070067 -comment in_slot_1000, 070068;
2      Tape: ansi;
3      File: BIG_LISTING;
4      replace: FILE_20;
5      number: 20;
6      expiration: 2weeks;
7      format: db;
8      block: 2048;
9      record: 133;
10     path: >udd>Example>Mega>test.list;
11     End;
```

1. The first member of the volume-set, 070067, is mounted without a ring, displaying the message "in_slot_10000" on the operator's console. Later if necessary, the volume-set member 070068 can be mounted to continue writing a large listing file. A message appears upon mounting the second member of the volume-set.

2. writing an ANSI standard tape.

3. tape file named BIG_LISTING, into which the storage system file is to be written.

4. is to replace tape file named FILE_20.

5. by the number statement FILE_20 is the 20th file on the current volume-set. As no density statement is included in the control file, the default for tape_ansi_, 800 bpi, is used. Upon execution of the control file, the tape is positioned at the 20th file automatically, providing 20 files exist on the tape. As no Storage statement is present in the control file, the default storage system format is unstructured, and as the files are written to tape, the NL character is stripped.

6. The file, BIG_LISTING, is protected against accidental overwriting for two weeks, meaning that if you attempt to overwrite the file within that time, you are first queried for permission to do so. The -force control argument in the command line inhibits a query for permission to overwrite FILE_20, in case it has not yet expired.

7. BIG_LISTING is recorded in variable length blocked record format. Mode is the default for tape_ansi_, namely ascii.

8. Block length is maximum allowed for ANSI interchange standard, 2048.

9. record length is 133.

10. the listing file is transferred from test.list in the storage system.

11. signifies termination of volume-group and of control file.

If, after putting the listing file out onto tape, you wish to delete the on-line listing, and at a later time, read the listing back from tape into storage, you can type:

```
! tape_in sample2.tcl
```

The output statements in the control file, namely the replace local statement and the expiration local statement are ignored on input.

Name: teco

SYNTAX AS A COMMAND

```
teco {path1} {path2}
```

FUNCTION

provides a basic set of requests for creating and editing ASCII text segments and an extensive macro facility for creating sophisticated text-editing request combinations. It is a character-oriented text editor.

ARGUMENTS

path1
: is the pathname of a text segment to be read into the teco text buffer. If it is not specified, the buffer is initially empty and you can read in or enter text segments from the terminal.

path2
: is the pathname used when writing out the text. If it is not specified, the buffer is initially empty and you can read in or enter text segments from the terminal. If not used, path1 is used when writing out the segment. (See "Start-Up Macro" below.)

teco

OVERVIEW OF teco

The teco editor implemented for Multics is modeled after the TECO in general use on the Digital Equipment Corporation PDP-10, which was originally written at MIT's Artificial Intelligence project. The teco editor allows simple editing requests on a line basis as well as a character basis. In addition, iterative and conditional facilities are provided for writing macro definitions. These permit you to do simple manual editing of ASCII files or to write complex macros that do automatic editing. Although this implementation is modeled after the teco editor in general use, many new requests and features have been added that make the macro facility more powerful and easy to use. Some of the additions include adding if...then...else... statements, allowing the contents of Q-registers to be used as quoted strings; allowing numeric and string arguments to be passed to macros; allowing searches using regular expressions, automatically executing a start_up macro whenever teco is invoked; and allowing macros that reside in files to be called directly from the editor.

The line-oriented features of teco are similar to those of the edm and qedx commands. The character-oriented requests use a pointer that can be positioned between any two characters in the buffer, permitting insertion, deletion, and so on of characters without the need to retype the line.

The teco editor reads request lines from your terminal line by line until a line ending with a dollar sign ($) is typed. Execution of the complete request string is started when this line is read. The teco editor types a prompt when it is waiting for a new request string. To exit from the editor, type the EQ request (followed by $ and a newline).

MACRO USAGE

Syntax as a Command

```
teco$macro macro {macro_arguments}
```

Arguments

macro
is the name of a teco macro to be executed when the editor is invoked.

macro_arguments
are optional arguments processed by the macro invoked. This entry point is provided for users who write teco "programs" that are intended to run without ever reaching teco request level. The command line:

```
! teco path1 path2
```

is equivalent to:

```
teco$macro start_up {path1} {path2}
```

teco STORAGE AREAS

The teco editor uses four storage areas:

The buffer

an area where text to be edited is examined and modified. At all times it contains a (possibly null) character string. There is a pointer into the buffer, denoting the current position. This pointer does not point to a character; it points between two characters. The pointer can assume any value between 0 and Z, where Z is the number of characters currently in the buffer. 0 indicates that the pointer is to the left of the first character, and Z represents the position to the right of the last character in the buffer. The value of the pointer is represented by ".".

Request String Area

editor requests are read into the request string area as a continuous character stream for subsequent parsing into operational requests. Uppercase and lowercase letters can be used interchangeably in requests.

Q-registers

locations for storing either numeric quantities or strings of text for later use. Each Q-register is designated by a single character name. There are 95 Q-registers, one for each printing ASCII character. Each Q-register can contain a positive or negative integer or a character string.

Q-register pushdown list

a Last-In-First-Out (LIFO) list that can be used to temporarily store the contents of a Q-register. It is cleared (i.e., the contents are lost) every time you return to teco request level.

NUMERIC EXPRESSIONS

The teco editor uses numeric expressions for many of its operations. These consist of operators and operands. Operands can be decimal numbers, octal numbers, teco requests that return values, teco macros that return values, or teco special symbols. Operators are unary minus (-), arithmetic binary operators addition (+), subtraction (-), multiplication (*), division (/), and the boolean binary operations or (|), and and (&). All operators are of equal precedence and expressions are evaluated from left to right. Notice, however, that parentheses can be used in their normal manner. Spaces are ignored except to terminate numbers. If two numeric quantities are given with no operator between them, the default operator + is used. Notice that a string of digits followed immediately by a "." is interpreted as an octal rather than a decimal number. Division using the "/" operator is integer division, i.e., the remainder is

ignored. The special symbols allowed in an expression at any point are:

B (Beginning) equivalent to 0.

Z equivalent to the number of characters in the buffer.

. equivalent to the current value of the pointer or the number of characters to the left of the pointer.

H (wHole) equivalent to 0,Z. It is the only symbol to have two values. It is useful for referring to the entire buffer.

Requests that return values can also be used in expressions, but they cannot appear immediately to the right of an operator if it requires arguments. This is because requests that take arguments assume that everything to its left is part of one of its arguments. If a request appears within parentheses, its arguments are entirely contained by the closest left parenthesis that encloses the request. A request does not read parts of an expression outside the parentheses in which it is enclosed.

The plus and minus binary operators assume a right operand of 1 if none is given.

The examples below show the evaluation of numeric expressions in teco. Assume that the current value of the pointer is 500.

```
     expression                value

(1)  (7 12)/3                   = 6
(2)  9+                         = 10
(3)  b-                         = -1
(4)  -                          = -1
(5)  4+8/2                      = 6
(6)  101.                       = 65
(7)  3|10                       = 11
(8)  1++++ ++ +++    +          = 11
(9)  9*-2                       = -18
(10) 9*--2                      = 18
(11) .10                        = 510
(12) 10.                        = 8
```

NOTES ON QUOTED STRINGS

Quoted strings are strings of text delimited by a quoting character. The quoting character can be any character not contained in the string except a letter or a digit. The contents of a Q-register can be used as a quoted string if the letter "q" followed immediately by the letter specifying the Q-register is typed instead of the first quoting character. The following examples show valid quoted strings:

(1) "hello"
(2) /This is a quoted string/
(3) ,This string is delimited by the comma character and contains 2 newline characters.
,
(4) q1

NOTES ON ERROR MESSAGES

Error messages are printed by teco in one of two modes: long or short. Short error messages are from one to eight characters long while long error messages are less than 50 characters long. The default mode is short. To change the error mode teco is using, give the following Multics commands:

```
   !  teco$teco_error_mode long
or:
   !  teco$teco_error_mode short
```

If a short error message, such as "/: ?", cannot be understood, the following Multics command prints the long error message:

```
   teco$teco_error "/: ?"
```

The above holds for teco error messages only.

IMPLEMENTATION RESTRICTIONS

The maximum number of characters allowed in a Q-register, in a quoted string, or in a teco request line is 1044480 characters. Notice that these sizes are all one segment long. When the Multics segment size changes, these restrictions also change. The maximum number of items in the pushdown list is 20. The maximum depth of macro calls is 20. The maximum depth of parentheses is 20.

LIST OF REQUESTS

The teco editor requests have the basic form:

```
   m,nX/string/
```

where m and n are optional numeric arguments, X is the request to be executed, and /string/ is a quoted string. In most cases, the request is just one character, though in some cases, it is two characters. Not all of the requests take arguments. Those that do generally have default values for missing arguments. Only a few requests expect quoted strings. The string must not be omitted if the request expects one. Some requests also return values; this is discussed later in "Advanced teco Commands."

Some letters chosen for requests have mnemonic meanings, which are indicated in the description of each request. Unfortunately, teco has a fairly long history, having originally been developed for editing paper tapes, and so some of the mnemonic meanings are lost now. As many requests as one wishes can be typed at a time.

Execution of the requests does not start until after a line is typed ending with a "$". Spaces can be inserted anywhere except in the middle of numbers, and newline characters can be inserted anywhere except between a request and its arguments. Uppercase and lowercase letters can be used interchangeably as requests.

Reading a File--EI (External Input)

EI/pathname/
reads in the file specified by pathname, which is assumed to be a standard Multics pathname. The contents of the file are inserted in the buffer at the current pointer position, and then the pointer is moved to the right of the text inserted.

Writing a File--EO (External Output)

EO/pathname/
is equivalent to HEO/pathname/. It writes the contents of the entire buffer to the file specified by pathname. This request takes arguments similar to the T request; it writes out that part of the buffer that would be printed by T. However, if no arguments are given, EO assumes B, Z as the default rather than 1. NOTE: The pointer is never moved by the EO request.

Typing the Buffer--T (Type)

T
is equivalent to 1T

nT n > 0
prints the buffer beginning at the current pointer position and terminating after n newline characters have been encountered. T prints the rest of the current line, and 2T prints the rest of the current line and the next line. The last character printed by T is a newline. If n is greater than the number of new line characters to the right of the pointer, all text to the right of the pointer is printed.

nT n <= 0
prints the characters between the (-n+1)th newline character and the pointer. The (-n+1)th newline character is not printed. If (-n+1)th is greater than the number of newline characters to the left of the pointer, all text to the left of the pointer is printed. 0T prints the beginning of the line up to the current pointer. -T prints the previous line and the beginning of the current line. If the pointer is at the beginning of a line, -T prints the previous line.

m,nT
prints the (m+1)th through the nth characters of the buffer.

NOTE: The pointer is never moved by the T request. Usually two T requests are given at once, such as 0tt, which prints the entire line that the pointer is in.

Moving the Pointer--J (Jump), C (Characters), R (Reverse), and L (Lines)

nJ
moves the pointer to the right of the nth character in the buffer, i.e., sets "." to the value of n. If n is not specified, 0 is assumed. That is, the pointer is moved to the left of the first character in the buffer. The value of n must be from 0 to z.

nC
moves the pointer n characters to the right of its current position (equivalent to .+nJ). If n is omitted, 1 is assumed. The new value of "." must be from 0 to z.

nR
like nC except it moves the pointer to the left (equivalent to -nC). If n is omitted, 1 is assumed. The new value of "." must be from 0 to z.

nL n >0
positions to the beginning of a line. Moves the pointer to the right, stopping after it has passed over n newline characters. If n is omitted, 1 is assumed. L moves the pointer to the beginning of the next line. There must be at least n newline characters to the right of the pointer.

nL n <= 0
moves the pointer to the left, stopping after it has passed over (-n+1) newline characters and then moving it to the right of the last newline character passed over. 0L moves the pointer to the beginning of the current line. -L moves the pointer to the beginning of the previous line. There must be at least (-n+1) newline characters to the left of the pointer.

Deleting Text--D (Delete) and K (Kill)

nD
deletes n characters. If n is positive, the characters are deleted to the right of the pointer. If n is negative, the characters are deleted to the left of the pointer. If n is omitted, 1 is assumed. If n is zero, nothing is deleted.

K
takes arguments like the T request except it deletes the text T prints. The pointer is moved to where the deletion took place. If no arguments are specified, 1K is assumed.

K n > 0
deletes all the characters beginning at the current pointer position and terminating after n newline characters have been encountered. There must be at least n newline characters to the right of the pointer. K deletes the rest of the current line and the newline character at the end of the line, while 2K deletes the rest of the current line and the next line. 0LK deletes the current line as does 0KK.

K n <=0
deletes all the characters between the (-n+1)th newline character and the pointer. There must be at least (-n+1) newline characters to the left of the pointer. 0K deletes the beginning of the current line without deleting the newline character at the end of the previous line. -K deletes the previous line and the beginning of the current line. To ensure that only the previous line is deleted, the request sequence 0L-K is used.

m,nK
deletes the (m+1)th through the nth characters of the buffer. The pointer is moved to m. Equivalent to mJ n-mD. HK deletes the entire buffer.

Inserting Text--I (Insert)

I/text/
inserts the text of the quoted string at the current pointer position and moves the pointer to the right of the inserted text. /text/ can also be specified as a Q-register, for example, Iq2.

nI
inserts the character whose ASCII code value is n. It moves the pointer to the right of the inserted character.

Search for Text--S (Search)

S/string/
is equivalent to 1S/string/

nS/string/
searches for the nth occurrence of the quoted string. If n is positive, the text is searched from the current pointer through the end of the buffer for the nth occurrence of the string. If found, the pointer is set to the right of the matching string. Otherwise, the pointer is not moved, and an error message is printed. If n is negative, the text is searched from the current pointer position to the beginning of the buffer for the (-n)th occurrence of the quoted string. The pointer is set to the left of the matched string. If the string is not found, the pointer is not moved, and an error message is printed.

m,nS/string/
searches m lines from the current pointer for the nth occurrence of the quoted string instead of searching the entire buffer. If m is positive, n must be positive,

and the only part of the buffer that is searched is from the current pointer to just after the mth newline character after the current pointer. If m is 0 or negative, n must be negative, and the only part of the buffer that is searched is from the current pointer to just after the (m+1)th newline before the current pointer. 1,1S/text/ only searches the rest of the current line. 0,-1S/text/ only searches the beginning of the current line.

Search for Regular Expression--N

N/string/
is equivalent to 1N/string/; searches from the current pointer position through the end of the buffer for the first occurrence of the regular expression, string.

The term "regular expression" refers to the character string used to address a line of text that contains that string of characters. In its simplest form, a regular expression is a character or string of characters delimited by the right slant character (/). For example, in the following text, the regular expression /abc/ matches line 2:

```
a:procedure
  abc = def
   x = y
    end a
```

nN/string/
searches from the current pointer position to the end of the buffer for the nth occurrence of the regular expression, string. The value of n must be greater than 0.

m,nN/string/
searches the next m lines for the nth occurrence of the regular expression. The values of m and n must be greater than 0.

Printing Values--= (Equals)

n= or: m,n=
prints the decimal value of its arguments, separated by spaces and followed by a newline.

n:= or: m,n:=
prints the octal value of its arguments, separated by spaces and followed by a newline.

Leaving teco--EQ (External Quit)

EQ
returns to the caller of teco (e.g., Multics command level). (Remember to do an EO request before the EQ if the editing is to be saved.)

Restarting teco After a Quit

If a quit signal is used to abort a request string, the Multics program_interrupt (pi) command can be used to restart the teco editor. Issuing a quit does not abort the entire command string; only those commands not yet executed. The current request is aborted when it is completed.

At times it is desirable to get around this feature. When doing an EO, for instance, teco does not allow you to return to teco request level until it has completed writing the file. To get around this, type:

```
(quit)
teco$abort
```

When teco$abort is called, the most recent invocation of teco aborts its current operation without checking for consistency of states. This is useful if an E0 request fails because of insufficient access. Using the program_interrupt command would cause teco to reattempt the write. Notice that teco is in a consistent state whenever it actually accesses a file, and so there should be no problems encountered if this feature is used to get out of an E0 request. Under other circumstances, however, it is wise for you to type:

```
-5t5t
```

to ensure that control is maintained. Except for the case of an unsuccessful E0 request, this feature should not be used.

STAND-ALONE EXAMPLES

Entering teco:

`teco source.pl1`
enters teco and reads in the file source.pl1 from the working directory.

`teco <x>y>z>a.ec`
enters teco and reads in the file specified.

`teco`
enters the buffer initially empty.

`teco >t>start_up.teco start_up.teco`
enters teco and reads in `>t>start_up.teco`. Q-register * is set to start_up.teco.

teco

Reading a File:

EI/source.pll/
inserts the text contained in source.pll at the current point in the buffer.

Writing a File:

EO/new_source.pll/
writes the whole buffer out into new_source.pll.

.,zEO/bottom/
writes out the buffer from the current pointer to the end into the file named bottom.

2EO/lines/
writes out two lines starting at the current pointer position to the file named lines.

Printing Text:

2T
prints from the pointer to the end of the next line.

0T
prints the current line from its beginning to the pointer.

0TT
prints all of the current line.

25,100T
prints the 25+1 (26th) through the 100th character of the buffer.

Moving the Pointer:

J
positions the pointer at the beginning of the buffer.

ZJ
positions the pointer at the end of the buffer.

L
positions the pointer at the beginning of the next line in the buffer.

0L
positions the pointer at the beginning of the current line.

-L
positions the pointer at the beginning of the previous line.

R
backs up the pointer by one character position.

812-388C
moves the pointer ahead 812-388 (424) character positions.

Deleting Text:

19,22K
deletes the 19+1 (20th) through the 22nd character of the file. Sets the pointer to 19.

19J 3D
moves the pointer to the right of the 19th character and then deletes the next three characters (20-22).

HK
deletes the whole buffer.

-D
deletes the character just to the left of the pointer.

Inserting Text:

I/abc
/
inserts the line abc followed by a newline character at the current pointer position.

I.abc.
inserts the string abc without a newline character.

65I
inserts the character with ASCII code 65 (A) at the current pointer position.

Printing Values:

Z =
prints how many characters are in the buffer.

Z,.=
prints how many characters are in the buffer followed by the current pointer position.

=
prints a newline character.

Q6+53 =
prints the value 53 plus the value contained in Q-register 6.

Searching for Text:

J S/Hello/
positions the pointer just to the right of the first occurrence of the string Hello in the buffer.

Z J -S"Hello"
positions the pointer just to the left of the last occurrence of the string Hello in the buffer.

J 3S"*
"
positions the pointer just after the third occurrence of a line ending with an asterisk (*).

J 1,1S/Hello
/
positions the pointer just after the first line in the buffer if it ends in Hello. If the first line does not end in Hello, prints an error message.

EXAMPLES OF BASIC EDITING REQUESTS

```
teco abc.pl1
```

enters teco and reads in the segment abc.pl1.

```
5LT$
```

moves to the 6th line and prints it out.

```
dcl a fixed bin;
```

```
S/a/-D1/b/0LT$
```

changes the "a" to a "b" and prints the line.

```
dcl b fixed bin;
```

```
S/dcl d/0LKT$
```

searches for "dcl d" and deletes the line that contains it. Then prints out the next line.

```
dcl f fixed bin;

Kl/dcl g char(2);
/$
```

deletes this line and then insert a declaration of g.

```
EO/abc.pl1/EQ$
```

writes the edited text out to the file and then returns from teco.

ADVANCED COMMANDS

In "List of Requests" above, the general form of a teco request was given. Some items were left out, however. A more complete format is:

```
m,nXq/string1//string2/.../stringN/
```

The q indicates a Q-register on which the request is to act. It should also be noted that more than one string can be given. Although no teco request currently accepts more than one quoted string, a macro can be called with multiple string arguments that can be retrieved inside the macro by the :X request.

"Numeric Expressions" specifies that expressions can be built from numbers, special valued requests, and symbols. Examples of valued requests are given in this section. Notice that requests with values that require arguments only appear on the left side of the first operator, or within parentheses. Otherwise, the part of the expression preceding the request is considered to be an argument to the request.

The effect of many requests can be changed by preceding the request with a colon (:). The colon has no fixed meaning--it is defined for each request individually. The following requests given earlier have the following changed effect:

:Iq/string/ or n:Iq
: is similar to the I request except that the specified string is inserted into Q-register q instead of the buffer. The former contents of Q-register q are lost.

n:L
: is equivalent to nLR. :L moves to the end of the line rather than the beginning.

:S/string/
: is similar to S except that it returns a value. The value is 0 if the search fails and -1 if it succeeds. Even if the search fails, teco continues execution.

:n/string/
: is similar to N except that it returns a value. The value is 0 if the search fails and -1 if it succeeds. Execution continues even if the search fails.

:T/string/
: prints the specified string on your terminal. This request takes no arguments.

:=
is identical to = except it prints values in octal instead of decimal.

:EI
is similar to EI except that it returns a value. The value returned is -1 if the read succeeds and 0 if the read fails. No error is printed if the read fails.

:J,n:J
is similar to J except that errors cannot occur. If n is less than 0, the pointer is moved to the beginning of the file. If n is greater than Z, the pointer is moved to the end of the file.

:C,:R
are similar to C or R except that errors cannot occur. If the pointer would be moved to before B, move it to B. If the pointer would be moved beyond Z, move it to Z.

NUMERIC Q-REGISTERS

Q-registers can be used to hold numeric values. These values can be used in expressions.

Saving a Value--U (Update)

Uq
sets Q-register q to a very large positive number.

nUq
sets Q-register q to n.

m,nUq
sets Q-register q to n and returns m as its value.

Reading Q-Registers--Q (Q-register)

Qq
returns the number stored in Q-register q as the value. Q is not a request—it is a special symbol. Thus, in the expression 5+Q3 the 5+ is not considered an argument to Q; the result is the sum of Q3 and 5. If Q-register q contains text, the length of the text, in characters, is returned.

Incrementing Q-Registers--%

%q
adds 1 to Q-register q and returns the new number as the value. Q-register q cannot contain text. Note that % is a special symbol, too.

TEXT Q-REGISTERS

Q-registers can also be used to hold character strings, to move text from one place in the buffer to another, to save request lines for execution as macros, or to provide quoted strings.

Extracting Text to a Q-Register--X (eXtract)

Xq
takes arguments like the T request, but copies the text into Q-register q. The former contents of Q-register q are deleted. The text is not deleted from the buffer, and the current pointer is not moved.

nXq n>0
copies all the text from the current pointer to past the nth newline character to the right of the pointer into Q-register q. X1 copies the rest of the current line including the newline at the end of the line into Q-register 1. 2Xa copies the text on the rest of the current line and all the next line into Q-register a.

nXq n<=0
copies the characters between the (-n+1)th character and the pointer. The (-n+1)th newline character is not copied. 0X/ copies the beginning of the current line into Q-register /. In this case, no newline character is put into Q-register /. -Xa puts the previous line and the beginning of the current line into Q-register a.

m,nXq
copies the (m+1)th character through the nth character into Q-register q.

Appending Text to a Q-Register--P (aPpend)

Takes arguments like the X request, except it appends to the former contents of the q-register instead of deleting the former contents. The text is not deleted from the buffer and the current pointer is not moved.

Inserting Text Directly into a Q-Register--:I (Insert)

:Iq/string/
is similar to the normal I request except that the text is inserted into Q-register q rather than the buffer. The former contents of Q-register q are deleted. The text buffer is not affected.

n:Iq
is similar to :I except that it puts the character corresponding to n into the Q-register q.

Getting Text from a Q-Register--G (Get)

Gq
: inserts the text contained in Q-register q into the buffer to the left of the current pointer. If the Q-register contains a number, the decimal representation of the number is inserted.

OBTAINING QUOTED STRINGS FROM Q-REGISTERS

Whenever teco expects a quoted string, it is possible to indicate that the string is in a Q-register. Normally, letters and digits are considered invalid quoting characters. If, however, the letter Q is found where a quoted string is expected, the next character after the Q is considered a Q-register name. Whenever a quoted string is retrieved by any request, it is loaded into Q-register ". As an example, SQ", immediately after another search, searches again for the same string. This notation is invalid if the specified Q-register contains a number.

THE Q-REGISTER PUSHDOWN STACK

There is one Q-register pushdown stack (not one per Q-register) in which the values of Q-registers can be saved. It is organized as a pushdown (LIFO) list. It is emptied every time teco waits for a new request string.

Pushing a Value onto the STACK--[(opposite of])

[q
: pushes the current value of Q-register q onto the top of the stack. The Q-register is not affected.

Popping a Value from the STACK--] (opposite of [ )

]q
: pops the top value on the stack into Q-register q. The previous contents of the Q-register are lost. It is an error to do a] request if the stack is empty.

LOOPS

The teco editor has the ability to execute a request string repeatedly, just as FORTRAN or PL/I provides do-loops.

Loops--< and > (opposite of each other)

<
: begins a loop. It is equivalent to n< except that n is set to a very large number that is for all practical purposes infinite.

n<

begins a loop to be executed n times. The value of n and the position of the < in the request string are saved. The value of n must not be negative.

:<,n:<

is similar to < except that errors that occur within the iteration group just terminate the iteration group and the > returns a value. The returned value is -1 if no errors occurred, and it is 0 if the group was terminated by an error. The error message that terminates the loop is not printed.

>

ends a loop. It returns to just after < if the string has not yet been executed n times.

n<...>

executes the string between the angle brackets n times.

Terminating a Loop Before n Executions--;

n;

if n is less than 0, then nothing is done. Otherwise, execution of the current loop is aborted and teco skips to just after the closing >. If n is not specified, the result of the most recent S or N request is used (terminate loop if search failed). The ; request cannot appear outside of a loop.

:;

is similar to ; except that the sense of the test is inverted. If n is less than 0, execution of the current loop is terminated and teco skips to just after the corresponding >.

Special Loop Facilities--throwing and catching values

F<!label!

provides for a nonlocal transfer of control. F< and > define an iteration group like < and >. From the time that the F< iteration group is entered until the time it is exited, F< sets up a handler to "catch" values "thrown" by the F;/label/ request. If no F;/label/ request with matching-string argument is executed before the F< iteration group is exited, the iteration group returns -1 as a value. If, however, an F;/label/ request is executed (where the label string matches the one in the F<!label!), the execution of all macros and iteration groups encountered since the F< is abandoned, and the F< iteration group returns the numeric argument of the F; request as a value.

:F<!label!

is similar to F< except that if an error is encountered during the execution of the :F< iteration group, the latter returns zero as a value.

nF;/string/
"throws" the numeric value n to the most recent F< or :F< iteration group where the string argument matches the string argument of the F; request. It is an error to execute a F;/string/ request when there is no F< or :F< iteration group in execution with a matching-string argument.

NOTE: These requests provide a method of exiting several nested loops at once. Execution of a F; request terminates the F< loop as well as any contained loops.

GOTOS

The teco editor provides the ability to transfer control to a different part of the request string.

Goto--O (gOto)

O/string/
searches the current macro (or, if we are not in a macro, the request line) for the label !string!. If it is found, teco begins interpreting requests just after the label. If not found, but execution is currently in a macro, the search is repeated in the previous execution level, i.e., the caller of the macro. This is repeated until teco has checked all the way down to the request line typed by you. Notice that although teco can exit a macro using an O request, it cannot use that request to exit a loop. Only a semicolon (;) can be used to terminate a loop.

MACROS

The teco editor has the ability to execute strings of text (macros) other than those read from your terminal. The associated requests are listed below:

Executing A Macro in a Q-Register--M (Macro)

Mq
executes the contents of Q-register q as a request string. Notice that if the M request is given any numeric arguments, they are passed to the first request inside the macro. String arguments can be fetched by the :X request.

:Mq
is similar to the M request except that if issued within a macro, the return from Q-register q causes the invoking macro to return also.

Executing A Macro in a File--EM (External Macro)

M/string/
is similar to the M request except that the request string is found in a file whose entryname is string.teco. This file is looked for in three directories: the working directory, your login directory, and the teco library.

Obtaining a String Argument to a Macro

:Xq

suspends execution of the current macro, returns to its caller to fetch a quoted string into Q-register q, and then restores the macro that was being executed. Notice that each :X request in a macro fetches another quoted string. The U request(s) should be the first request in a macro if one wishes to fetch numeric arguments in a macro.

NOTES

1. Loops cannot cross macro boundaries, i.e., a loop cannot start in one macro and end in another. This does not, however, prohibit the M or EM request from being used within a loop.

2. A macro can modify itself if it is in a Q-register. Notice, however, that the current invocation of the macro is not affected; only future accesses to the Q-register. If the macro is invoked by the EM request, the results of modifying the file are hard to predict as teco reads the request string directly from the file.

3. When a macro is invoked by the EM request, it should be noted that the name of the macro is found in the Q-register named ". Thus several macros can be put in one segment with the first request in the segment being OQ". (Do not forget to put all the appropriate names on the segment.)

4. If an M or EM request is given as the last request in one macro, the request is interpreted as a goto rather than a call. Thus, unlimited M's can be done in this manner although there is an implementation-defined limit to the depth of calls.

5. When the teco editor is entered, a macro named start_up is searched for. If it is found, the arguments to teco are put onto the pushdown stack, and the start_up macro is executed. There is a default start_up macro if you do not provide yours. This macro is described below.

CODING CONVENTIONS FOR MACROS

Since there are only a small number of Q-registers (95), each with a one-character name, there are serious problems in writing a set of macros that are compatible. A set of macros become incompatible if one macro uses a Q-register for long-term storage that any other macro uses at all. There are two ways this effect can be combated. First, by establishing certain coding conventions, and second, by use of a documented macro library. Probably the most important coding convention is the specification of which Q-registers can be used inside a macro for temporary storage. Many library teco macros use the ten Q-registers 1,2,3,4,5,6,7,8,9, and 0 for temporary storage. If one macro calls another macro that destroys the contents of one of these registers, the calling macro can save the value of the Q-register in the pushdown list and then restore it after the other macro has been called.

Fortunately, calling a macro is a very inexpensive operation in teco if the macro is in a Q-register. The EM request is more expensive. This leads to the practice of creating a macro in a macro library that loads a Q-register with a useful macro. Realizing that you want the macro, you give the EM request that loads that macro into a Q-register, where you can then call it anytime. It now becomes necessary to have coding conventions that specify which registers can be loaded permanently with macros. Since it should be easy to type the macro names, the lowercase alphabetic letters should be used for this purpose. Sometimes a macro uses a Q-register for long-term storage. If you do not have to type the name of this Q-register, names that must be escaped are good; otherwise, other special characters can be used. This leaves the uppercase alphabetic letters entirely for you to store intermediate results in editing. Besides, the special characters -, ,, ., /, space, tab, and newline should be reserved for you since these are all lowercase on most terminals.

An extremely useful feature of teco is that the last quoted string is loaded into Q-register ". To allow this to continue to be useful, all macros should make sure that Q-register " either contains the last quoted string argument to the macro, if there are any, or contains what it contained before the macro was called. Q-register " can be saved on the pushdown list on entry to a macro and then restored just before leaving the macro. Use of the pushdown list is very inexpensive.

RELATIVE COSTS IN TECO

The teco editor stores the buffer in two pieces. The first piece, all the characters from the beginning of the buffer to the current pointer, is stored at the beginning of one buffer segment. The second piece, all the characters from the current pointer to the end of the buffer, is stored at the end of another buffer segment. Inserting text merely adds text to the end of the first buffer segment and increases the number of valid characters in the first buffer segment. Deleting text merely changes the number of valid characters in one or both of the buffer segments. In order to move the pointer, a string copy from one buffer segment to the other is performed unless an unmodified copy of the string already exists in the other buffer. It does not matter to teco which direction the pointer is moved.

Reading a file into an empty buffer causes that file to be used as the buffer until the text is modified. Thus this request string causes an invalid segment fault:

```
HK EI/File/ EC/dl File/$
```

Positioning to the end of the buffer (ZJ) puts all the text into one temporary segment. Thereafter, pointer moves do not actually move text. As long as all the text remains in one temporary segment, pointer moves do not actually move text. An insertion or deletion anywhere but at the end of the text causes the text to be split up.

Each text Q-register is presently kept in its own segment. This means that if a start_up macro loads many Q-registers with macros, entering teco for the first time in a process is somewhat slow since all these segments must be created. The teco command has its own segment manager (get_temp_seg_) that allows it to reuse segments without calling hardcore to create and delete segments when the values of

Q-registers are changed. Whenever a string is quoted, or a Q-register loaded with text, a new segment is retrieved from get_temp_seg_ and loaded with the value. If the string that is being loaded into the Q-register is in another Q-register, the new Q-register is just made to point to the same copy of the text in the first Q-register. :IAQB is therefore a very simple operation, as are [(Push) and] (Pop). The feature of keeping the last quoted string in Q-register " lets you take advantage of this scheme.

If you want to write a macro that must do some editing on another file, it is much cheaper if you save the value of "." and "Z-." , insert the text to be edited, edit it, write it out or copies it into a Q-register, and then delete what you were just editing from the buffer. The net change to the buffer by all these operations is zero, but the text that you were editing was never moved. This method is much cheaper than storing the entire buffer in one Q-register, the value of the pointer in another, and then using the buffer for the editing within the macro.

There are four ways to transfer control in teco, by the > request, the ; request, the " or :' request, and the O request. Of these, the > request is the fastest, since teco already knows exactly where to transfer it. The ;, ", and :' requests are next, since they merely search from where they are. Although the > request and the ; request cannot change macro levels, the ", and :' requests can. This adds a small expense. The ; and :; requests have to check so that a ; request completely skips over another nested loop and looks beyond it for a >. Similarly, the " transfer skips over nested if statements, as does the :' request. Usually, the matching ' or > is not far from the transfer, so this only causes a short search.

WARNING: The teco editor implements < and > searches very simply. It does not check the semantics of the request string. The request string is searched forward for the first < or >. If a < is encountered, a counter is incremented. If a > is encountered and the counter is zero, the search is complete; otherwise the counter is decremented. Any and all < and > appearing in the searched portion of a request string participate in this process.

O is the most general and most expensive transfer of control in teco. It must search the entire macro from the beginning, then the entire macro that called the present macro, etc., until it finds it or finishes searching the request line and gives an error. Although this is the most expensive transfer, its cost is proportional to the distance of the label from the goto request.

CONDITIONALS

The teco editor has the ability to conditionally execute strings. The " request corresponds to the PL/I statement "if ... then do;". The ' request corresponds to the PL/I statement "end;". " and ' are matched and can be nested.

WARNING: The teco editor implements " and ' searches very simply. It does not check the semantics of the request string. The request string is searched forward for the first " or '. If a " is encountered, a counter is incremented. If a ' is encountered and the counter is

zero, the search is complete; otherwise, the counter is decremented. Any and all " and ' appearing the searched portion of a request string participate in this process.

The letter following the " determines what test is made.

Numeric Comparisons--"E (Equals), "N (Not equal), "G (Greater than), "L (Less than)

m,n"E
if m=n, then execution continues; otherwise, execution skips to just after the corresponding '.

n"E
is identical to n,0"E.

m,n"N
is similar to m,n"E except it tests for m ^= n.

n"N
is identical to n,0"N.

m,n"G
is similar to m,n"E except it tests for m > n.

n"G
is identical to n,0"G.

m,n"L
is similar to m,n"E except it tests for m < n.

n"L
is identical to n,0"L.

Testing for a Symbol Constituent--"C (Symbol Constituent)

n"C
if n is the ASCII code for a letter, a digit, or one of the characters ., _, or $; then execution continues; otherwise, execution skips to the corresponding '.

String Comparison--"M (Match)

"M/string/
if the specified string appears immediately to the right of the pointer, then execution continues; otherwise, execution skips to just after the corresponding '.

:"M/string/
is similar to "M/string/ except that the sense of the test is inverted.

Terminating a Conditional Do--' (Matches ")

'

is ignored when executed in normal execution. It is used to close a conditional statement.

:'

transfers to the next ', just as a 1"e does. Since this request looks like a ', it can serve to close a conditional statement. This is useful if an if...then...else... statement is desired. The if expression is a " statement, then the expression is terminated by the :' request and the else expression is terminated by the ' request. (See the warning under "Conditionals" above.)

Reading Input from the User's Terminal--VW (V then wait for input)

VW

does a V request (presently does nothing on Multics) and then reads one character from your terminal. The ASCII value of the character is returned as the value of the request.

:VWq

does a V request and then reads one line from your terminal. The line is put into register q. The newline is the last character read in.

Passing a Command to the Command Processor--EC (External Command)

EC/string/

passes the specified string to the Multics request processor for execution.

Invoking an Active Function--EA

EAq/string/

passes the specified string to the command processor's active function application entry. The result of the active function application is returned in Q-register q. The specified string must not include square brackets.

Examining a Character in the Buffer--A (ASCII)

nA

The ASCII code for the (.+n)th character in the buffer is returned as the value of the request. n must be specified. (Notice that 1 indicates the character just to the right of the current pointer; 0 indicates the character just to the left.)

Tracing Command Execution--?

?
turns tracing on. When tracing is on, each request executed by teco is printed on your terminal just before it is executed.

??
turns off tracing.

Translating Numbers to ASCII and Vice Versa--

\
reads the decimal number found to the right of the current pointer and returns its value as the value of the request. The pointer is moved to the right of the number. The number can be signed and can be preceded by any number of blanks or tabs. It is an error if no number is found.

n\
inserts the decimal interpretation of n into the buffer to the left of the current pointer.

m,n\
inserts the decimal interpretation of m into the buffer to the left of the current pointer. The interpretation is padded on the left to be at least n characters wide.

:\
is similar to \ except that it converts to and from octal representations of numbers.

Null Command--W

W
does nothing. It is most useful for throwing away unneeded numeric arguments.

newline character
has the same effect as W.

$
has the same effect as W.

EXAMPLES OF MACROS

Write Macro

*.
This macro writes out the entire buffer into a file whose name is in Q-register *. The pathname is changed by doing:

```
:i*/new_name/
```

EOQ*
assumes that the name of the file we are editing is in Q-register *. It writes out the entire buffer into this file.

A Restart Macro

This macro zeros out the buffer, changes Q-register * to be a new file name and reads the file into the buffer:

```
:x* hk eiq* j
```

:X*
takes one string argument and loads it into Q-register *.

HK
deletes all the text in the current buffer before editing is restarted.

EIQ*
reads the new file into the buffer.

J
puts the pointer at the beginning of the buffer.

Start-Up Macro

This macro only uses the first and second argument to teco. It treats it as a file name, loads it into Q-register * and reads the file into the buffer. It also loads the writing macro into Q-register w:

```
]1 :iw|eoq*|ifqwq1"n ]* eiq* j q1-1"q ]*'
```

]1
pops the top item off the pushdown list and puts it into Q-register 1. This is the number of arguments teco was called with.

```
:iw|eoq|
```
loads Q-register w with the write macro given in the above example.

```
:ifqw
```
loads Q-register f with a copy of the contents of Q-register w.

```
q1"n
```
if the contents of Q-register 1 are not zero, then execute the following statements; otherwise, transfer to the matching '.

```
]*
```
pops the first argument to teco off the pushdown list and into Q-register *.

```
eiq*
```
reads the file whose name is the contents of Q-register *, into the buffer.

```
j
```
moves the pointer to the beginning of the buffer.

```
q1-1"q
```
if the value of the expression (q1-1) is greater than zero, execute the following; otherwise, skip to the matching '.

```
]*
```
pops the next (second) argument off the pushdown list and into Q-register *.

```
'
```
matches q1-1"q. End of request string for second argument.

```
'
```
matches q1"n. End of request string for processing arguments.

Substitute Macro

This macro takes two string arguments. The first string argument is searched for, then it is deleted, and the second string inserted:

```
:x1 :x2 sq1 -q1d g2
```

:x1
loads the first string argument into Q-register 1.

:x2
loads the second string argument into Q-register 2.

sq1
searches for the first string.

-q1d
deletes the first string when it is found. (Could also be -q"d.)

g2
replaces the string found with the second string argument.

When the macro returns Q-register, 1 and 2 contain the first and second strings, respectively. Q-register " contains the second quoted string.

SUMMARY

NAME	USE AND EXPLANATION
a	nA The value of the request is the ASCII code for the (.+n)th character in the buffer.
b	B The value of this symbol is always zero.
c	nC moves the pointer n characters to the right. If n is omitted, 1 is assumed.
:C	n:C is similar to c, only error messages are not printed.
d	D deletes the one character to the right of the pointer. +nD deletes n characters to the right of the pointer. -nD deletes n characters to the left of the pointer.
ea	EAq/string/ passes the string to the Multics active function command process; result is put in Q-register q.
ec	EC/request/ passes the string to the Multics command processor.
ei	EI/file/ reads the file into the buffer to the left of the current pointer.
:ei	:EI/file/ is similar to EI, only no errors are possible. Returns 0 if read fails; -1 if it succeeds.

NAME USE AND EXPLANATION

em EM/macro_name/
searches for the file macro_name.teco, first in the working directory, then the login directory, then the teco library. If found, it executes it as a macro.

eo EO/file_name/
writes out the entire buffer into the file specified.

+nEO/file_name/
writes out the next n lines.

(0 or -n)EO/file_name/
writes out the last n lines.

m,nEO/file_name/
writes out the (m+1)th through the nth characters.

eq EQ
returns to its caller.

g GQ
inserts the text contained in Q-register q into the buffer to the left of the pointer. If Q-register q contains a number, it is converted to a character string and inserted.

h H
is equivalent to 0,Z. It is the only symbol that has two values.

i I/string/
inserts the quoted string to the left of the pointer.

nI
n is the ASCII code for a letter that is inserted.

:i :Iq/string/
inserts the quoted string into Q-register q.

n:Iq
inserts the single character whose code is n into register q.

j nJ
moves the pointer to the right of the nth character in the buffer. If n is omitted, 0 is assumed.

:j n:j
is similar to j, only no errors.

NAME USE AND EXPLANATION

k K
deletes the rest of the current line from the buffer.

+nK
deletes the next n lines from the buffer.

(0 or -n)K
deletes the last n lines from the buffer.

m,nK
deletes the (m+1)th through the nth characters from the buffer.

l L
moves the pointer to the beginning of the next line.

+nL
moves the pointer to the beginning of the next nth line.

(0 or -n)L moves the pointer to the beginning of the last nth line.

:l :L
moves the pointer to the end of the current line.

+n:L
moves the pointer to the end of the next (n-1)th line.

(0 or -n):L
moves the pointer to the end of the lst (n+1)th line.

m m,nMq/string1//string2/.../stringN/
starts executing the text in Q-register q as a macro. m and n are numeric arguments to the first request in the macro. string1 through stringN are string arguments to the macro that can be retrieved with the :X request. EM also takes all these arguments.

:m m,n:M/string1//string2/.../stringN/
is similar to m only when control returns from Q-register q, macro containing :m request returns as well.

n N/string/
searches from the current pointer to the end of the buffer for the regular expression "string".

:n :N/string/
is similar to N/string/ except that :N returns a value. It returns 0 if the string is not found; -1 if it is.

NAME USE AND EXPLANATION

o O/label/
transfers control to just after label in the current macro, its caller, etc., or the request string.

P Pq
appends texts to Q-register q.

q Qq
the value of this request is the value of Q-register q if it is a numeric Q-register or the number of characters in Q-register q if it contains text. This request can also replace any quoted string if Q-register q contains text. The contents of the Q-register are used as the quoted string.

r R
moves the pointer one character to the left.

nR
moves the pointer n characters to the left.

:r R
is similar to R only no errors are possible.

s S/string/
searches from the current pointer to the end of the buffer for "string"; if found, it moves the pointer to the right of the string.

+nS/string/
searches for n occurrences of the string. Moves the pointer to the right of the nth occurrence.

-nS/string/
searches for n occurrences of "string" from the current pointer to the beginning of the file. If found, it moves the pointer to the left of the nth occurrence.

+m,+nS/string/
only searches from the current pointer to the beginning of the next mth line.

(0 or -m),-nS/string/
only searches from the current pointer to the beginning of the last mth line.

NAME USE AND EXPLANATION

:s :S/string/
takes arguments in all the ways S does, except that if S does not find the string, it prints out an error message and returns to teco request level. :S does not. Instead, :S has the value −1 if the search succeeds and 0 if the search fails.

t T
prints out the rest of the current line of the terminal.

+nT
prints out the buffer from the current pointer to the beginning of the next nth line.

(0 or −n)T
prints the buffer from the beginning of the last nth line to the current pointer.

m,nT
prints the (m+1)th through the nth characters of the buffer.

:t :T/string/
prints the quoted string on the terminal.

u Uq
sets Q−register q to a very large positive number.

nUq
sets Q−register q to n.

m,nUq
sets Q−register q to n and returns m as its value. This can be used inside a macro to get the numeric arguments to the macro.

vw VW
when this request is executed, one character is read from the terminal. The ASCII code for the character read is the value of the VW request.

:vw :VWq
reads in an entire line from the terminal and puts it into Q−register q. The newline character is the last character in the register.

w W
is used for throwing away unwanted numeric arguments.

x Xq
loads the rest of the current line into Q−register q.

NAME	USE AND EXPLANATION
	+nXq loads Q-register q with everything from the current pointer to the beginning of the next nth line. (0 or −n)Xq loads Q-register q with everything from the beginning of the last nth line to the current pointer. m,nXq loads Q-register q with everything from the (m+1) character to the nth character.
:x	:Xq loads Q-register q with the next string argument to the macro we are executing in.
z	Z is the total number of characters in the buffer. ZJ moves the pointer to the right of the last character in the buffer.
%	%q if Q-register q contains a numeric value, this request increments the register by 1. The value of the request is the new value of the Q-register.
$	$ throws away its arguments and does nothing.
newline	newline throws away its arguments and does nothing.
?	? turns tracing on.
??	?? turns tracing off.
\	\ is the decimal number immediately to the right of the pointer. It moves the pointer to just after the number. n\ inserts the decimal representation of n to the left of the pointer. m,n\ inserts the decimal representation of m to the left of the pointer. The representation is padded on the left to be at least n characters wide.

NAME USE AND EXPLANATION

:\ :\
is similar to \ except the values are octal and not decimal.

:[[q
pushes the contents of Q-register q onto the pushdown list.

]]q
pops the top element off the pushdown list and into Q-register q.

< <
marks the place in the request string that is transferred to by the > request. This loop can only be exited by the ; request.

:< :<
is similar to < except inhibits errors within the loop and causes > to return a value.

> >
transfers control to just after the last < request executed and decrements the loop count. If enough loops have occurred, this request does nothing. Nested loops are allowed.

; ;
if the last :s,n, *s, or :n request was unsuccessful, transfers to just after the next > and exits the present loop; otherwise, does nothing.

n;
if n is positive, transfers control to just after the next > request and exits the present loop; otherwise, does nothing.

:; :;
is similar to ; except that the sense of the test is inverted.

"C n"C
if n is the ASCII code for a letter, a digit, ., _, or $ does nothing. Otherwise, it transfers to just after the matching '.

"e m,n"E
if m=n, then "E does nothing; otherwise, transfers to just after the next '.

n"E
if n=0, then "E does nothing; otherwise, it transfers to just after the matching '.

NAME USE AND EXPLANATION

"g m,n"G
if m>n, then "G does nothing; otherwise, it transfers to just after the matching '.

n"G
if n>0, then "G does nothing; otherwise, it transfers to just after the matching '.

"l m,n"L
if m<n, then "L does nothing; otherwise, it transfers to just after the matching '.

n"L
if n<0, then "L does nothing; otherwise, it transfers to just after the matching '.

"n m,n"N
if m^=n, then "N does nothing; otherwise, it transfers to just after the matching '.

n"N
if n^=0, then "N does nothing; otherwise, it transfers to just after the matching '.

"m "m/string/
if characters immediately to the right of the pointer are equal to string, "m does nothing; otherwise; it transfers to just after the matching '.

:"m :"m/string/
if the characters immediately to the right of the pointer are not equal to string, "m does nothing; otherwise, it transfers to just after the matching '.

, ,
marks the location a " request transfers to. If executed as a request, it does nothing.

:' :'
marks the location a " request transfers to. If executed as a request, it transfers to just after the next '.

! !label!
is a label; it is ignored if it is executed.

. .
its request is the value of the current pointer.

NAME	USE AND EXPLANATION
=	= prints a newline. n= prints n in decimal followed by a newline. m,n= prints the value of m followed by a space, followed by the value of n, followed by a newline. The values are printed in decimal.
:=	m,n:= is similar to = except that the values are printed in octal.
F<	F; used to define throw-catch loops.

Name: teco_error

USAGE

```
declare teco_error entry (char(*));

call teco_error (name);
```

FUNCTION

prints the long form of a teco error message given the short term.

ARGUMENTS

name
is the short form of a teco error message. (Input)

Name: teco_ssd

SYNTAX AS A COMMAND

```
teco_ssd path
```

FUNCTION

specifies a directory for teco to search when trying to find a teco macro to execute.

ARGUMENTS

path
is the absolute pathname of a directory, instead of your home directory, to be searched by teco_get_macro_.

Name: terminal_output, to

SYNTAX AS A COMMAND

```
to {-control_arg}
```

FUNCTION

directs I/O output switches to your terminal. The effects of this command can be stacked.

CONTROL ARGUMENTS

-source_switch STR, -ssw STR
specifies the name of an I/O switch to be redirected. (Default: user_output)

NOTES

Each command invocation of terminal_output stacks up another attachment for each of the specified switches.

See the file_output, revert_output, and syn_output commands.

Name: terminate, tm

SYNTAX AS A COMMAND

```
tm paths
```

FUNCTION

removes a segment or multisegment file (MSF) from your address space and resets links to the terminated segment. It is commonly used prior to initiating a different version of a program.

ARGUMENTS

paths
: are pathnames of files to be terminated.

CONTROL ARGUMENTS

-brief, -bf
: suppresses the error message printed when a file to be terminated is not known (initiated).

-long, -lg
: does not suppress the above error message. (Default)

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

NOTES

Be careful not to unintentionally terminate a segment of the command language interpreter or another critical piece of the environment. Fatal process errors usually result from such an action.

You can't use the star convention.

Terminating a MSF terminates only component 0 of the MSF unless it is an object MSF, in which case all components are terminated.

Name: terminate_refname, tmr

SYNTAX AS A COMMAND

```
tmr ref_names
```

FUNCTION

removes a segment or multisegment file (MSF) from your address space and resets links to the terminated segment. It is commonly used prior to initiating a different version of a program.

ARGUMENTS

ref_names
are the reference names of segments or MSFs to be terminated.

CONTROL ARGUMENTS

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

NOTES

This command allows termination by reference name rather than by pathname. The segment or MSF itself is terminated, not merely the reference name specified.

Be careful not to unintentionally terminate a segment of the command language interpreter or another critical piece of the environment. Fatal process errors usually result from such an action.

You can't use the star convention.

If the reference name you specified is on a component of an object MSF, all components of the MSF are terminated and links to them unsnapped.

Name: terminate_segno, tms

SYNTAX AS A COMMAND

```
tms segnos
```

FUNCTION

removes a segment number(s) from your address space and resets links to the terminated segment. It is commonly used prior to initiating a different version of a program.

ARGUMENTS

segnos
are segment numbers (in octal) to be terminated.

CONTROL ARGUMENTS

-brief, -bf
suppresses the error message printed when you give an invalid segno.

-long, -lg
does not suppress the above error message. (Default)

NOTES

This command allows termination by segment number rather than by pathname.

Be careful not to unintentionally terminate a segment of the command language interpreter or another critical piece of the environment. Fatal process errors usually result from such an action.

You can't use the star convention.

If the segment number given identifies a component of an object MSF, all components of the MSF are terminated and links to them unsnapped.

terminate_single_refname

Name: terminate_single_refname, tmsr

SYNTAX AS A COMMAND

```
tmsr ref_names
```

FUNCTION

removes a file or a single reference name from your address space and resets links to the terminated segment or multisegment file (MSF). It is commonly used prior to initiating a different version of a program.

ARGUMENTS

ref_names
are the reference names of segments or MSFs to be terminated.

CONTROL ARGUMENTS

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

NOTES

This command terminates a single reference name. Unless the specified reference name is the only one by which the segment or MSF is known, the file itself is not terminated.

Be careful not to unintentionally terminate a segment of the command language interpreter or another critical piece of the environment. Fatal process errors usually result from such an action.

You can't use the star convention.

If the reference name is on a component of an object MSF, links to any of the components are unsnapped; if it is the only name on the MSF, all components are terminated.

Name: test_archive

SYNTAX AS A COMMAND

```
test_archive paths
```

FUNCTION

checks an archive segment for archive format errors and other inconsistencies. It is a library maintenance tool that is run weekly to check all archive segments in the online libraries.

ARGUMENTS

paths
: are the pathnames of the archive segments in question. The archive suffix is assumed.

Name: time

SYNTAX AS A COMMAND

```
time {time_string} {-control-arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[time {time_string} {-control-arg}]
```

FUNCTION

returns a five-character time of day of the form "HH:MM", e.g., "16:15". The format string to produce this is "^Hd:^MH".

ARGUMENTS

time_string
: indicates the time about which information is desired. If you supply no time_string, the current time is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-zone STR
: STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default zone. Use the display_time_info command to display a list of all acceptable zone values.

Due to exec_coms, etc., that have been built around the expected date_time format, this command does not honor the process date_time format (set by set_time_default). You are encouraged to use "clock time" in place of time to get the proper default handling.

Name: times

SYNTAX AS A COMMAND

```
times {num_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[times {num_args}]
```

FUNCTION

returns the product of the num_args. If you give no num_args, 1 (the multiplicative identity) is returned.

EXAMPLES

```
! string [times 6 7.3]
  43.8
```

This page intentionally left blank.

Name: total_output_requests, tor

SYNTAX AS A COMMAND

```
tor {request_types} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[tor {request_type}]
```

FUNCTION

prints the total number of requests in one or more I/O daemon queues.

ARGUMENTS

request_type
identifies the request type(s) for which totals are to be listed. The default is to list totals in the queues of the default printer request type used by enter_output_request -print (as displayed by print_request_types).

CONTROL ARGUMENTS

you can't give them when you invoke tor as an active function.

-all, -a
lists totals for all I/O daemon request type queues.

-brief, -bf
omits request types that are empty.

-inhibit_error, -ihe
suppresses error messages for request type queues to which you don't have access. Totals for such queues are printed as *****.

-long, -lg
includes request types that are empty. (Default)

ACCESS REQUIRED

You need status extended access to the queue segments for the request type.

NOTES

Each request type can have from one through four queues. The totals by queue are reflected in from one through four numbers listed for each request type. If an error occurs while accessing a queue, an asterisk is printed for the total in that queue.

EXAMPLES

The command line

```
! tor -all -brief
```

requesting a summary of all queues, but excluding any empty request types, might print the following:

```
printer:                0     0     3     2
fllx8:                  0     2
hdsa_prt:             *****
          Incorrect access on entry.  hdsa_prt queue 1
punch:                  2
```

indicating the current number of output requests by queue for the first, second, and fourth request types and an access error for the third type. As an active function, tor might be embedded in an admin.ec to test for cards to punch. The following exec_com fragment demonstrates this usage:

```
&if &[ngreater [plus [tor punch]] 0]
&then &goto START_PUNCHING_CARDS
&print No cards to punch.
&quit
```

Name: trace

SYNTAX AS A COMMAND

```
trace {entrypoints} {-control_args}
```

FUNCTION

debugs systems of programs, stops execution at any call or return, and measures the resources spent in each program. It helps you isolate which programs might be malfunctioning by tracing calls, returns, argument lists, and signals. A metering mode helps expose inefficient programs by measuring and accumulating real time, virtual CPU time, and page faults spent in each program. The trace facility works with programs written in PL/I, FORTRAN, COBOL, Pascal, and ALM. (See also the trace_meters and watch commands.)

ARGUMENTS

entrypoints
represent program entrypoints. You must separate multiple entrypoints by spaces (see "Notes on the Syntax of Entrypoints"). If you don't use any control arguments, the entrypoints are added to the trace table with the default trace parameters currently in effect.

CONTROL ARGUMENTS
fall into three groups: those that tell how to trace the specified entrypoints (trace parameters), those that tell trace how to behave in general (global parameters), and those that specify actions (such as information display).

Trace Parameters

-arguments in|out|on|off, -ag in|out|on|off
lists the arguments when the specified entrypoints are traced at call (in), return (out), both (on), or neither (off). Tracing arguments is useful for debugging, but greatly increases the volume of trace information. (Default: initially off)

-call COMMAND-LINE
executes a command line whenever the specified entrypoints are traced at both call and return, but not at unwind. When COMMAND-LINE contains spaces, quote it. It can't exceed 256 characters. Trace is temporarily disabled while COMMAND-LINE is being executed. Specifying -no_call or -call "" turns off this function. (Default: initially -no_call)

-every N, -ev N
prevents the specified entrypoints from being traced unless their call count is a multiple of N; for example, -every 10 permits every tenth call to be traced. Specifying -no_every or -every 0 eliminates this constraint. (Default: initially -no_every)

-first N, -ft N
prevents the specified entrypoints from being traced when their call count is less than N; for example, if an entrypoint's -first is 1000, the first 999 calls are not traced. Specifying -no_first or -first 0 eliminates this constraint. (Default: initially -no_first)

-high N
prevents the specified entrypoints from being traced when their recursion level is greater than N; for instance, if an entrypoint's -high is 1, no recursive invocations are traced. Specifying -no_high or -high 0 eliminates this constraint. (Default: initially -no_high)

-last N, -lt N
prevents the specified entrypoints from being traced when their call count is greater than N; for example, -last 1 traces only the first call of each entrypoint. Specifying -no_last or -last 0 eliminates this constraint. (Default: initially -no_last)

-low N
prevents the specified entrypoints from being traced when their recursion level is less than N; for instance, -low 2 permits only recursive invocations to be traced. Specifying -no_low or -low 0 eliminates this constraint. (Default: initially -no_low)

-new_high on|off
permits the specified entrypoints to be traced only when their recursion level has reached a new high. It is useful for tracing runaway recursion. (Default: initially off)

-no_call
turns off the -call function for the specified entrypoints. (Default, initially)

-no_every, -nev
eliminates the -every constraint for the specified entrypoints. (Default, initially)

-no_first, -nft
eliminates the -first constraint for the specified entrypoints. (Default, initially)

-no_high
eliminates the -high constraint for the specified entrypoints. (Default, initially)

-no_last, -nlt
eliminates the -last constraint for the specified entrypoints. (Default, initially)

-no_low
eliminates the -low constraint for the specified entrypoints. (Default, initially)

-no_stop_every, -nspev
eliminates the -stop_every constraint for the specified entrypoints. (Default, initially)

-no_stop_low, -nsplow
eliminates the -stop_low constraint for the specified entrypoints. (Default, initially)

-stop in|out|on|off, -sp in|out|on|off
stops execution of the specified entrypoints when they are called (in), when they return (out), both (on), or neither (off). (Default: initially off)

-stop_every N, -spev N
prevents the specified entrypoints from being stopped unless their call count is a multiple of N; for instance, -stop_every 10 permits every tenth call to be stopped. Execution can be stopped at call, return, or both by specifying -stop in, -stop out, or -stop on. Specifying -no_stop_every or -stop_every 0 eliminates this constraint. (Default: initially -no_stop_every)

-stop_low N, -splow N
prevents the specified entrypoints from being stopped unless their recursion level is N or greater; for example, -stop_low 2 only stops recursive invocations. Execution can be stopped at call, return, or both by specifying -stop in, -stop out, or -stop on. Specifying -no_stop_low or -stop_low 0 eliminates this constraint. (Default: initially -no_stop_low)

-trace in|out|on|off
traces the specified entrypoints when they are called (in), when they return (out), both (on), or neither (off). (Default: initially on)

Global Parameters

-alm on|off
sets the -alm global parameter on or off. ALM (Assembly Language for Multics) programs sometimes use nonstandard call-return protocols that malfunction when traced, or make trace malfunction. This parameter controls how ALM entrypoints are handled. When -alm is on, they are handled like ordinary entrypoints. When -alm is off, they are ignored by the trace facility, even if they are in the trace table. Initially -alm is off.

-automatic on|off, -auto on|off
sets the -automatic global parameter on or off. This parameter provides an easy way to trace everything. It automatically adds entrypoints to the trace table when they are first called. Their trace parameters are set to the current defaults. Specifying -automatic on implies -signals on, and specifying -automatic off implies -signals off. If you want automatic mode without signal tracing, specify -automatic on -signals off. Initially -automatic is off.

-brief, -bf
sets the -brief global parameter, which abbreviates trace messages by excluding the time of the trace event, the caller of the entrypoint being traced, and the meters when the entrypoint returns. This reduces wraparound when the trace is displayed on an 80-column terminal instead of a line printer. Initially -brief is set.

-buffer on|off, -buf on|off
sets the -buffer global parameter on or off. Specifying -buffer on redirects the trace information to a circular buffer in the process directory. The buffer contains 8192 entries. You can display it with -print_buffer. Buffering is much more efficient than regular tracing, but buffer entries do not have room for argument lists. Initially -buffer is off.

-disable, -disa
disables trace; for instance, to stop the trace messages, to "freeze" the meters, or to turn trace off when it is not being used. Trace is enabled when you use the trace command for the first time in a process.

-enable, -ena
reverses the effect of -disable.

-long, -lg
sets the -long global parameter for full trace messages, which include clock time and meters. This setting is appropriate for a 132-column output device. Initially -brief is set.

-loud
sets the -loud global parameter, which tells the trace and watch commands to summarize the effect of each command line and warn when trace is disabled. Initially -loud is set.

-meter on|off, -mt on|off
sets the -meter global parameter on or off. Trace always meters, even if you specify -meter off. Specifying -meter on tells trace to concentrate on metering and skip all trace, stop, and watch checks. The trace_meters command displays and resets the meters. Initially -meter is off.

-no_output_file, -nof
resets the -output_switch global parameter to its initial value, user_output.

-no_output_switch, -nosw
resets the -output_switch global parameter to its initial value, user_output.

-no_stop_proc, -nspp
resets the -stop_proc global parameter to its initial value, the command processor (cu_$cl).

-output_file PATH, -of PATH
sets the -output_switch global parameter so that the trace is written to the file specified by PATH. The ".trace" suffix is added to PATH if you don't give it; the file is truncated if it already exists, or created if it does not. If the trace was already being written to a file, that file is closed after the new one is opened. Specifying -no_output_file or -output_file "" resets -output_switch to its initial value, user_output.

-output_switch SWITCH, -osw SWITCH
sets the -output_switch global parameter to SWITCH. This parameter determines the I/O switch through which trace, watch, and stop messages are written. If SWITCH is not attached to the same device as error_output, watch and stop messages are also written to error_output. The switch must be open and prepared to receive stream output. Specifying -no_output_switch or -output_switch "" resets -output_switch to its initial value, user_output.

-quiet
sets the -quiet global parameter, which tells the trace and watch commands not to summarize the effect of each command line or warn when trace is disabled. Initially -loud is set.

-signals on|off, -sig on|off
sets the -signals global parameter on or off. It controls whether signaled conditions are traced. Initially -signals is off.

-stop_proc ENTRYNAME, -spp ENTRYNAME
sets the -stop_proc global parameter to ENTRYNAME, where ENTRYNAME can be any string acceptable to the cv_entry_ subroutine. This parameter is the entrypoint that trace calls to stop execution. It is called with trace temporarily disabled. When the -stop_proc returns, trace is re-enabled and execution resumes. If the -stop_proc is the command processor, the start command makes it return. Specifying -no_stop_proc or -stop_proc "" reset -stop_proc to its initial value, the command processor (cu_$cl).

Actions

-add
adds the specified entrypoints to the trace table. If any of them are already in the trace table, their trace parameters are updated. In either case, their trace parameters assume the current default values amended by any control arguments that specify trace parameters.

-off
turns the specified entrypoints off. They remain in the trace table, but tracing, watching, stopping, and metering are disabled. When an entrypoint is turned off, calls to it continue to be counted.

-on
turns the specified entrypoints on.

-parameters, -pm
displays the default trace parameters and the global parameters.

-print_buffer N, -prbuf N
displays the last N events in the circular trace buffer (see -buffer). If N is greater than 8191, the entire buffer is displayed. The amount of information displayed depends on whether -brief or -long is in effect.

-remove, -rm
removes the specified entrypoints from the trace table.

-set_defaults, -sdft
makes the trace parameters specified in the command line be the defaults.

-status, -st
displays the counters and trace parameters of the specified entrypoints in the trace table. Use the trace_meters command to display the meters.

NOTES

If you specify entrypoints and don't specify -add, -remove, -on, -off, or -status, -add is assumed.

Execution is stopped either before an entrypoint has pushed its stack frame or after it has popped its stack frame; therefore, its stack frame cannot be inspected. When execution is stopped, trace is temporarily disabled until execution is resumed.

The recursion level of an entrypoint is actually the number of invocations that have not yet returned, not the number of recursive invocations, which would be one less since the first invocation is not a recursive invocation.

The order of entrypoints in the trace table is determined by their segment numbers and offsets. The table is ordered first by ascending segment number and then by ascending offset. This permits rapid lookup by binary search.

Tracing with -automatic on and -meter on typically doubles execution time, but the overhead is excluded from the meters. Tracing with -automatic on and -buffer on typically triples execution time. Tracing with -automatic on and -arguments on into an output file requires about 20 milliseconds to trace each event and typically increases execution time by a factor of 50. It also uses up quota fast.

The trace facility can watch virtual memory locations for changes in their contents. See the documentation of the watch command for more information. The trace facility has the following restrictions:

1. Gates cannot be traced. Trace must be separately invoked in each ring.

2. Some programs use nonstandard call-return protocols that malfunction when traced or make trace malfunction. Programs that look at their caller's stack frame malfunction because trace inserts its own stack frame ahead of every entrypoint that it traces or meters (to detect the return or unwind).

   The trace facility has a list of some entrypoints that cannot be traced. These entrypoints may be added to the trace table but they are effectively turned off (see -off). The list includes: cobol_control_*, cobol_rts_*, condition_$*, cu_$*, formline_*, fortran_io_*, link_trap_caller_*, lisp_*, nonlocal_goto_$*, pascal_area_management_$*, pascal_errors_$*, pascal_io_$*, pascal_time$*, probe*, ssu_$standalone_invocation, ssu_invocation_$create_standalone, unwind_stack_, and unwinder_.

3. The trace table can hold up to 10,000 entrypoints.

4. An ALM entrypoint can only be traced if it invokes the ALM entry operator. The "entry" and "get_lp" pseudo-ops do this. The first instruction of an ALM entrypoint must be a transfer to the ALM entry operator, otherwise it is effectively turned off (see -off). Entrypoints that do not have segdefs may be added in automatic mode with names like bound_foo_$1234.

NOTES ON THE SYNTAX OF ENTRYPOINTS

Trace uses an entryname syntax that is capable of referring to individual entrypoints or sets of entrypoints. An entryname can have the following forms:

pathname|entryname
designates an entrypoint by the absolute or relative pathname of its segment and by its symbolic offset within that segment (e.g., >sss>bound_fscom2_|copy).

pathname
same as pathname|[entry pathname]. If pathname contains no "<" or ">" characters, it is interpreted as reference_name.

pathname|*
designates some or all entrypoints in a segment. If the segment is not bound, it designates all entrypoints. If the segment is bound and the entry portion of the pathname is the name of the bound object, it designates all entrypoints (e.g., >sss>bound_fscom2_|*). If the segment is bound and the entry portion of the pathname is the name of a component of the bound object, it designates all entrypoints in the component (e.g., >sss>copy|*).

reference_name$entryname
designates an entrypoint by the reference name, absolute pathname, or relative pathname of its segment and by its symbolic offset within that segment (e.g., copy$cp). If reference_name contains "<" or ">" characters, it is interpreted as a pathname, otherwise it is interpreted as a reference name and is located via the search rules for executable segments.

reference_name
same as reference_name$reference_name.

reference_name$*
similar to pathname|*, except that the segment is designated by its reference name.

*
designates all entrypoints in the trace table. If you specify *, you cannot specify any other entrypoints.

NOTES ON ERROR MESSAGES

The trace facility frequently checks the consistency of the trace table. When an inconsistency is detected, it halts with a message of the form:

```
Error: Linkage error by trace_catch_|141
(>sss>bound_trace_)
referencing trace_error_halt_|table_index_oob
Segment not found.
```

or

```
Error:  error condition by trace_tables_$parameters_ptr|6247
        (>sss>bound_trace_)
```

Malfunctioning programs can cause this kind of error by accidentally writing on the trace table. You can cause this kind of error by modifying the trace table while an invocation of trace is suspended. For example, if you QUIT while tracing, and then do "trace -remove *; start", an error of this form is likely. You can, however, do QUIT, "trace -disable; start", or QUIT, "trace -remove *; release".

EXAMPLES

The command sequence

```
! trace
```

tells the number of entrypoints in the trace table and displays the global parameters, the default parameters, and a list of the action control arguments.

The command sequence

```
! trace -meter on -automatic on
! some script
! trace -disable
! trace_meters -output_file PATH.tmt
```

puts a report in the file PATH.tmt that shows real time, CPU time, and page fault meters for all entrypoints that were invoked during the script. Some inaccuracy results from linkage faults and other initializations that occur the first time the script is run in a process. This error can be eliminated by running the script twice with "trace_meters -reset" in between. The command sequence

```
! trace -buffer on -auto on
! some script that blows up
! trace -disable
! fo PATH.trace; trace -print_buffer 9999; ro
```

puts the last 8192 trace events in the file PATH.trace. It shows all calls, returns, unwinds, and signals leading up to the failure.

The command sequence

```
!  trace -of PATH.trace -auto on -arguments on -set_defaults
!  some script
!  trace -nof -auto off -arguments off -set_defaults -remove *
```

puts a complete call-return history in the file PATH.trace. It shows all calls and returns with arguments and all conditions that were signaled. This is a good way to diagnose fatal process errors (use the adjust_bit_count command on the file before examining it).

The command line

```
!  trace -parameters -status *
```

displays the default trace parameters and the global parameters. It also displays a list of all the entrypoints in the trace table with their counts and parameters that are different from the defaults.

SUMMARY OF TRACE PARAMETERS

```
-trace in|out|on|off                Trace calls, returns, both, neither. (on)
                                    Don't trace unless...
-every N                 (-ev)      call count is a multiple of N. (-no_every)
-first N                 (-ft)      call count is N or greater. (-no_first)
-last N                  (-lt)      call count is N or less. (-no_last)
-low N                              recursion level is N or greater. (-no_low)
-high N                             recursion level is N or less. (-no_high)
-new_high on|off                    recursion level is the highest yet. (off)

-stop in|out|on|off      (-sp)      Stop calls, returns, both, neither. (off)
                                    Don't stop unless...
-stop_every N            (-spev)    call count is a multiple of N. (-no_stop_every)
-stop_low N              (-splow)   recursion level is N or greater. (-no_stop_low)

-arguments in|out|on|off (-ag)  Args at call, return, both, neither. (off)
-call COMMAND-LINE              Call whenever entrypoint is traced. (-no_call)
```

SUMMARY OF GLOBAL PARAMETERS

```
-alm on|off                         How to handle ALM programs. (off)
-automatic on|off        (-auto)    Add entrypoints automatically. (off)
-buffer on|off           (-buf)     Send the trace to a circular buffer. (off)
-meter on|off            (-mt)      Just meter, don't trace or stop. (off)
-signals on|off          (-sig)     Trace signaled conditions. (off)
-enable/-disable         (-ena/-disa) Enable/disable trace. (-enable)
-long/-brief             (-lg/-bf)  Use full/abbreviated trace messages. (-bf)
```

```
-loud/-quiet                    Summarize effect of each command. (-loud)
-output_file PATH    (-of)      Write trace to PATH.trace. (-no_output_file)
-output_switch SWITCH (-osw)    Write trace through SWITCH. (user_output)
-stop_proc ENTRYNAME (-spp)     Call ENTRYNAME to stop execution. (cu_$cl)
```

SUMMARY OF ACTIONS

```
-add                            Add entrypoints to the trace table.
-remove              (-rm)      Remove entrypoints from the trace table.
-on                             Turn on entrypoints in the trace table.
-off                            Turn off entrypoints in the trace table.
-status              (-st)      Display entrypoints in the trace table.
-set_defaults        (-sdft)    Make specified trace parms the defaults.
-parameters          (-pm)      Display the trace and global parameters.
-print_buffer N      (-prbuf)   Display the last N events in the buffer.
```

Name: trace_meters, tmt

SYNTAX AS A COMMAND

```
tmt {-control_args}
```

FUNCTION

formats and displays the cpu time and page fault meters of entry points that are traced with "-meter on" (see the trace command).

CONTROL ARGUMENTS

-global_percent N, -gpct N, -g% N
: reports only entry points whose global cpu time or global page faults exceed N% of the total. N must be a whole number.

-output_file path, -of path
: causes output to be directed to the file specified by path. The file is overwritten if it already exists, or created if it does not. The ".tmt" suffix is added to path if it is not given.

-percent N, -pct N, -% N
: reports only entry points whose local cpu time or local page faults exceed N% of the total. N must be a whole number.

-report_reset, -rr
: displays the report and zeroes the meters.

-reset, -rs
set the meters of every entry point in the trace table to zero and does not display the report.

NOTES

If no arguments are supplied, the report is displayed and the meters are not reset.

Entry points that have not been called since the last reset are not reported.

Some programs are on a special list of programs that can never be traced (see the trace command). Some entry points can not be traced because they are written in alm and are added to the trace table when "-alm off" is in effect. The tracing of some entry points may have been turned off by the "trace -off" command. Entry points that are not metered either because they can not be traced or because they have been turned off are, nevertheless, counted and are reported with empty meter columns.

If the local virtual cpu time reported for a procedure that has been called only a few times is unbelievably large, consider the possibility that it snapped some dynamic links during its run. This metering error can be eliminated by reseting the meters and repeating the run within the same process. It can be completely avoided by making a trial run before metering.

NOTES ON COLUMN HEADINGS

The column headings of the report are interpreted as follows:

GREAL	global real time in seconds
GVCPU	global virtual cpu time in seconds
GPF	global page faults
LREAL	local real time in seconds
LVCPU	local virtual cpu time in seconds
LPF	local page faults
LVCPU/CALL	local virtual cpu time in seconds per call
LVCPU%	local vcpu time as a percentage of total vcpu time
CALLS	number of calls

Global means the resources used by an entry point and everything it calls. Local means the resources used by an entry point less the resources used by all traced entry points that it calls.

Name: trace_stack, ts

SYNTAX AS A COMMAND

```
ts {-control_args}
```

FUNCTION

prints a detailed explanation of the current process stack history in reverse order (most recent frame first). For each stack frame, all available information about the procedure that established the frame (including, if possible, the source statement last executed), the arguments to that (the owning) procedure, and the condition handlers established in the frame are printed.

CONTROL ARGUMENTS

-brief, -bf
: suppresses listing of source lines, arguments, and handlers. It is incompatible with -long.

-depth N, -dh N
: dumps only N frames.

-long, -lg
: prints octal dump of each stack frame.

-stack_ptr PTR, -sp PTR
: starts tracing from stack frame at PTR, where PTR is a virtual pointer acceptable to cv_ptr_. PTR points to the stack frame at which tracing is to begin.

NOTES

This command is most useful after a fault or other error condition. If you invoke trace_stack after such an error, the machine registers at the time of the fault are also printed, as well as an explanation of the fault. The source line in which it occurred can be given if you compile the object segment with the -table option.

For a discussion of the Multics stack frame, see the Programmer's Reference Manual.

NOTES ON OUTPUT FORMAT

When you invoke trace_stack with no -stack_ptr, it first searches backward through the stack for a stack frame containing saved machine conditions as the result of a signaled condition. If such a frame is found, tracing proceeds backward from that point; otherwise, tracing begins with the stack frame preceding trace_stack.

If a machine-conditions frame is found and you didn't supply -brief, trace_stack repeats the system error message describing the fault, source line, and faulting instruction and a listing of the machine registers at the time the error occurred.

The command then performs a backward trace of the stack, for N frames if you gave the -depth argument or else until the beginning of the stack is reached.

For each stack frame trace_stack prints the offset of the frame, the condition name if an error occurred in the frame, and the identification of the procedure that established the frame. If the procedure is a component of a bound segment, the bound segment name and the offset of the procedure within the bound segment are also printed.

Unless you supply -brief, trace_stack then locates and prints the source line associated with the last instruction executed in the procedure that owns the frame (i.e., either a call forward or a line that encountered an error). The source line can be printed only if the procedure has a symbol table and if the source for the procedure is available in your working directory. If the source line cannot be printed, trace_stack prints a comment explaining why.

Next, trace_stack prints the machine instruction last executed by the procedure that owns the current frame. If the machine instruction is a call to a PL/I operator, the command also prints the name of the operator. If the instruction is a procedure call, trace_stack suppresses the octal printout of the machine instruction and prints the name of the procedure being called.

Trace_stack then lists the arguments supplied to the procedure that owns the current frame and also lists any enabled condition, default, and cleanup handlers established in the frame.

If you select -long, the command then prints an octal dump of the stack frame, with eight words per line.

Name: transaction, txn

SYNTAX AS A COMMAND

```
txn key {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn key {-control_args}]
```

FUNCTION

enables you to define and execute atomic operations interactively. You can invoke the services of the transaction manager to begin, commit, abort, rollback, abandon, or kill a transaction. There is also a status request for displaying information about the current transaction. There is an execute request to wrap a given command line in a transaction. This command is part of the command level interface to Multics data management (DM) (see the Programmer's Reference Manual).

ARGUMENTS

key
designates the operation to be performed. See "List of Operations" below for a description of each operation, its command syntax line, and specific application.

CONTROL ARGUMENTS
can be one or more control arguments, depending on the particular operation.

LIST OF OPERATIONS

Each operation is described in the general format of a command/active function. Where appropriate, notes and examples are included for clarity.

Operation: abandon

SYNTAX AS A COMMAND

```
txn abandon
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn abandon]
```

FUNCTION

your process surrenders control of the transaction to the DM Daemon, which aborts it as part of its normal caretaker responsibilities. The active function returns true if the transaction is successfully abandoned, false otherwise.

NOTES

By abandoning a transaction, your process can start another transaction without waiting for the abort operation to conclude (your process is still charged for the abort). The data locked by the original transaction remains inaccessible, however, until the rollback is completed.

Operation: abort

SYNTAX AS A COMMAND

```
txn abort
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn abort]
```

FUNCTION

aborts the current transaction so that, in effect, it never existed. Any modifications to protected files caused by the aborted transaction are rolled back, and references to the transaction are removed from system tables. The active function returns true if the transaction is successfully aborted, false otherwise.

Operation: begin

SYNTAX AS A COMMAND

```
txn begin {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn begin {-control_args}]
```

FUNCTION

starts a transaction by reserving a slot in the transaction definition table (TDT) for your process, with a unique transaction identifier, date/time of the start, pathname of the before journal, and other information pertinent to the transaction (see the status operation). If your process already owns a transaction, an error occurs. The active function returns true if a transaction is started successfully, false otherwise.

CONTROL ARGUMENTS

-no_wait, -nwt
: causes an error if the data management system (DMS) is not currently invoked. (Default)

-wait N, -wt N
: if DMS is not currently invoked, wait N seconds before starting the transaction. An error occurs if DMS is still not up after the elapsed time.

-wait_indefinitely, -wti
: if DMS is not currently invoked, wait as long as necessary to start the transaction. The status of DMS is checked at 10-second intervals, and notification is given when command line execution begins.

NOTES

This operation is a tool for isolating and testing the transaction startup function. In a production environment the transaction execute command is the recommended method of starting transactions from command level because it builds in the atomicity: it begins the transaction, executes a command line, and then terminates the transaction, within the one request (see the execute operation).

EXAMPLES

The following example shows an absentee job intended not to run until a transaction can be started in absentee.

```
&if &[not [txn begin -wait 100]] &then &do
    ear &ec_path -time "+1 hour" -ag &f1
    &quit
&end
```

Operation: commit

SYNTAX AS A COMMAND

```
txn commit
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn commit]
```

FUNCTION

signals successful completion of the currently active transaction. Modifications made to protected files by this transaction are considered permanent. Any locks held by the transaction are released, making the data public again. The active function returns true if the commit operation is successful, false otherwise.

Operation: execute, e

SYNTAX AS A COMMAND

```
txn e {-control_args} {command_line}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn e {-control_args} {command_line}]
```

FUNCTION

starts a transaction, executes a command line, and, provided the command line is successfully executed, commits the transaction. Control arguments govern what action to take based on conditions encountered. The active function returns true if the execute operation is successful, false otherwise.

ARGUMENTS

command_line
specifies the command line to be executed as part of the transaction. Enclose it in quotes if it contains parentheses, brackets, or semicolons. If you omit it, the system prompts "Command line:".

CONTROL ARGUMENTS

-abandon_on CONDITION_LIST
abandons the transaction and results in a nonlocal exit of the command line if any of the listed conditions is encountered during command line execution. Separate the listed conditions by commas, with no intervening whitespace. The list can include any_other. The default action is as described under "Notes" below. This control argument is incompatible with -existing_transaction_allowed and -existing_transaction_required.

-abort_on CONDITION_LIST
aborts the transaction and results in a nonlocal exit of the command line if any of the listed conditions is encountered during command line execution. Separate the listed conditions by commas, with no intervening whitespace. The list can include any_other. The default action is as described under "Notes" below. This control argument is incompatible with -existing_transaction_allowed and -existing_transaction_required.

-command_level, -cl
places your process at the next command level, from which commands can be entered in the transaction. You can use the start or release command to exit this command level.

-existing_transaction_allowed, -eta
accepts the existing transaction (if one already exists in your process) as the origin of command line execution. No new transaction is begun. This control argument is incompatible with -retry_on and -suspend_on. (Default: to return an error if a transaction already exists)

-existing_transaction_required, -etr
requires that a transaction already exist in your process; returns an error if no transaction exists. This control argument is incompatible with -retry_on and -suspend_on. (Default: to return an error if a transaction already exists)

-no_action_on CONDITION_LIST
overrides any special action (e.g., -abandon_on, -retry_on) you previously specified in the command line for the listed conditions. The default action (see "Notes") is also overridden.

-no_existing_transaction_allowed, -neta
causes an error if a transaction already exists in your process. (Default)

-no_wait, -nwt
causes an error if DMS is not currently invoked. (Default)

-retry_on N CONDITION_LIST
executes the command line up to N times if any of the listed conditions is encountered during command line execution. If N is 0, the command line is not retried. Separate the listed conditions by commas, with no intervening whitespace. The list can include any_other. The default action is as described under "Notes" below.

-suspend_on CONDITION_LIST
suspends the transaction and goes to the next command level if any of the listed conditions is encountered during command line execution. Separate the listed conditions by commas, with no intervening whitespace. The list can include any_other. The default action is as described under "Notes" below.

-wait N, -wt N
if DMS is not currently invoked, waits N seconds before starting the transaction and executing the command line (you are notified when command line execution begins). An error condition is returned if DMS is still not up after the elapsed time. This operation is useful for absentee jobs submitted to perform operations within transactions.

-wait_indefinitely, -wti
if DMS is not currently invoked, waits as long as necessary to start the transaction and execute the command line. The status of DMS is checked at 10-second intervals, and notification is given when command line execution begins.

NOTES

If a transaction already exists in your process, the default action is -no_action_on any_other; otherwise the default action is -suspend_on any_other -abort_on cleanup.

A transaction begun by txn execute is committed unless the command line fails to execute properly, in which case the transaction is aborted.

A transaction severity code (displayable by the "severity transaction" command) denotes the status of the execute operation, as follows:

0 the operation was completed without errors and was not retried.
1 the operation was completed, but was retried one or more times.
2 the operation failed; the transaction was aborted or abandoned.
3 the operation failed; the transaction could not be aborted or abandoned.
4 the transaction could not be begun.

The active function returns true if the severity after execution is 0 or 1; false if it is 2, 3, or 4.

If a transaction is currently suspended in your process, the txn execute command gets an error and the active function returns false.

Operation: kill

SYNTAX AS A COMMAND

```
txn kill {ID}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn kill {ID}]
```

FUNCTION

expunges the current or specified transaction with no attempt to preserve consistency of any DM files that might have been modified by this transaction. Killing a transaction may destroy the consistency of any databases that the transaction is using; therefore use this operation when neither you nor the Daemon is able to complete the transaction. The active function returns true if the operation is executed successfully, false otherwise.

ARGUMENTS

ID
: is the unique identifier of the transaction to be killed (obtainable through txn status). (Default: the current transaction in your process)

ACCESS REQUIRED

You need re access to dm_daemon_gate_.

Operation: rollback

SYNTAX AS A COMMAND

```
txn rollback
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn rollback]
```

FUNCTION

rolls back the current transaction to its beginning (txn begin), undoing any changes to protected files caused by the transaction and releasing the locks held by it. The transaction is still considered active in your process. The active function returns true if the transaction was successfully rolled back, false otherwise.

Operation: status, st

SYNTAX AS A COMMAND

```
txn st {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[txn st {-control_args}]
```

FUNCTION

displays information about the current transaction, selected transactions, or all transactions, depending on the nature of the request and your access permissions. The active function takes only one information control argument.

CONTROL ARGUMENTS FOR SELECTING TRANSACTIONS

If you supply no control arguments, or lack the proper access, only information pertaining to your current transaction is displayed.

-abandoned
displays the requested information about all TDT entries marked as abandoned.

-all, -a
displays the requested information about all TDT entries.

-dead
displays the requested information about all TDT entries belonging to dead processes.

-transaction_id ID, -tid ID, -id ID
displays the requested information about the transaction with unique identifier ID, where ID is a decimal integer. Transaction identifiers are assigned at txn begin time and can be viewed by the txn status command.

-transaction_index N, -tix N, -index N
displays the requested information about entry number N in the TDT. TDT entry indexes are of interest mainly to data management maintainers and can be viewed by the txn status command.

CONTROL ARGUMENTS FOR SELECTING INFORMATION

If you give none of the following control arguments, all information is displayed for each TDT entry selected. You can specify only one control argument for the active function.

-before_journal_path, -bj_path
returns the pathname of the before journal used by the current transaction.

-date_time_begun, -dtbg, -begun
returns the date and starting time of each transaction.

-error, error_info
returns a description of the latest error, if any, to have occurred while processing each transaction.

-owner
identifies the owner (User_id.Project_id) of each TDT entry.

-process_id, -pid
returns the octal process_id of the owner of each TDT entry.

-rollback_count, -rbc
returns the number of times each transaction has been rolled back.

-state
indicates the state of each transaction, which must be one of the following:

no transaction (e.g., the process might have owned a transaction, which has been taken over by the DM Daemon)

in progress

{Error - } OPERATION, calling PROGRAM_NAME, which gives the operation currently in progress, such as abort or commit, and the entry point being called; and is followed by an error message if appropriate.

-switches, -switch, -sw
lists those transactions that are either abandoned, killed, or suspended or whose owner processes are dead.

-total, -tt
prints totals information for the TDT, including:

number of slots available (not yet reserved by processes)

number in use (i.e., reserved by processes at first invocation of DMS)

number of entries owned by dead processes (of the number in use)

number of abandoned entries (of the number in use)

number of entries occupied by transactions (i.e., slots reserved by processes that have started transactions)

number of transactions in error.

-transaction_id, -tid, -id
supplies the unique identifier of each transaction. Use of -transaction_id with a specific transaction ID returns information about that transaction.

-transaction_index, -tix, -index
returns the index of entries in the TDT. This index is mainly of interest to data management maintainers. Use of -transaction_index with a specific integer N returns information about a given TDT entry number.

NOTES

You can't use the following control arguments with the active function: -abandoned, -all, -dead, and -total.

You need re access to dm_admin_gate_ to view the status of any other user's transactions.

EXAMPLES

The command

```
! txn status -tid
  9
```

asks for the unique identifier of the transaction currently owned by the requesting user process.

The command

```
txn status -a -owner -dtbg -tid
TDT size: 6 entries
In use: 4
Dead processes: 1
Abandoned entries: 0
Transactions: 3
Error transactions: 0

Transaction id: 4
Owner: Merrill.Multdev
Begun at: 02/12/84 0837.11 est wed

Owner: Lynch.Multdev
No Transaction.

Transaction id: 9
Owner: Pierce.Debug
Begun at: 02/12/84 0846.3 est wed
```

```
Transaction id: 12
Owner: Fenner.Support
Begun at: 02/12/84 0901.5 est wed
```

requests that each transaction in the TDT be identified as to its unique identifier, owner, and date/time of origin.

The command

```
txn status
Transaction id: 4
TDT index: 2
Process id: 467265315627
Owner: Smith.Applications
Begun at: 02/12/84 0846.3 est wed
State: In progress
Error: none
Checkpoint id: 0
Rollback count: 0
Before journal path: >site>dm>system_low>system_default.bj
Switches: none
```

requests all available information on the transaction owned by the requesting user process.

The command

```
txn status -tix 1 -pid -state -error -switches
Process id: 625731253642 (dead)
State: Error - Abort, calling bjm_$write_aborted_mark
Error: The before journal is full.
Switches: ABANDONED, DEAD_PROCESS
```

requests the process id, state, error condition, and switch settings for the specified transaction index entry.

Name: translate

SYNTAX AS A COMMAND

```
translate STRA STRB {STRC}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[translate STRA STRB {STRC}]
```

FUNCTION

returns translation in which all the characters of a string STRA that appear in string STRC are translated to the corresponding characters in string STRB. If STRC is omitted, a default string containing all possible 9-bit bit patterns is used, as returned by collate9.

EXAMPLES

```
! string [translate abcdefgh BDFH bdfh]
  aBcDeFgH

! string [translate "My work" KLMNOPQRSTUVWXYZ klmnortvwxyz]
  MY WORK
```

Name: trunc

SYNTAX AS A COMMAND

```
trunc num
```

SYNTAX AS AN ACTIVE FUNCTION

```
[trunc num]
```

FUNCTION

returns the largest decimal integer whose absolute value is less than or equal to the absolute value of num.

EXAMPLES

```
! string [trunc 7.6]
  7

! string [trunc -7.6]
  -7
```

Name: truncate, tc

SYNTAX AS A COMMAND

```
tc {-control_arg} path {length}

tc segno {length}
```

FUNCTION

truncates a segment to an optionally specified length and resets the bit count accordingly, setting the bit count author to be the user who invoked the command.

ARGUMENTS

path
is the pathname of a segment. You can't use the star convention.

length
is an octal integer indicating the length of the segment in words after truncation. If you don't provide length, zero is assumed.

segno
is an octal segment number.

CONTROL ARGUMENTS

-name, -nm
specifies that the octal number following it is a pathname.

ACCESS REQUIRED

You need write access on the segment to be truncated.

NOTES

If the segment is already shorter than the specified length, its length is unchanged, but the bit count is set to the length given.

Don't use truncate on segments that are, or are components of, structured files.

If you use truncate on a consistent MSF, it is operated on as in a single segment. If the truncation length is less than the current length, components are deleted until the sum of the bit counts of all the components is equal to the truncation length; if the truncation length is greater than that sum, components are created as needed.

EXAMPLES

The command line

```
! tc alpha 50
```

truncates segment alpha to 50 words; i.e., all words from word 50 (octal) on are zero. The bit count of the segment is set to the truncated length.

Name: tutorial

SYNTAX AS A COMMAND

```
tutorial
```

FUNCTION

invokes the Multics Tutorial, a menu-driven introduction to Multics.

NOTES

The Tutorial covers seven main topics: the help system, commands, the text editors, the mail systems, the storage system, logging in and out, and WORDPRO. Operation of the Tutorial is explained as you proceed through it. It also has a glossary facility that enables you to get definitions of terms encountered in the various explanations.

Name: unassign_resource, ur

SYNTAX AS A COMMAND

```
ur resources {-control_args}
```

FUNCTION

unassigns one or more resources that have been assigned to your process by the Resource Control Package (RCP).

ARGUMENTS

resources
: specifies the resources to be unassigned from your process. Currently, the only resources managed by RCP are devices. If a device is attached, it is automatically detached.

CONTROL ARGUMENTS

-admin, -am
forces an unassignment. This control argument should be specified by highly privileged users who want to unassign a resource that is assigned to some other process.

-all
specifies that all devices assigned to the process be unassigned.

-comment STR, -com STR
is a comment string that is displayed to the operator when the resource is unassigned. This comment is displayed only once, even if several resources are being unassigned. (See the assign_resource command for details about comment strings.)

NOTES

This command must not be used to unassign a device attached through the tape_ansi_ or tape_ibm_ I/O module with -retain all specified. In that case, the user must specify argument 1 (no retention) of the retention operation, before detaching the I/O module. See the descriptions of tape_ansi_ and tape_ibm_ I/O modules in the Subroutines manual.

EXAMPLES

In the example that follows, the user unassigns a tape previously assigned by the assign_resource command by typing the command line:

```
! ur tape_03
```

Name: underline

SYNTAX AS A COMMAND

```
underline str_args
```

SYNTAX AS AN ACTIVE FUNCTION

```
[underline str_args]
```

FUNCTION

underlines str_args. Each str_arg is underlined separately in the return value, and this value is canonicalized.

EXAMPLES

```
! underline abcdefg
  abcdefg

! underline "abc def ghij"
  abc def ghij

! underline abc DEF ghij
  abc DEF ghij
```

Name: unique

SYNTAX AS A COMMAND

```
unique {arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[unique {arg}]
```

FUNCTION

returns a unique character string as generated by the unique_chars_ subroutine (described in the Subroutines manual). Unique character strings are 15 characters long and begin with an exclamation mark (!).

ARGUMENTS

arg
: is an octal number from which the unique character string is to be generated. If arg is omitted, the current clock value is assumed.

EXAMPLES

```
!  string [unique]
   !BBBJHwHMzmmxMF

!  string [unique [user process_id]]
   !BPGBzBBBBBBB
```

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
does not print 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 entryname such as a name containing >, <, *, or ? (interpreted literally). It allows you to unlink strangely named links.

ACCESS REQUIRED

You require modify permission on the directory containing the link.

NOTES

Use delete to delete segments and multisegment files. Use delete_dir to delete directories.

For a discussion of links see the Programmer's Reference Manual.

Name: upper_case, uppercase

SYNTAX AS A COMMAND

```
uppercase strings {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[uppercase strings {-control_args}]
```

FUNCTION

returns all strings with lowercase alphabetic characters translated to uppercase characters.

CONTROL ARGUMENTS

-argument, -ag
: specifies that the following argument begins the string to be translated, whether or not it begins with a minus. The default is for the string to begin with the first noncontrol argument.

-leading
: translates only the first character of each word, where words are separated by spaces.

NOTES

Returned strings are separated from each other by single spaces. Each input string is returned as a separate output string, enclosed in quotes if necessary.

EXAMPLES

```
! string [uppercase Now is the time.]
  NOW IS THE TIME.

! string [uppercase "Now is the time"]
  NOW IS THE TIME

! string [uppercase -leading Now is the time.]
  Now Is The Time.
```

Name: user

SYNTAX AS A COMMAND

```
user key
```

SYNTAX AS AN ACTIVE FUNCTION

```
[user key]
```

FUNCTION

returns various user parameters.

LIST OF KEYS

256k_switch, 256k
: returns "true" if 265K segments are allowed in the process. This feature is currently used by FORTRAN programs for very large array. (Default: off)

abs_queue
: is the queue number in which your absentee process is running. It returns "interactive" if you have no absentee process.

absentee
: returns "true" if you are an absentee user, "false" otherwise.

absentee_request_id, abs_rqid
: is the request ID corresponding to this absentee process. Use the request ID only in full-length character string comparisons. Make no assumptions regarding the construction of a request ID by the system. For an interactive or daemon process, the request_id returned is 0.

absentee_restarted
: returns "true" if the absentee process has been restarted after a system crash, "false" otherwise; see the enter_abs_request (ear) and list_abs_requests (lar) commands.

absin
: is the absolute pathname of your absentee input segment, including the absin suffix; otherwise it returns a null string.

absout
: is the absolute pathname of your absentee output segment, including the absout suffix; otherwise it returns a null string.

absout_truncation
: returns "true" if you have used -truncate with ear or lar, "false" otherwise.

user

all
prints all the information available in alphabetical order sorted by keyword name. You can't use it in the active function.

anonymous
returns "true" if you are an anonymous user, "false" otherwise.

attributes
are your attributes determined at login time. They are separated by a comma and a blank and end with a semicolon. You can choose them from the following:

anonymous	multip	nopreempt
bumping	no_eo	nostartup
brief	no_prime	primary_line
daemon	no_secondary	save_on_disconnect
dialok	no_warning	save_pdir
disconnect_ok	nobump	vhomedir
guaranteed_login	nolist	vinitproc
igroup		

auth
is a short character string describing the authorization of your process or "system_low."

auth_long
is a long character string (in quotes) describing the authorization of your process or "system_low."

auth_range
returns your authorization range as a standard low/high aim range.

auth_range_long
returns your authorization range as a standard low/high aim range in long mode.

brief_bit
returns "true" if you specified -brief in the login line, "false" otherwise.

charge_type
is the device charge type associated with your terminal.

cpu_secs
is your CPU usage (in seconds) since login, in the form sss.t, with leading zeros suppressed.

device_channel
is the I/O device channel associated with your terminal.

user

group
is your load control at login.

initial_term_id
is your terminal identifier code at login.

initial_term_type
is your terminal type at login. If you change your terminal type and then do a new process or reconnect after disconnecting, initial_term_type will reflect the new terminal type.

This page intentionally left blank.

limit
is your absolute spending limit in dollars.

limit_type
is your spending reset mode. It can be one of the following:

absolute
spending is never reset.

day
spending is reset each day.

month
spending is reset each month.

year
spending is reset each year.

calendar
spending is reset each calendar year.

fiscal_year
spending is reset each fiscal year.

line_type
is the line type of your terminal. It can have one of the following values:

MC	Sync	SYNC1
TELNET	G115	SYNC2
none	BSC	SYNC3
ASCII	202ETX	POLLED_VIP
1050	ASYNC1	VIP
2741	ASYNC2	
ARDS	ASYNC3	

log_time
is your connect time (in minutes) since login, in the form "mmm.t".

login_date
is the date at login time, in the form "mm/dd/yy".

login_time
is the time of login, in the form "hhmm.t".

login_word
is the word you used to log in (login, enter, or enterp).

max_auth
is a short string describing the maximum authorization of your process or system_low.

max_auth_long
is a long string (in quotes) for the maximum authorization of your process or system_low.

min_auth
returns the user's minimum login authorization.

min_auth_long
returns the user's minimum login authorization in long mode.

monthly_limit
is your monthly spending limit in dollars.

monthly_spending
is your total spending in dollars for the current month.

n_processes
is the number of processes created for you since login: 1 plus the number of new_proc commands plus the number of fatal process errors.

name
is your Person.id at login time.

outer_module
is the initial outer module for the terminal channel.

preemption_time
is the time at which the primary user becomes eligible for group preemption, in the form "hhmm.t".

process_id
is your process identification in octal.

process_type
is your process type. It can have one of the following values:
interactive
absentee
daemon

process_overseer
is the name of your process overseer.

project
is your Project_id.

protected
returns "true" if you are currently a primary user and protected from preemption, "false" otherwise.

rate_structure_name
returns the name of the rate structure that is in effect for this process.

rate_structure_number
returns the number of the rate structure that is in effect for this process.

secondary
returns "true" if you are currently subject to preemption, "false" otherwise.

service_type
is the service type of your terminal (login or FTP).

shift_limit
is your spending limit in dollars for the current shift.

shift_spending
is your total spending in dollars for the current shift within the current month.

spending
is your total spending in dollars.

term_id
is your terminal identifier code. It is "none" if your terminal does not have the answerback feature.

term_type
is your terminal type, which can be any terminal type name defined in the terminal type file described in the Programmer's Reference Manual.

weight
is the loading factor that the system assumes for your process.

Name: validate_pictured_data, vpd

SYNTAX AS AN ACTIVE FUNCTION

```
[vpd pic_string values]
```

FUNCTION

returns "true" if all values can be formatted via pic_string, "false" otherwise.

ARGUMENTS

pic_string
is a valid PL/I picture.

values
is a string to be edited into the picture.

NOTES

For information on PL/I picture and picture strings, see the *PL/I Reference Manual* (Order No. AM83) or the *PL/I Language Specification* (Order No. AG94).

Name: validate_info_seg, vis

SYNTAX AS A COMMAND

```
vis paths {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[vis paths {-control_args}]
```

FUNCTION

validates the syntax of an information segment (info seg). The active function returns the number of the highest severity error that occurs.

ARGUMENTS

paths
are the pathnames of info segs. You need not supply the info suffix. You can use the star convention.

CONTROL ARGUMENTS

-names, -nm
changes the names on the info seg, if necessary, to match the names used in it.

-no_names, -nnm
does not change the names on the info seg, but merely reports discrepancies. (Default)

-severity N, -sv N
* suppresses error messages of severity less than N.

-total, -tt
prints only the total number of errors for each severity or nothing if there are
* no errors.

LIST OF ERROR MESSAGES

The following is a list of error messages printed by vis, in order of severity. N represents line numbers referenced by vis. For information on the proper formatting of, and standards on, info segs, see the help command and info_seg.gi.info.

SEVERITY 4
- Invalid or missing header line
- Header longer than one line
- Invalid date on header line
- Missing Syntax section(s)
- Missing Function section(s)

SEVERITY 3
- Lines >71 chars: N
- Nonstandard section breaks (not 2 blank lines)
- Sections out of sequence: N
- Segment ends in blank lines
- Segment does not end in newline
- Variant section title format:
  - <N section titles>
- Backspaces: N

SEVERITY 2
- Blank lines containing white space: N
- Paragraphs >15 lines: N
- Sections >50 lines: N

SEVERITY 1
- Sections >30 lines: N
- Nonstandard section titles:
  - <N section titles>

*

Name: value_defined, vdf

SYNTAX AS A COMMAND

```
vdf name {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[vdf name {-control_args}]
```

FUNCTION

returns "true" if name has a value set by value_set or by "value_get -call", "false" otherwise. The value can be per process or reside in a value segment (see value_get).

ARGUMENTS

name
is a character string. It can be -name STR to specify a name beginning with a minus sign, to distinguish it from a control argument.

CONTROL ARGUMENTS

-pathname path, -pn path
specifies a value segment other than the current default one, without changing the default (see "Notes on Value Segment").

-permanent, -perm
returns true only if a value is defined in the value segment, regardless of whether a per-process value exists. (Default: to return true for either a per-process or a permanent value)

-perprocess, -pp
returns true only if a per-process value is defined.

ACCESS REQUIRED

You require r access to the value segment, except for per-process values. Lack of r access is equivalent to no value defined in the segment.

NOTES ON VALUE SEGMENT

The value segment searched is either the one specified by -pathname or the current default value segment. The default segment is initially

```
[home_dir]>[user name].value
```

but can be changed by means of value_set_path and listed by value_path. Use of -pathname does not change the default segment.

NOTES

See value_delete, value_get, value_list, value_path, value_set, and value_set_path.

Name: value_delete, vdl

SYNTAX AS A COMMAND

```
vdl {name} {-control_args}
```

FUNCTION

causes one or more names not to have defined values, as set by value_set or value_get -call.

ARGUMENTS

name
is a character string. It can be -name STR to specify a name beginning with a minus sign, to distinguish it from a control argument. (See "Notes.")

CONTROL ARGUMENTS

-all, -a
deletes data values set by value_$set_data as well as other values.

-brief, -bf
suppresses the warning message "No match for starname."

-data
deletes values set by value_$set_data, which you can list by giving -all or -data to value_list. (Default: delete values set by value_set or value_$set)

-exclude STR, -ex STR
deletes all existing values except those for names that match STR. If STR is surrounded by slashes (/), it is interpreted as a qedx regular expression to match names; otherwise, it is interpreted as a starname. Only per-process values are deleted if you supply -perprocess, and only permanent ones if you give -permanent. (See "Notes.")

-long, -lg
allows the warning message "No match for starname." (Default)

-match STR
deletes all existing values for names that match STR. If STR is surrounded by slashes, it is interpreted as a qedx regular expression to match names; otherwise, it is interpreted as a starname. Only per-process values are deleted if you supply -perprocess, and only permanent ones if you give -permanent. (See "Notes.")

-pathname path, -pn path
specifies a value segment other than the current default one, without changing the default (see "Notes on Value Segment").

-permanent, -perm
deletes only values stored in the value segment.

-perprocess, -pp
deletes only per-process values. The default is to delete the per-process value if one exists, otherwise to delete any permanent value.

ACCESS REQUIRED

You require rw access on the value segment except for per-process values.

NOTES

The -match and -exclude control arguments are applied in the order specified. Successive -match control arguments add to the set of names processed (union), and successive -exclude control arguments narrow down the set (intersection). They are incompatible with the name argument and can appear multiple times together. (See "Examples.")

See value_defined, value_get, value_list, value_path, value_set, and value_set_path.

NOTES ON VALUE SEGMENT

The value segment searched is either the one specified by -pathname or the current default value segment. The default segment is initially

```
[home_dir]>[user name].value
```

but you can change it by value_set_path. Use of -pathname does not change the default segment.

EXAMPLES

The following are examples using -match and -exclude:

Assume the defined variables to be

rs_seg_length, rs_area_length, rs_str_len, arg_str_len, buf_size

The command line

```
! vdl -match /_len/ -exclude /_length/ -match /seg_length/
```

operates as follows: The first -match /_len/ makes this set of selected names:

rs_seg_length, rs_area_length, rs_str_len, arg_str_len

The following -exclude /_length/ produces the intersection of this set with the set of names not matching /_length/:

rs_str_len, arg_str_len

The following -match /seg_length/ produces the union of this set with the one matching /_seg_length/:

rs_str_len, arg_str_len, rs_seg_length

Finally, the value of each of these selected variables is deleted.

Name: value_get, vg

SYNTAX AS A COMMAND

```
vg name {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[vg name {-control_args}]
```

FUNCTION

returns the character string value of a name, as set by value_set. If the name has no value and you choose -default, an error occurs. Values, except for per-process values, are stored in a value segment with suffix "value" (see "Notes on Value Segment").

ARGUMENTS

name
is a character string. It can be -name STR to specify a name beginning with a minus sign, to distinguish it from a control argument.

CONTROL ARGUMENTS

-call STR
if no value is found for name, the active function [STR] is expanded to produce a value, which is both set for name and returned. Enclose STR in quotes and omit the brackets if it contains special characters such as spaces. This control argument is incompatible with -default.

-default STR, -df STR
specifies a default value to be returned if none is set. Enclose STR in quotes if it contains special characters. A null string is returned if STR is "". If you don't give -default and no value exists, an error occurs.

-pathname path, -pn path
specifies a value segment other than the current default one, without changing the default (see "Notes on Value Segment"). It is incompatible with -perprocess.

-permanent, -perm
does not look for a per-process value. The default is to return the per-process value if one exists, otherwise to return the value stored in the value segment; if none exists, an error occurs.

-perprocess, -pp
looks only for a per-process value, not for one stored in any value segment. If a per-process value is not found, an error occurs.

-pop
deletes the current value that it prints or returns. If a previous value was saved by value_set -push, that value is reinstated.

ACCESS REQUIRED

You require read access on the value segment, except for per-process values.

NOTES

Per-process values are stored in a temporary value segment in the process directory and disappear when the process terminates.

By default, both "vg name" and "vg name -pn path" return the per-process value of name if there is one; otherwise, they return the value stored in the appropriate value segment. By contrast, "vg -pp" returns only the per-process value, and "vg -perm" returns only the one in the value segment.

See value_defined, value_delete, value_list, value_path, value_set, and value_set_path.

NOTES ON VALUE SEGMENT

The value segment searched is either the one specified by -pathname or the current default value segment. The default segment is initially

```
[home_dir]>[user name].value
```

but you can change it by value_set_path. Use of -pathname does not change the default segment.

Name: value_list, vls

SYNTAX AS A COMMAND

```
vls {name} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[vls {name} {-control_args}]
```

FUNCTION

lists one or more name-value pairs as set by value_set and value_get -call.

ARGUMENTS

name
: is a character string. It can be -name STR to specify a name beginning with a minus sign, to distinguish it from a control argument. (See "Notes.")

CONTROL ARGUMENTS

-all, -a
: lists variables set by value_$set_data in addition to the variables set by value_$set and the value commands. These are listed in the form

```
foo                 (N words)
```

Word counts alone are listed for data variables since their values have meaning only to the caller of value_. If you select -all, the default is to omit the data variables.

-brief, -bf
: suppresses the error messages allowed by -long.

-data
lists only the values set by value_$set_data.

-depth N, -dh N
lists the latest N-1 pushed values for any variable in addition to the current value. Any further pushed values result in the message "(M more pushed values)". The default is to print the latest value followed by the message "(M pushed values)". In the active function, -depth returns only the latest N values.

-exclude STR, -ex STR
lists all values except those for names that match STR. The character string STR is searched for in names: if it is surrounded by slashes (/), it is interpreted as a qedx regular expression to match names; otherwise it is interpreted as a starname. Only per-process values are listed if you supply -perprocess, and only permanent ones if you give -permanent. (See "Notes.")

-long, -lg
allows the error messages "Name not found" and "No match for..." for individual name and -match arguments. (Default)

-match STR
lists all values for names that match STR. The character string STR is searched for in names: if it is surrounded by slashes (/), it is interpreted as a qedx regular expression to match names; otherwise it is interpreted as a starname. Only per-process values are listed if you supply -perprocess, and only permanent ones if you give -permanent. (See "Notes.")

-pathname path, -pn path
specifies a value segment other than the current default one, without changing the default (see "Notes on Value Segment"). You are allowed multiple -pathname control arguments to list values in more than one value segment.

-permanent, -perm
lists only values stored in the value segment.

-perprocess, -pp
lists only per-process values.

-value, -val
lists values only.

-variable, -var
lists variable names only.

ACCESS REQUIRED

You require read access on the value segment, except for per-process values.

NOTES

The list is sorted alphabetically by name, the per-process value first where both exist.

By default, this command lists both variable names and values, and both per-process and permanent values interspersed, the per-process names preceded by "(P)".

Either -value or -variable is required by the active function. The active function returns the selected names or values separated by spaces.

The -match and -exclude control arguments are applied in the order specified. Successive -match control arguments add to the set of names processed (union), and successive -exclude control arguments narrow down the set (intersection). They are incompatible with the name argument and can appear multiple times together. (See "Examples.")

See value_defined, value_delete, value_get, value_path, value_set, and value_set_path.

NOTES ON VALUE SEGMENT

The value segment searched is either the one supplied by -pathname or the current default value segment. The default segment is initially

```
[home_dir]>[user name].value
```

but you can change it by value_set_path. Use of -pathname does not change the default segment.

EXAMPLES

The following are examples using -match and -exclude:

Assume the defined variables to be

rs_seg_length, rs_area_length, rs_str_len, arg_str_len, buf_size

The command line

```
! vls -match /_len/ -exclude /_length/ -match /seg_length/
```

operates as follows:

The first -match /_len/ makes this set of selected names:

rs_seg_length, rs_area_length, rs_str_len, arg_str_len

The following -exclude /_length/ produces the intersection of this set with the set of names not matching /_length/:

rs_str_len, arg_str_len

The following -match /seg_length/ produces the union of this set with the set of names matching /seg_length/:

rs_str_len, arg_str_len, rs_seg_length

Finally, the value of each of these selected variables is listed.

Name: value_path, vp

SYNTAX AS A COMMAND

vp

SYNTAX AS AN ACTIVE FUNCTION

[vp]

FUNCTION

returns the pathname of the current default value segment used by the value commands without -pathname.

NOTES

See value_defined, value_delete, value_get, value_list, value_set, and value_set_path.

Name: value_set, vs

SYNTAX AS A COMMAND

```
vs {name} {value_string} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[vs {name} {value_string} {-control_args}]
```

FUNCTION

associates a character string name with a character string value. The value replaces any previous value for name.

ARGUMENTS

name
is a character string. It can be -name STR to specify a name beginning with a minus sign, to distinguish it from a control argument. There is no restriction on the length of the name.

value_string
is a character string value, quoted if it contains blanks or other special characters. It can be -value STR to specify a value STR that begins with a minus sign, to distinguish it from a control argument. There is no restriction on the length of the value.

CONTROL ARGUMENTS

-add N
adds N to the integer value of each name selected by the other control arguments. If any of the names has no value or has a value that is not the character string representation of an integer, an error occurs. N can be negative or zero, as can be the resulting value.

-exclude STR, -ex STR
changes all existing associations except those for names that match STR. If STR is surrounded by slashes (/), it is interpreted as a qedx regular expression to match names; otherwise, it is interpreted as a starname. Only per-process associations are changed if you select -perprocess, only permanent ones if you supply -permanent, and both are changed by default. (See "Notes.")

-if VALUE_STR
sets value_string only if an old value exists and is equal to VALUE_STR, otherwise returns an error. If you also specify -match and/or -exclude, all selected names with current values equal to VALUE_STR are set to value_string.

-match STR
changes all existing associations for names that match STR. If STR is surrounded by slashes, it is interpreted as a qedx regular expression to match names; otherwise, it is interpreted as a starname. Only per-process associations are changed if you select -perprocess, only permanent ones if you supply -permanent, and both are changed by default. (See "Notes.")

-pathname path, -pn path
specifies a value segment other than the current default one, without changing the default (see "Notes on Value Segment").

-permanent, -perm
sets a value in the value segment, regardless of whether any old value is per process or permanent. The default is to change any existing per-process value, otherwise to change the permanent value if one exists, otherwise to set a permanent value.

-perprocess, -pp
sets a per-process value, regardless of whether any old value is per process or permanent.

-pop
restores the previous value, saved by an invocation of "value_set -push", for each variable specified on the command line. If any given variable lacks a previous value, an error message is printed and the other variables' values are still popped. This control argument is incompatible with specifying a value and with -push.

-push
saves the old value of each variable before setting the value specified.

-update, -ud
makes the active function return the previous value or null string if there is no previous value. (Default: return the value that is set)

ACCESS REQUIRED

You need rw access on the value segment, except for per-process values.

NOTES

You must give one of value_string, -value STR, -add, or -pop.

The -match and -exclude control arguments are applied in the order specified. Successive -match control arguments add to the set of names processed (union), and successive -exclude control arguments narrow down the set (intersection). They are incompatible with the name argument and can appear multiple times together. (See "Examples.") You can't use either -match or -exclude in the active function.

If you supply -perprocess or the old value is a per-process one, the value set is per process; otherwise, the association is stored in a value segment (see "Notes on Value Segment"). Per-process values are stored in a temporary value segment in the process directory and disappear when the process terminates.

When a value is set in a value segment that does not exist, you are asked whether to create the segment. Your default value segment [hd]>[user name].value is created automatically and a message is printed.

See value_defined, value_delete, value_get, value_list, value_path, and value_set_path.

NOTES ON VALUE SEGMENT

The value segment searched is either the one specified by -pathname or the current default value segment. The default segment is initially

```
[home_dir]>[user name].value
```

but you can change it by value_set_path. Use of -pathname does not change the default segment.

EXAMPLES

The following are examples using -match and -exclude:

Assume the defined variables to be

rs_seg_length, rs_area_length, rs_str_len, arg_str_len, buf_size

The command line

```
! vs 0 -match /_len/ -exclude /_length/ -match /seg_length/
```

operates as follows:

The first -match /_len/ makes this set of selected names:

rs_seg_length, rs_area_length, rs_str_len, arg_str_len

The following -exclude /_length/ produces the intersection of this set with the set of names not matching /_length/:

rs_str_len, arg_str_len

The following -match /seg_length/ produces the union of this set with the set of names matching /_seg_length/:

rs_str_len, arg_str_len, rs_seg_length

Finally, the value of each of these selected variables is set to 0.

Name: value_set_path, vsp

SYNTAX AS A COMMAND

```
vsp {path} {-control_arg}
```

FUNCTION

sets the default value segment used by the value commands without -pathname.

ARGUMENTS

path
: is the pathname of a value segment or a nonexistent segment, which is created. The value suffix is assumed. If you don't give path or is equal to the null string (""), it is restored to your default value segment.

CONTROL ARGUMENTS

-brief, -bf
: suppresses the warning printed when you lack write access to the value segment.

ACCESS REQUIRED

You need at least r access to the value segment, and rw is preferred. If you lack r access, the default path is not changed and an error message is printed; if you lack rw, the default path is changed, but a warning is printed. You can use -brief to suppress this warning.

NOTES

The default value segment in a process is initially

```
[home_dir]>[user name].value
```

See value_defined, value_delete, value_get, value_list, value_path, and value_set.

Name: verify

SYNTAX AS A COMMAND

```
verify STRA STRB
```

SYNTAX AS AN ACTIVE FUNCTION

```
[verify STRA STRB]
```

FUNCTION

returns an integer representing the first character position in strA that contains a character that does not occur anywhere in strB. If every character of strA occurs in strB, 0 is returned.

EXAMPLES

The following interactions illustrate the verify active function.

```
! string [verify chart chapter]
  0

! string [verify chapter chart]
  4

! string [verify 31 0123456789]
  0

! string [verify 31q22 0123456789]
  3
```

Name: vfile_adjust, vfa

SYNTAX AS A COMMAND

```
vfa path {-control_arg}
```

FUNCTION

adjusts structured files left in an inconsistent state by an interrupted opening, or unstructured files in any state.

ARGUMENTS

path
is the pathname of a file to be adjusted.

CONTROL ARGUMENTS
must be specified only for unstructured files.

-set_bc
sets the bit count of the file's last nonempty segment to the last nonzero byte in that segment. Any components beyond it are deleted.

-set_nl
appends a newline character if the last nonzero byte in the file is not a newline character. The bit count of the file's last nonempty segment is then set to the file's last nonzero byte (which is now sure to be a newline character).

-use_bc {N}
truncates the file to the byte specified by the bit count of multisegment file component N. If N is not given, it is taken to be the last nonempty component.

-use_nl
truncates the file after the last newline character.

NOTES

For unstructured files a control argument must specify the desired adjustment; otherwise, no control arguments are allowed. A sequential or blocked file is adjusted by truncation after the last complete record. An indexed file is adjusted by finishing the interrupted operation.

The adjust_bit_count command used with -character is equivalent to vfile_adjust used with -set_bc except that the latter only operates on a file that appears to be unstructured.

See the description of the vfile_ I/O module in the Subroutines manual for further details.

Name: vfile_find_bad_nodes

SYNTAX AS A COMMAND

```
vfile_find_bad_nodes {path} {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[vfile_find_bad_nodes {path} {-control_args}]
```

FUNCTION

examines a vfile_ keyed file to determine whether the vfile_ multisegment file (MSF) components that contain keys are in a consistent state. The keys in a keyed file are maintained in a tree structure in which each node of the tree is stored in a separate page of an MSF component. The consistency checks that are performed are summarized below. Nodes reported as bad by this heuristic are almost certainly damaged.

ARGUMENTS

path
: is pathname of the indexed file whose nodes are to be checked.

CONTROL ARGUMENTS

-check MODES, -ck MODES
: enables only the types of checking given in the MODES string (see "List of Modes" below). (Default: -check default)

-io_switch STR, -isw STR
: identifies an I/O switch that is already attached to the indexed file to be checked. The switch may be closed; if open, it must be opened for keyed_sequential_input.

-no_request_loop, -nrql
: prints information about the bad nodes and then continues checking without entring the request loop. (Default: when invoked as an active function)

-request_loop, -rql
: enters the request loop when bad nodes are found. (Default: when invoked as a command)

LIST OF MODES

all
: is a shorthand for enabling all possible checking. It it equivalent to node_branch,key_region,key_loc,key_overlap,key_order,node_tree.

default
is a shorthand way of enabling checks that can be quickly performed. It is equivalent to node_branch,key_region,key_loc. The settings of other modes are not affected.

key_loc, ^key_loc
performs key-location checking, as described below.

key_order, ^key_order
performs key-order checking, as described below.

key_overlap, ^key_overlap
performs key-overlap checking, as described below.

key_region, ^key_region
performs key-region checking, as described below.

node_branch, ^node_branch
performs node-branch checking, as described below.

node_tree, ^node_tree
performs node-tree checking, as described below.

LIST OF CONSISTENCY CHECKS

The following consistency checks are always performed to validate the file header:

1) Does the counted number of nonempty (key-containing) nodes equal the count stored in the file header? If not, the file header may have been damaged.
2) Does the counted number of keys in all nodes equal the count stored in the file header? If not, the file header may have been damaged.
3) Does the counted total length of all keys equal the count stored in the file header? If not, the file header may have been damaged.

For each node in the file the following consistency checks are performed:

4) Is this a freed node? If so, skip further checks.
5) Are there any branches (keys) in this node? If not, skip further checks.

Node-branch checks

6) Is branch_count greater than 313? If so, node is bad because space in a page limits a node to having, at most, 313 one-character keys.
7) Is branch_count less than 0? If so, node is bad.

vfile_find_bad_nodes

Key-region checks

8) Is start_of_key_region greater than 4096? If so, the node is bad because the character position of the first key must lie within the node page.
9) Does start_of_key_region overwrite the branch array? If so, the node is bad because keys have overwritten the array of branches in the node.
10) Is scattered_free_key_space greater than 4096 minus start_of_key_region? If so, the node is bad because the count of unused space within the key region is greater than the size of the key region itself.
11) Is scattered_free_key_space less than 0? If so, the node is bad.
12) Is length of all keys in node equal to 4096 minus start_of_key_region minus scattered_free_key_space? If not, the node header is bad.

Key-location checks

13) Does any branch declare its key to begin prior to start_of_key_region? If so, the node is bad.
14) Does any branch declare its key to extend beyond the end-of-node page? If so, the node is bad.

Key-overlap check

15) Does the storage for any key overlap storage for another key? If so, the node is bad. This test is somewhat time consuming.

Key-order check

16) Are the keys within the node ordered in increasing ASCII collating sequence? If not, the node is bad. This test is somewhat time consuming.

Node-tree check

17) For each child pointer in the node, does the child pointer reference another node that resides in a node-containing component of the vfile? If not, the child pointer is bad.
18) Does the child pointer reference another node that contains keys (is nonempty)? If not, the child pointer is bad.
19) Does the child pointer reference another node that is not on the list of freed nodes? If not, the child pointer is bad.
20) Does each child pointer in the node reference another node that is not the root node? If not, the child pointer is bad.
21) Is every nonempty node but the root node referenced by a child pointer of some other node? If not, the node is inaccessible and its keys are effectively not part of the key tree.
22) Is any nonempty node referenced by more than one superior node? If so, the key tree is inconsistent.

REQUEST LOOP OPERATION

When a bad node is found, its location is printed out, followed by the number of branches in the node, its low_key_pos, and its unused key space (scat_space). Then a request loop is entered that allows you to continue checking other nodes, to quit further checking, or to enter a totaling loop that counts the number of damaged nodes in the current component without printing their statistics. The request

..ds node_seg node_offset count -ch

is useful. Type "c" in the request loop to continue checking the next node.

LIST OF REQUESTS

gives the name and version number of this program, plus pathname or I/O switch of the file being examined.

.. command_line"
escapes Multics command level to execute command_line.

?
lists available requests.

continue, c
continues searching for damaged nodes.

quit, q
stops further processing, reporting total of damage found so far.

total, tt
stops reporting, for the remainder of this MSF component, information about each damaged node and counts the damaged nodes in this component.

NOTES

Give either a pathname argument or -io_switch to identify the file to be checked.

As an active function, returns "true" if bad nodes are found, "false" otherwise. Normal diagnostic messages are still printed.

EXAMPLES

```
!  vfile_find_bad_nodes >scl>perm_syserr_log -ck default,key_order
[1]
[2]     Begin checking free node list (node_ptr = 473|314000).
[3]     Found 59 undamaged free nodes.  Processing continues.
[4]
[5]     Begin checking component 0, node:
[6]      25 50 75 100 125 150 175 200 225 250
[7]     Begin checking component 6, node:
[8]      25 50 75 100 125 150 175 200 225 250
[9]     No damaged nodes.
```

Lines 2 and 3 of the output show that the key-containing components of the file contain some unused node pages. These free node pages are catalogued, and no further checking occurs on them.

Line 5 shows the beginning of testing in component 0 of the file. Each component contains 255 pages, numbered from 1 to 255. The numbers printed on line 6 show the progress of checking through these pages (i.e., 25 is printed after the first 25 pages are checked, 50, when 50 pages are checked, etc.).

Line 9 is printed when no damage is found.

```
   vfile_find_bad_nodes user_reg -check all
[1]
[2]     Begin checking component 0, node:
[3]
[4]     ERROR 13 in Comp 0, node 5 (node_ptr = 464|10000)
[5]     Key(2) > Key(3)
[6]         branch_count = 203 keys
[7]         start_of_key_region = char position 2470
[8]         key_space = 1626 chars,
[9]         scattered_free_key_space = 0 chars
[10]    vfile_find_bad_nodes:    c
[11]
[12]    ERROR 6 in Comp 0, node 6 (node_ptr = 464|12000)
[13]    branch_count > 313
[14]        branch_count = 9420723823 keys
[15]        start_of_key_region = char position 15733420590
[16]        key_space = -15733416494 chars,
[17]        scattered_free_key_space = 11171849844 chars
[18]    vfile_find_bad_nodes:    tt
[19]     25 50 75 100 125 150 175 200 225 250
[20]    4 bad nodes in comp 0
[21]
[22]    Begin checking references between 247 nonempty tree
[23]    nodes:
[24]
[25]    ERROR 21, Comp 0, node 7 (node_ptr= 464|14000)
```

```
[26]    never referenced by superior node and it is not the
[27]    root node.
[28]
[29]    5 key nodes were damaged.
```

Line 2 shows the beginning of checking on component 0 of another file. The error message on lines 4 to 9 shows a key-order error in node 5 of component 0.

Line 10 shows the request loop. The c request was issued to continue checking.

Lines 12 to 17 show a second error, in node 6 of component 0. In this error, more than 313 keys were found in the node.

Line 18 shows issuing tt gets a count of the remaining errors in component 0. The count is shown in line 20--4 bad nodes--which includes the two for which errors were shown plus two others.

Lines 22 and 23 show the beginning of the check that insures that all nodes but the root are children of some other node.

Lines 25 to 27 show an error in a node that contains keys but is never referenced by any superior node in the key tree.

Line 29 prints a summary of all checking, showing that a total of five key nodes were found among all the components of the file.

Name: vfile_status, vfs

SYNTAX AS A COMMAND

```
vfs path
```

FUNCTION

prints information about and the apparent type (unstructured, sequential, blocked, or indexed) and length of files.

ARGUMENTS

path
: is the pathname of a segment or multisegment file. If the entryname portion of a pathname denotes a directory, it is ignored. If no files are found for the given pathname, a message is printed. If the entry is a link, the information returned pertains to the entry to which the link points. The star convention is allowed.

NOTES

For structured files, information about the state of the file (if busy) and the file version (unless current) is printed. For blocked files the maximum record length is printed. For indexed files the following statistics are printed:

1. the number of records in the file, including zero-length records
2. the number of nonnull records in the file, if different from the above
3. the total length of the records (bytes)
4. the number of blocks in the free-space list for records
5. the height of the index tree (equal to zero for empty files)
6. the number of nodes (each 1K words, page aligned) in the index tree
7. the total length of all keys (bytes)
8. the number of keys (if different from record count)
9. the number of duplicate keys (if nonzero)
10. the total length of duplicate keys (if any).

For additional information see the status command.

EXAMPLES

Assume that the file foo is in your working directory. The command line

```
vfile_status foo
```

might produce the following output:

```
type: unstructured
bytes: 4993
```

if the file is unstructured, or:

```
type: sequential
records: 603
```

if the file is sequential, or:

```
type: blocked
records: 1200
max recl: 7 bytes
```

if the file is blocked, or:

```
type: indexed
records: 397
state: locked by this process
action: write in progress
record bytes: 3970
free blocks: 1
index height: 2
nodes: 3
key bytes: 3176
```

if the file is indexed and a write operation has been interrupted in your process.

Name: walk_subtree, ws

SYNTAX AS A COMMAND

```
ws path command_line {-control_args}
```

FUNCTION

executes a specified command line in the directory selected (called the starting node) and in its inferior directories. It prints the pathname of every directory in which the command line is executed.

ARGUMENTS

path
: is the starting node. This must be the first argument. A path of -working_directory (-wd) specifies the working directory.

command_line
: is the command line to be executed. The entire command line is taken to be a single argument. A multiple-word command line should be typed as a quoted string.

CONTROL ARGUMENTS

-brief, -bf
: suppresses printing of the names of the directories in which the command line is executed.

-bottom_up, -bu
causes execution of the command line to commence at the last level and to proceed upward through the storage system hierarchy until the first level is reached. In the default mode, execution begins at the highest (first) level and proceeds downward to the lowest (last) level.

-first N, -ft N
makes N the first level in the storage system hierarchy at which the command line is to be executed, where, by definition, the starting node is level 1. The default is -ft 1.

-last N, -lt N
makes N the last level in the storage system hierarchy at which the command line is to be executed. The default is -lt 99999, i.e., all levels.

-msf
treats multisegment files as directories, which, normally, are not considered as such.

-priv
invokes a highly privileged primitive to list directories. It requires access to the hphcs_ gate.

NOTES

This command has a cleanup handler--if one quits out of it and immediately types "rl" (release), one's directory is changed back to what it was prior to walk_subtree's invocation.

EXAMPLES

To list all segments in the current working directory having a two-component name with a second component of "pl1" the user types:

```
! ws -wd "list *.pl1"
```

To list two-component names with a second component of "pl1" in directories subordinate to the working directory named George, the user types:

```
! ws >udd>m>George "list *.pl1" -all
```

Name: watch

SYNTAX AS A COMMAND

```
watch {locations} {-control_args}
```

FUNCTION

manipulates the table of virtual memory locations under scrutiny by the trace facility, which can monitor up to 1024 locations for content changes. The word at each location is checked whenever any entrypoint in the trace table is called or returns. When a change is detected, execution is stopped and the old and new contents are displayed. (See the trace command for more information about the trace facility.)

ARGUMENTS

locations
represent memory locations in a form acceptable to the cv_ptr_ subroutine. You must separate multiple locations by spaces (refer to virtual pointers in Section 1).

CONTROL ARGUMENTS
tell what to do with the specified locations. If you select neither -add nor -remove, -add is assumed.

-add
adds the specified locations to the watch table. When a location is added to the watch table, its content is saved there for later comparison; when a specified location is already in the watch table, its saved content is brought up to date.

-changed
specifies all locations with content changes.

-remove, -rm
removes the specified locations from the watch table.

-status, -st
displays the specified locations and their contents from the watch table.

NOTES

To prevent trace messages while watching, use the "-trace off" argument to the trace command when adding entrypoints to the trace table.

The order of entrypoints in the watch table is determined by their segment numbers and word offsets. The table is ordered first by ascending segment number and then by ascending word offset.

All locations in the watch table are checked whenever any entrypoint in the trace table is called or returns unless the entrypoint is off or all tracing is disabled. If any locations have changed, the trace facility prints a stop message, displays the old and new contents, and calls the -stop_proc. When the -stop_proc returns, the new contents are recorded in the watch table and execution resumes. Unless you change it, the -stop_proc is the command processor, so you can invoke a debugger to get further information. To get a trace of the changing contents without stopping, use the command "trace -stop_proc nothing".

NOTES ON THE SYNTAX OF LOCATIONS

Acceptable representations include the following:

```
pathname|word_number
pathname                    (same as pathname|0)
pathname|entryname
pathname$entryname          (pathname must contain > or <)
reference_name$word_number
reference_name$             (same as reference_name$0)
reference_name$entryname
segment_number|word_number
segment_number              (same as segment_number|0)
segment_number|entryname
*                           (all locations in the watch table)
```

Pathnames can be relative or absolute. All numbers are octal. Bit offsets are ignored. If you use *, you can't specify any other locations.

EXAMPLES

The command sequence

```
! watch
```

tells the number of locations in the watch table and lists the action control arguments.

The command sequence

```
! watch 244|0 333|100
! trace bound_graphics_$* -trace off
! some script
```

causes trace to watch two words: word 0 in segment number 244 (octal) and word 100 (octal) in segment number 333 (octal). These words are checked for changes in their contents each time any entrypoint in bound_graphics_ is called or returns. If either word has changed, trace prints the name of the entrypoint, displays the old and new contents, and stops execution by calling the command processor. You can resume execution with the start command; execution is stopped again if further changes occur.

The command sequence

```
! watch 234|(0 1 2 3 4 5 6 7)
! trace -automatic on -buffer on
! some script
```

watches eight locations during the running of a script. Automatic mode traces everything (except gates, which can't currently be traced). Specifying "-buffer on" records all calls, returns, unwinds, and signals in a circular trace buffer. Whenever one or several of the locations change, a message is printed and execution is halted. You can display the events leading up to the change by typing "trace -print_buffer N" (where N is the number of events you want to see).

The command line

```
watch -status *
```

displays every location in the watch table and its contents.

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 file or entry point.

ARGUMENTS

names
are file and entry point names. You can't use the star convention.

CONTROL ARGUMENTS

-all, -a
lists the pathnames of all files and entry points with the specified names that you can find using the current search rules, your effective access to each segment or entry point, and the name of the search rule used to find each file or entry point. It is incompatible with -lg.

-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, where searches for the entry point name$name.

-inhibit_error, -ihe
does not print an error message if no file can be found for a given name. For the command no output is printed, for the active function the null string is returned.

-long, -lg
prints the pathname, the name of the search rule used to find each segment or multisegment file (MSF), and your effective access to the segment.

-no_inhibit_error, -nihe
prints an error message if no segments or MSFs can be found for a given name. (Default)

-segment, -sm, file
searches for segments or MSFs. (Default, unless name contains a $)

NOTES

This command prints out the full pathname of the file, using its primary name, and the entry point name if you request one. If the file 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 you supply -a, where prints information only about the first matching file or entry point encountered.

The -ep and -file control arguments are mutually exclusive. If you provide one of them, all the name arguments are assumed to be of the type specified. If you provide neither -ep nor -file, where scans the name arguments: Any name arguments that contain a dollar sign are assumed to be names of entry points; all others, names of files.

See "Search Rules" in the Programmer's Reference Manual.

NOTES ON ACTIVE FUNCTION

The active function returns the pathname of the segment. You can't use -a, -bf, and -lg. Unless you supply -ihe, an error occurs if no segment or MSF can be found.

EXAMPLES

If you have a private copy of the cwd command in your working directory, and you have initiated that copy, the command line

```
! wh cwd -all
```

prints three lines:

```
>udd>Project>Person>wd>cwd (re) search rule "initiated_segments"
>udd>Project>Person>wd>cwd (re) search rule "wd"
>sss>cwd (re) search rule "system_library_standard"
```

Name: where_doc, wdoc

SYNTAX AS A COMMAND

```
wdoc topic_name {-control_args}
```

FUNCTION

returns the names of manuals that contain information about the specified topic name.

ARGUMENTS

topic_name
is the name of a topic, command, subroutine, or I/O module. Use iteration to get more than one topic (see "Examples" below).

CONTROL ARGUMENTS

-all, -a
prints all the sections of manual information.

-allow_partial_matches, -apm
searches for a partial match of the topic name if an exact match is not found.

-audience, -aud
describes the audience for which the manual is intended.

-database_pathname PATH, -dbpn PATH
specifies the pathname of the database you want instead of the default one. Once you supply -database_pathname, the specified database is used for all subsequent invocations of wdoc during your process until you select another database.

-description, -desc
returns a brief description of the manual's contents. (Default)

-dont_allow_partial_matches, -dapm
does not allow partial matches. (Default)

-new_features, -nf
lists all new features that have been added to the manual with the last update (revision or addendum).

-no_audience, -no_aud
does not describe the manual's intended audience. (Default)

-no_description, -no_desc
suppresses printing of the brief description of the manual's contents.

-no_new_features, -no_nf
does not list new features. (Default)

-no_table_of_contents, -no_toc
does not print the manual's table of contents. (Default)

-output_file PATH, -of PATH
directs the output to a file instead of to your terminal.

-table_of_contents, -toc
prints the manual's table of contents.

NOTES

If you can't find a match for topic_name and didn't select -apm or -dapm, you are asked whether or not you want wdoc to search for partial matches.

When you use a control argument giving additional information about the manuals found (e.g., -aud or -toc) and wdoc finds more than one manual with information about the topic_name, a menu containing the names of the manuals is displayed. You can then choose to see the information on one or more of the manuals listed or return to command level. *

EXAMPLES

When wdoc finds a manual that contains the topic name, it displays the manual's title, order number, and the Multics release that the current revision or addendum supports. For example,

```
! wdoc dynamic linking

Title:  Introduction to Programming on Multics
Order No.:  AG90-03
Release Supported:  MR9.0
```

Certain control arguments enable you to get additional information about any manuals found:

```
! wdoc help -audience

Title: Multics Commands and Active Functions
Order No.: AG92-05
Release Supported: MR10.1

Audience:
   Programmers and nonprogrammers who use Multics commands and
   active functions.
```

When using iteration, enclose in quotes names containing blank spaces; for instance,

```
wdoc (access create_dir "user ring")
```

Name: where_search_paths, wsp

SYNTAX AS A COMMAND

```
wsp search_list entryname {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wsp search_list entryname {-control_arg}]
```

FUNCTION

prints or returns the absolute pathname(s) of entryname when you give search_list and entryname. The search for the entryname is made using the current search paths contained in the specified search list.

ARGUMENTS

search_list
is the name of the search list searched.

entryname
is the entryname sought.

CONTROL ARGUMENTS

-all, -a
specifies that all occurrences of this entryname found by probing this search list should be returned.

ACCESS REQUIRED

You must have s access on the containing directory or nonnull access to the entry.

NOTES

For a complete list of the search facility commands see add_search_paths.

EXAMPLES

To find the include file struct.incl.pl1 using the translator search list, type:

```
! wsp translator struct.incl.pl1
  >user_dir_dir>Project_id>Person_id>struct.incl.pl1
```

Here is a case in which -all is selected and there is more than one occurrence of the entryname chosen:

```
! wsp translator struct.incl.pl1 -all
  >user_dir_dir>Project_id>Person_id>struct.incl.pl1
  >user_dir_dir>Project_id>include>struct.incl.pl1
  >library_dir_dir>include>struct.incl.pl1
```

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 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, name, and project of each user. 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 you supply none of -interactive, -absentee, or -daemon and give no User_ids, then all interactive and absentee users are listed; but if you specify User_ids, then all matching users are listed. If you provide one or more of -interactive, -absentee, or -daemon, only processes of the selected type(s) are listed; if you also select User_ids, then only users matching those control arguments and the User_ids are listed.

Absentee users are denoted in the list by an asterisk following Person_id.Project_id.

If you omit -name and -project, the output is sorted on login time. You can't use both arguments together because the sort is performed on one key at a time.

If you supply a User_id, the header is suppressed even if you give -long. If you use who with no arguments, the system responds with a two-line header followed by a list of interactive users sorted according to login time.

Sometimes a Person_id.Project_id returned by the command is followed by a "D"* and/or an "S", where "D" refers to a disconnected process and "S" refers to a suspended process.

You can prevent your own name from being listed by all users' invocations of who; to do this, see your project administrator.

NOTES ON ACTIVE FUNCTION

The active function returns a list of Person_id.Project_id pairs, requoted and separated by spaces. You can use control arguments to select and sort.

Name: window_call, wdc

SYNTAX AS A COMMAND

```
wdc arguments {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wdc arguments {-control_args}]
```

FUNCTION

provides a command interface to the video system (see the Programmer's Reference Manual for a description of the video system).

ARGUMENTS

are listed below; some of them require control arguments. A detailed description follows the control arguments section.

bell
change_window, chgwd
clear_region, clrgn
clear_to_end_of_line, cleol
clear_to_end_of_window, cleowd
clear_window, clwd
create_window, crwd
delete_chars, dlch
delete_window, dlwd
get_echoed_chars, gech
get_first_line, gfl
get_one_unechoed_char, gouch
get_position, gpos
get_terminal_height, gtmhgt
get_terminal_width, gtmwid
get_unechoed_chars, guch
get_window_height, gwdhgt
insert_text, itx
invoke
overwrite_text, otx
revoke
scroll_region, scrgn
set_position, spos
set_position_rel, sposrel
supported_terminal
sync
video_invoked
write_sync_read, wsr

CONTROL ARGUMENTS

-column N, -col N
specifies a column on the screen. If you don't give it, the default is the remainder of the screen. (Default: 1, the leftmost column)

-count N, -ct N
specifies a count.

-height N, -hgt N
specifies the height of a region or a window for a request. If you don't supply it, the default is the remainder of the screen.

-io_switch STR, -is STR
specifies the name of an I/O switch for a window. This serves to identify the window. If you don't provide it, user_i/o is assumed.

-line N
specifies a line on the screen. (Default: 1, the top line)

-line_speed N, -ls N
specifies the speed of the terminal's connection to Multics, where N is in characters per second.

-string STR, -str STR
specifies a text string for display, where N must be quoted if it contains blanks.

-terminal_type STR, -ttp STR
states the name of the terminal type, where STR is a terminal type. To see accepted terminal types, type "print_terminal_types."

-width N, -wid N
specifies the width of a region for a request. If you don't provide it, the default is the remainder of the screen.

LIST OF ARGUMENTS

bell

SYNTAX AS A COMMAND

```
wdc bell {-io_switch STR}
```

FUNCTION

activates the terminal bell. On some terminals, this may produce a visual indication instead of an audible tone. The cursor position must be defined. The cursor is positioned to the current position of the specified window, if it is elsewhere on the screen.

change_window, chgwd

SYNTAX AS A COMMAND

```
wdc chgwd {-line N} {-column N} {-height N} {-width N} {-io_switch STR}
```

FUNCTION

changes the origin or size of the specified window. You must give at least one of -column, -height, -line, or -width. If you give only -line (changing the top line of the window), the window length is automatically adjusted: i.e., if -line increases the value of the top line number (moving the window down), the window length shrinks accordingly; however if -line decreases the top line number (moving the window up), the length remains the same. If you supply only -height (changing the window length), the origin line remains the same. If you select only -width (changing the window width), the origin column remains the same.

clear_region, clrgn

SYNTAX AS A COMMAND

```
wdc clrgn -line N -column N -height N -width N {-io_switch STR}
```

FUNCTION

clears the specified rectangular region of the window to blanks. The region may be part or all of the window.

clear_to_end_of_line, cleol

SYNTAX AS A COMMAND

```
wdc cleol {-io_switch STR}
```

FUNCTION

clears the line from the current cursor position to the end of the line to blanks. You must define the current cursor position.

clear_to_end_of_window, cleowd

SYNTAX AS A COMMAND

```
wdc cleowd {-io_switch STR}
```

FUNCTION

clears the window from the current cursor position to the end of the window to blanks. You must define the current cursor position.

clear_window, clwd

SYNTAX AS A COMMAND

```
wdc clwd {-io_switch STR}
```

FUNCTION

clears the specified window so that its content becomes entirely blank. The current cursor position is defined to be at line 1, column 1 of the specified window.

create_window, crwd

SYNTAX AS A COMMAND

```
wdc crwd -io_switch STR {-line N -column N -height N -width N}
```

FUNCTION

creates a new window on the screen with name (and I/O switch) STR. The window is blank when created, and the cursor position is line 1, column 1 of the new window.

delete_chars, dlch

SYNTAX AS A COMMAND

```
wdc dlch -count N {-io_switch STR}
```

FUNCTION

deletes N characters to the right of the current cursor position on the current line. The cursor remains stationary; characters to the right of the deleted characters move to the left to fill the vacated space. You must define the current cursor position.

delete_window, dlwd

SYNTAX AS A COMMAND

```
wdc dlwd -io_switch STR
```

FUNCTION

destroys the specified window. The I/O switch is closed and detached.

get_echoed_chars, gech

SYNTAX AS A COMMAND

```
wdc gech -count N {-io_switch STR}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wdc gech -count N {-io_switch STR}]
```

FUNCTION

reads characters from the terminal until either N characters or a break character is read. All characters except the break are echoed on the screen in the current window. For information on break characters, see the break_table control order in the description of window_io_. You must define the current cursor position. As an active function, two strings are returned: the first contains any nonbreak characters read, and the second contains the break character, if any.

get_first_line, gfl

SYNTAX AS A COMMAND

```
wdc gfl {-io_switch STR}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wdc gfl {-io_switch STR}]
```

FUNCTION

prints/returns the line on the screen where the specified window begins.

get_one_unechoed_char, gouch

SYNTAX AS A COMMAND

```
wdc gouch {-io_switch STR}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wdc gouch {-io_switch STR}]
```

FUNCTION

reads/returns a single unechoed character from the terminal.

get_position, gpos

SYNTAX AS A COMMAND

```
wdc gpos {-io_switch STR}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wdc gpos {-io_switch STR}]
```

FUNCTION

prints the current line and column position of the cursor. As an active function, returns the line and column position as a pair of integers separated by a space.

get_terminal_height, gtmhgt

SYNTAX AS A COMMAND

```
wdc gtmhgt
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wdc gtmhgt]
```

FUNCTION

prints/returns the total number of lines on your terminal.

get_terminal_width, gtmwid

SYNTAX AS A COMMAND

```
wdc gtmwid
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wdc gtmwid]
```

FUNCTION

prints the total number of columns on your terminal. As an active function, returns the total number of columns on your the terminal until either a break character or N characters are read. You must define the current cursor position.

get_unechoed_chars, guch

SYNTAX AS A COMMAND

```
wdc guch -count N {-io_switch STR}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wdc guch -count N {-io_switch STR}]
```

FUNCTION

reads characters from the terminal until either N characters or a break character are read. You must define the current cursor position. As an active string, returns two strings: the first contains any nonbreak characters read, the second contains the break character, if any.

get_window_height, gwdhgt

SYNTAX AS A COMMAND

```
wdc gwdhgt {-io_switch STR}
```

FUNCTION

prints the height of the specified window.

insert_text, itx

SYNTAX AS A COMMAND

```
wdc itx -string STR {-io_switch STR}
```

FUNCTION

displays the text string STR at the current cursor position. If there are any characters to the right of the current position on the current line, they are moved to the right to accommodate the new string. There is no wrap-around feature: if text goes off the screen, it is dropped. The text string STR must contain only printable ASCII characters. Use "io_call put_chars" to display nonprintable characters in a readable form.

invoke

SYNTAX AS A COMMAND

```
wdc invoke {-line_speed N}
```

FUNCTION

activates the video system on your terminal. If you give no line speed, the current one is used. Your terminal must be attached with the tty_ I/O module. If graphics or auditing is in use, remove it before you give this command. The settings of the following tty_ modes are copied when you invoke the video system: vertsp, can, erkl, esc, red, and ctl_char. In addition, if ^pl is set on video system invocation, ^more will be set in the video system. (For more details on modes, see the window_io_ I/O module.) Similarly, the settings of the current erase and kill characters are copied when the video system is invoked.

overwrite_text, otx

SYNTAX AS A COMMAND

```
wdc otx -string STR {-io_switch STR}
```

FUNCTION

displays the text string STR at the current cursor position in the window. If there is any text to the right of the current position in the window, it is overwritten with the string supplied. The text string STR must contain only printable ASCII characters. Use "io_call put_chars" to display nonprintable characters in a readable form.

revoke

SYNTAX AS A COMMAND

```
wdc revoke
```

FUNCTION

removes the video system from your terminal. The standard tty_ attachment is restored. The settings of the following modes are copied when you revoke the video system: vertsp, can, erkl, esc, red, and ctl_char. If ^more is set while in the video system, ^pl mode is set after revoking the video system. (For more details on modes, see the window_io_ I/O module.) Similarly, the settings of the current erase and kill characters are copied when the video system is revoked.

scroll_region, scrgn

SYNTAX AS A COMMAND

```
wdc scrgn -count N {-line N -height N -io_switch STR}
```

FUNCTION

scrolls the specified region N lines as indicated by -count. The specified region is the whole width of the screen. It can be a whole window or part of a window. If -count N is negative, the window is scrolled down; if positive, the window is scrolled up. If lines are scrolled off the screen, they are dropped.

set_position, spos

SYNTAX AS A COMMAND

```
wdc spos -line N -column N {-io_switch STR}
```

FUNCTION

positions the cursor to the specified line and column of the specific window.

set_position_rel, sposrel

SYNTAX AS A COMMAND

```
wdc sposrel -line N -column N {-io_switch STR}
```

FUNCTION

changes the cursor position by N lines and N columns. You must define the current cursor position. You must give one of -line or -column and can use both; whichever control argument you don't supply defaults to zero.

supported_terminal

SYNTAX AS A COMMAND

```
wdc supported_terminal {-ttp terminal_type}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wdc supported_terminal {-ttp terminal_type}]
```

FUNCTION

prints the terminal type that is supported by the video system. If you don't supply a terminal type, the current one is used. As an active function, returns "true" if you can invoke the video system on the given terminal type, "false" otherwise.

sync

SYNTAX AS A COMMAND

```
wdc sync {-io_switch STR}
```

FUNCTION

waits for the last operation performed on the window to be completed. Over certain networks it may not be possible to actually wait for delivery of the characters to the terminals.

video_invoked

SYNTAX AS A COMMAND

```
wdc video_invoked
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wdc video_invoked]
```

FUNCTION

prints the message "The video system has been invoked" if you are already in the video system; otherwise it prints "The video system has not been invoked." As an active function, returns "true" if the video system is in use in your process, "false" othewise.

write_sync_read, wsr

SYNTAX AS A COMMAND

```
wdc wsr -string STR -count N {-io_switch STR}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[wdc wsr -string STR -count N {-io_switch STR}]
```

FUNCTION

displays a prompting string STR at the current cursor position in the window and then reads input typed in response to the prompt. Characters are read unechoed until either N characters or a break character is read. As an active function, prints a prompting string and returns the characters read.

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 you invoke it.

Name: year

SYNTAX AS A COMMAND

```
year {time_string} {-control_arg}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[year {time_string} {-control_arg}]
```

FUNCTION

returns the two-digit number of a year of the century from 00 through 99. The format string to produce this is "^yc".

ARGUMENTS

time_string
: indicates the year about which information is desired. If you supply no time_string, the current year is used. The time string is concatenated to form a single argument even if it contains spaces; you need not quote it. (See Section 1 for a description of valid time_string values.)

CONTROL ARGUMENTS

-zone STR
: STR specifies the zone that is to be used to express the result. (Default: the process default)

NOTES

Use the print_time_defaults command to display the default zone. Use the display_time_info command to display a list of all acceptable zone values.

Name: zero_segments, zsegs

SYNTAX AS A COMMAND

```
zsegs star_names {-control_args}
```

SYNTAX AS AN ACTIVE FUNCTION

```
[zsegs star_names {-control_args}]
```

FUNCTION

returns the entrynames or absolute pathnames of segments with a zero-bit count that match one or more star names.

ARGUMENTS

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

CONTROL ARGUMENTS

-absolute_pathname, -absp
returns absolute pathnames rather than entrynames.

-chase
processes the targets of links when you specify a starname.

-inhibit_error, -ihe
returns false if star_name is an invalid name or if access to tell of an entry's existence is lacking.

-no_chase
does not process the targets of links when you specify a starname. (Default)

-no_inhibit_error, -nihe
signals an error if star_name is an invalid name or if access to tell of an entry's existence is lacking. (Default)

NOTES

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

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

This page intentionally left blank.

SECTION 4

ACCESS TO THE SYSTEM

This section describes the requests interpreted by the answering service. These requests can only be issued from a terminal connected to the answering service; that is, one that has just dialed up or one that has been returned to the answering service after a session terminated with a "logout -hold" command.

For clarity, this section identifies two categories of answering service requests: preaccess and access. The preaccess requests are necessary because certain terminals do not have an answerback. By convention, Multics uses a terminal answerback to identify the particular type of device being used. The device type is used by the system to interpret all input/output. Therefore, for input to be understood by Multics and output understood by the user, these requests must be specified before the access requests. The access requests connect the terminal to a process. This process may exist already (e.g., dial) or be created in response to the request (e.g., login).

Name: access_class, acc

SYNTAX

```
acc
```

FUNCTION

prints the current terminal channel access class on your terminal.

EXAMPLES

```
! acc
  Channel access class: system_low: system_high
```

Name: dial, d

SYNTAX

```
d dial_id {User_id} {-control_args}
```

FUNCTION

connects an additional terminal to an existing process. It requests the answering service to connect and notifies your process of the connection.

ARGUMENTS

dial_id
: is the identifying keyword, supplied by a logged-in process, that uniquely specifies that process that is accepting dial connections.

User_id
: is the Person_id.Project_id of the process that you wish to connect to. This argument is required only if the dial_id is not registered with the system. Registered aliases are allowed for the Person_id and Project_id.

CONTROL ARGUMENTS

-authorization ACCESS_CLASS, -auth ACCESS_CLASS
: specifies the AIM level and category, or categories, of the data that will be transmitted to and from the system in this session. If not provided, the default authorization from the PNT is used. Only can be supplied with -user.

-no_print_off, -npf
: overtypes a string of characters providing a black area for you to type the password. You can only give it with -user. (Default: depends on the terminal type)

-print_off, -pf
suppresses the overtyping of the password. You can only provide it with -user. (Default: depends on the terminal type)

-user Own_person_id, -user Own_user_id
specifies a User_id to give when validating access to the communications channel. Own_person_id is your registered personal identifier; Own_user_id is your Person_id.Project_id. If you supply no Project_id, the default project associated with the Own_person_id is used. Registered aliases are allowed for Person_id and Project_id.

NOTES

When you invoke dial, the answering service searches for a logged-in process accepting dial connections using the dial_id you provided. If not found, the message "Dial line not active." is printed and you can try again, with a different dial_id; if found, a one-line message verifying the connection is printed. All further messages printed on the terminal are from your process.

This request is administratively restricted. The project administrator must register you and your project if you want dialed terminals. The system administrator must register the dial_id if you want dialed terminals without Own_user_id and can restrict your access to a login service communications channel. Give -user with your User_id. This request then asks for your password ensuring its nonvisibility. When your identification and permission to use the channel are verified the dial request is processed.

All arguments must be supplied in the correct order.

If your process terminates or logs out, a message is printed and control of the terminal is returned to the answering service.

Name: echo

SYNTAX

echo

FUNCTION

used to set the terminal into echoplex mode before login.

NOTES

This command is equivalent to:

modes echoplex

Name: enter, e

SYNTAX

```
e {anonymous_name} Project_id {-control_args}

ep {anonymous_name} Project_id {-control_args}
```

FUNCTION

used by anonymous users to gain access to Multics. Either one is actually a request to the answering service to create a process for the anonymous user. Anonymous users who are not to supply a password use the enter (e) request. Anonymous users who are to supply a password use the enterp (ep) request. (See "Notes on Passwords" below.)

CONTROL ARGUMENTS

-arguments STR, -ag STR
: supplies arguments to the process. STR can be one or more arguments. All arguments following -ag on the command line are taken as arguments to the process. Therefore -ag, if present, must be the last control argument to the enter request. The process can determine the number and value of each argument with the login_args active function.

-brief, -bf
: suppresses messages associated with a successful login. If the standard process overseer is being used, the message of the day is not printed.

-force
: logs the user in if at all possible, provided the user has the guaranteed login attribute. Only system users who perform emergency repair functions have the necessary attribute.

-home_dir path, -hd path
: sets the user's home directory to the path specified, if the user's project administrator allows that user specify a home directory.

-modes STR, -mode STR, -md STR
: sets the I/O modes associated with the user's terminal to STR, where the string STR consists of modes acceptable to the tty_ I/O module. (See the tty_ I/O module description in the Subroutines manual for a complete explanation of possible modes.) The STR string is usually a list of modes separated by commas; the STR string must not contain blanks. (See "Examples" below.)

-no_preempt, -np
: refuses to log the user in if login can be achieved only by preempting some other user in the load control group.

-no_print_off, -npf
: causes the system to overtype a string of characters to provide a black area for typing the password.

-no_start_up, -ns
instructs the standard process overseer not to execute the user's start_up.ec segment, if one exists, and if the project administrator allows the user to avoid it.

-no_warning, -nw
suppresses even urgent system warning and emergency messages from the operator, both at login and during the user's session. Use of this argument is recommended only for users who are using a remote computer to simulate a terminal, or are typing out long memoranda, when the process output should not be interrupted by even the most serious messages.

-outer_module p, -om p
attaches the user's terminal via the outer module named p rather than the user's registered outer module, if the user has the privilege of specifying an outer module.

-print_off, -pf
suppresses overtyping for the password. (Default: determined by the terminal type)

-process_overseer path, -po path
sets the user's process overseer to the procedure given by the path specified, if the user's project administrator allows that user to specify a process overseer. If path ends in the characters ",direct", the specified procedure is called directly during process initialization rather than by the init_admin procedure provided by the system. This means that the program specified by path must perform the tasks that would have been performed by the init_admin procedure.

-ring N, -rg N
sets the user's initial ring to be ring N, if this ring number is greater than or equal to the user's registered initial ring and less than the user's registered maximum ring.

-subsystem path, -ss path
creates the user's process using the prelinked subsystem in the directory specified by path. The permission to specify a process overseer, which can be given by the user's project administrator, also governs the use of the -subsystem argument. To override a default subsystem by the project administrator, type -ss "".

-terminal_type STR, -ttp STR
sets the user's terminal type to STR, where STR is any terminal type name defined in the standard terminal type table. (To obtain a list of terminal types, refer to the print_terminal_types command.) This control argument overrides the default terminal type.

NOTES

If neither the -print_off nor -no_print_off control argument is specified at log-in, the system attempts to choose the option most appropriate for the user's terminal type.

If the project administrator does not allow the user to specify the -subsystem, -outer_module, -home_dir, -process_overseer, or -ring control arguments or if the administrator does allow one or more of these control arguments and they are incorrectly specified by the user, a message is printed and the login is refused.

NOTES ON PASSWORDS

The password is a string of one to eight characters. The characters can be any printing character from the ASCII character set except space and semi-colon. The backspace character is also allowed and is counted as a character. The password used for interactive logins cannot be "quit", "help", "HELP", or "?", because these have special meaning to the password processor. Typing a password of "quit" terminates the login attempt. A response of "help", "HELP", or "?" produces an explanatory message, and the request for the password is repeated.

Name: hangup

SYNTAX

hangup

FUNCTION

terminates communication between the terminal and Multics system.
If the communication is via a dial-up phone line, the line is hung up. A user who is unable to log in can issue the hangup request as an alternative to manually hanging up the phone.

Name: hello

SYNTAX

hello

FUNCTION

repeats the greeting message that is printed whenever a terminal is first connected to the system. The request is particularly useful after a 963 or 029 request since the greeting message is then printed in the proper code.

Name: help, HELP

SYNTAX

```
help
```

FUNCTION

provides online assistance for logging in by giving examples of correct login and the phone number of the system administrator to call for more help. Use HELP with those terminals whose keyboards generate only uppercase characters.

Name: login, l

SYNTAX

```
l Person_id{.Project_id} {-control_args}
```

FUNCTION

gives you access to the system. It is a request to the answering service to start your identification procedure and then either create a process for you or connect the terminal to your disconnected process. The command line can be up to 300 characters long.

ARGUMENTS

Person_id
: is your registered personal identifier, which can be replaced by a registered "login alias" if you have one. Aliases, like personal identifiers, are registered by the system administrator and are unique at the site. The login alias is translated into your 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. (Required)

Project_id
: is the identification of your project. If you don't give it, the default project associated with the Person_id is used. (See -change_default_project.)

CONTROL ARGUMENTS

The following is an alphabetized list of control arguments; their description is provided in one of the three functional lists given below.

-arguments	-new_proc
-authorization	-no_preempt
-brief	-no_print_off
-change_default_auth	-no_save_on_disconnect
-change_default_project	-no_start_up
-change_password	-no_warning
-connect	-outer_module
-create	-print_off
-destroy	-process_overseer
-force	-ring
-generate_password	-save_on_disconnect
-home_dir	-subsystem
-list	-terminal_id
-modes	-terminal_type

LIST OF GENERAL CONTROL ARGUMENTS

The following are permitted in any use of the login command:

-brief, -bf
suppresses messages associated with a successful login except the ones indicating that you have incorrectly specified your password and the ones indicating all your login attempts with the same Person_id. If you are using the standard process overseer, the message of the day is not printed.

-change_default_auth, -cda
changes your registered default login authorization to the one specified by -authorization. If you give a valid authorization, the default authorization is changed for subsequent logins and the message "default authorization changed" is printed. If you give -cda without -authorization, an error message is printed.

-change_default_project, -cdp
changes your default project to the Project_id specified on this login request line. The default Project_id is changed for subsequent logins, and the message "default project changed" is printed. If you specify -cdp without a Project_id, an error message is printed.

-change_password, -cpw
changes your password to a new one. The login request asks for the old password before it requests the new one, twice, to verify the spelling. If you don't type it the same way both times, the login and the password change are refused. If the old password is correct, the new one replaces the old for subsequent logins and the message "password changed" is printed. Don't type the new password as part of the control argument. (See "Notes on Passwords" below.)

-generate_password, -gpw
changes your password to a new one, generated for you by the system. The login request asks for the old password first; then, a new password is generated and typed on your terminal. You are asked to retype the new password, to verify having seen it. If you type it correctly, it replaces the old for subsequent logins and the message "password changed" is printed. If you mistype it, the login and password change are refused.

-long, -lg
reverses the effect of -brief, or the brief attribute in the project definition table (see the MAM Project, AK51).

-modes STR, -mode STR, -md STR
sets the I/O modes associated with your terminal to STR, where STR consists of modes acceptable to the tty_ I/O module (see the set_tty command). STR is usually a list of modes separated by commas; it must not contain blanks. (See "Examples.")

-no_print_off, -npf
overtypes a string of characters to provide a black area for you to type the password.

-no_warning, -nw
suppresses even urgent system warning messages and emergency messages from the operator, both at login and during your session. Give this argument when using a remote computer to simulate a terminal or when typing out long memoranda, when the process output should not be interrupted by even serious messages.

-print_off, -pf
suppresses overtyping for the password. (Default: depends on the terminal type)

-terminal_id STR, -tid STR
sets your terminal identification to STR. This control argument is illegal if the site has specified answerback checking.

-terminal_type STR, -ttp STR
sets your terminal type to STR, where STR is any terminal type name defined in the standard terminal type table. This control argument overrides the default terminal type.

-warning
reverses the effect of -no_warning, or the no_warning attribute in the project definition table (see the MAM Project, AK51).

LIST OF CONTROL ARGUMENTS FOR PROCESS CREATION

Use the following when requesting the creation of a new process.

-arguments STR, -ag STR
supplies arguments to the process; STR can be one or more arguments. If you use -arguments, put it last because everything following it on the command line is taken as arguments to the process. The process can determine the number and value of each argument with the login_args active function.

-authorization STR, -auth STR
sets the authorization of the process to that specified by STR; STR is a character string composed of level and category names for the desired authorization, separated by commas. STR cannot contain any embedded blank or tab characters. (The short names for each level and category always contain no blanks or tabs, and can be used whenever the corresponding long names contain blanks or tabs.) STR must represent an authorization that is less than or equal to the maximum authorization of Person_id on the Project_id. If -authorization is omitted, your registered default login authorization is used. (See the Programmer's Reference Manual for more information about process authorizations.)

-force
logs you in, provided you have the guaranteed login attribute. Only system users who perform emergency repair functions have the necessary attribute.

-home_dir path, -hd path
sets your home directory to the path specified if your project administrator allows it.

-no_save_on_disconnect, -nosave
logs your process out instead of saving it if it is disconnected from its login terminal. This control argument is used to override a default of -save_on_disconnect if that default has been set by your project administrator.

-no_preempt, -np
does not log you in if you might preempt somebody in this user's load control group.

-no_start_up, -ns
instructs the standard process overseer not to execute your start_up.ec segment if the project administrator allows it.

-outer_module path, -om path
attaches your terminal via the outer module named path rather than your registered outer module if you are allowed.

-process_overseer path, -po path
sets your process overseer to the procedure given by path if your project administrator allows it. If path ends in the characters ",direct", the specified procedure is called directly during process initialization rather than by the standard system-provided procedure. This means that the program used by path must perform the tasks that would have been performed by the standard procedure. The combined length of the -po and -ss character strings must be less than 64 characters.

-ring N, -rg N
sets your initial ring to N if this ring number is greater than or equal to your registered initial ring and less than your registered maximum ring.

-save_on_disconnect, -save
saves your process if it is disconnected from its login terminal because of a communications line hangup or FNP crash. Your project administrator gives permission to use the process-saving facility and to enable it by default. (See -nosave and the save_on_disconnect and no_save_on_disconnect commands.)

-subsystem path, -ss path
creates your process using the prelinked subsystem in the directory specified by path if your project administrator allows it. To override a default subsystem specified by the project administrator, type -ss "".

LIST OF CONTROL ARGUMENTS FOR DISCONNECTED PROCESSES

Use the following to specify the disposition of disconnected processes (see "Notes on Disconnected Processes" below):

-connect {N}
connects the terminal to your disconnected process. If more than one such process exists, indicate the process number N.

-create
creates a new process without destroying any disconnected ones. This is permitted only if you are allowed to have multiple interactive processes.

-destroy {N}
destroys your disconnected process and logs out. If more than one such process exists, specify the process number N.

-immediate
bypasses termination of the existing process by the trm_ IPS signal (which causes running of finish and epilogue handlers in the existing process) and instead tells the hardcore to destroy the existing process immediately.

-list
lists your disconnected process, its number, the time of the original login, and the ID of the channel and terminal that were last connected to the process.

-new_proc {N}
destroys your disconnected process and creates a new one. If more than one such process exists, give the process number N.

NOTES

Unless you already have one or more processes, login creates a process for you. The load control mechanism is consulted to determine if the creation of your process overloads either the system or your load control group.

If the mechanism allows it, a process is created for you and the terminal is connected to it (i.e., the terminal is placed under that process's control). (See "List of Control Arguments for Process Creation" above.)

You might have a disconnected process because of a phone line hangup or an FNP crash. Then, you can choose among the following alternatives: connecting the terminal to the process; destroying the disconnected process, with or without creating a new one; or logging out without affecting the disconnected process (see "List of Control Arguments for Disconnected Processes" above and "Notes on Disconnected Processes" below).

If you specify neither -pf nor -npf at log-in, the system chooses the option most appropriate for your terminal type.

Several parameters of your process, as noted above, can be controlled by your project administrator; for example, allowing you to override attributes by specifying control arguments on the login line.

If the project administrator does not allow you to use -hd -om, -po, -rg, -save, or -ss or does allow you to give one or more of them and you specify them incorrectly, a message is printed and the login is refused.

NOTES ON PASSWORDS

The login request asks you for a password and ensures either that the password does not appear on your terminal or that it is thoroughly hidden in a string of cover-up characters. The password is a string of one to eight characters, which can be any character from the ASCII character set (including the backspace) except space and semicolon. The password used for interactive logins cannot be "quit", "help", "HELP", or "?" because these have special meaning to the password processor. Typing "quit" terminates the login attempt; "help", "HELP", or "?" produces an explanatory message and repeats the request for the password.

After you type the password the answering service looks up the Person_id, the Project_id, and the password in its tables and verifies that the Person_id and the Project_id are valid, that you are a legal user of the project, and that the password given matches the registered password. If these tests succeed, you are logged in.

NOTES ON DISCONNECTED PROCESSES

If your project administrator allows it, your process can be preserved when it becomes disconnected from its terminal. You can call back any time before the installation-defined maximum inactive time and ask to be reconnected. This feature is controlled by -save and -nosave; your project administrator sets the default.

If your project administrator allows you to have several interactive processes simultaneously, you can have more than one disconnected process. Multiple disconnected processes are numbered consecutively starting with 1, in the order of their login times. Use these process numbers as arguments when referring to one of a set of multiple disconnected processes. The number and login time of each is printed by -list or the "list" request. You can, however, anticipate the number and use it with a control argument. The time listed and sorted on is the time of the original login from which the process is descended; this time is not affected by new_proc or reconnection.

LIST OF REQUESTS FOR DISCONNECTED PROCESSES

If you do not specify on the login line what to do with the disconnected processes, you are told of the disconnected processes and given these choices:

connect {N}
    to connect the terminal to a disconnected process

create
    to create an additional process

destroy {N} {-control_args}
    to destroy a disconnected process and log out

help
    to print a description of these options

list
    to list your disconnected processes

logout {-control_args}
    to log out without affecting any process

new_proc {N} {-control_args}
    to destroy a disconnected process, create a new one with the same attributes, and connect the terminal to it.

When issued from a logged-in but disconnected terminal, the help request explains these options, not how to log in.

LIST OF CONTROL ARGUMENTS FOR DISCONNECTED PROCESS REQUESTS

-hold, -hd
prevents the breaking of the connection between the terminal and the answering service. You can use it only with destroy and logout. (Default)

-immediate
bypasses termination of the existing process by the trm_ IPS signal (which causes running of finish and epilogue handlers in the existing process) and instead tells the hardcore to destroy the existing process immediately. You can use it only with destroy and new_proc.

-no_hold, -nhd
drops the connection. You can use it only with destroy and logout.

EXAMPLES

In the examples below, the user's password is shown even though in most cases the system either prints a string of cover-up characters to hide the password or temporarily turns off the printing mechanism of the user's terminal.

Probably the most common form of the login request is to specify the Person_id and the Project_id and then the password:

```
login GDScarlatti Demo
Password:
mypass
```

To set (or change) the default project to Demo:

```
login GDScarlatti Demo -cdp
Password:
mypass
Default project changed.
```

To set the tabs and crecho I/O modes so the terminal uses tabs rather than spaces where appropriate on output and echoes a carriage return when a linefeed is typed (assuming the user has a default project):

```
login GDScarlatti -modes tabs,crecho
Password:
mypass
```

To change the password from mypass to newpass (assuming the user has a default project):

```
login GDScarlatti -cpw
Password:
mypass
New Password:
newpass
New Password Again:
newpass
Password changed.
```

The following example illustrates a login involving a disconnected process:

```
   login JBrahms.Demo
   Password:
   mypass

You have 1 disconnected process.
JBrahms.Demo logged in 11/16/84  1435.9 est Fri from ROSY terminal "none"
Last login 11/16/84  1435.1 est Fri from ROSY terminal "none"
Please give instructions regarding your disconnected process(es).
Please type list, create, connect, new_proc, destroy, logout, or help.

   list
1)  logged in 11/16/84  1435.1 est Fri over channel a.h001, terminal "none"
Please type list, create, connect, new_proc, destroy, logout, or help.

   connect
Your disconnected process will be connected to this terminal
Wait for QUIT.
QUIT
r 1503:03 .47 12 Level 2
```

Name: logout

SYNTAX

```
logout {-control_args}
```

FUNCTION

terminates your session and ends communication with the Multics system. It is used from a terminal that is logged in but not connected to a process. (See "Notes on Disconnected Processes" under the login request.) It informs the answering service that the user who gave a correct Person_id-password combination is no longer using the terminal.

CONTROL ARGUMENTS

-brief, -bf
: prints neither the logout message nor, if you give -hold, the login message.

-hold, -hd
: terminates your session but not communication with the system: you can immediately log in without redialing.

NOTES

If your site is security conscious, it may have disabled "logout -hold"; in this case if you wish to change authorization, do this:

1. log out
2. verify, using terminal/modem indications, that the terminal has dropped DTR and that the system acknowledged by dropping DSR
3. log in at the new authorization.

This procedure is the only way to guarantee that you are communicating with the answering service and not with a Trojan horse.

DTR and DSR are EIA RS232 control signals that are part of the interface between your terminal and the system.

Name: MAP

SYNTAX

MAP

FUNCTION

tells the system that the user is attempting to gain access from a terminal whose keyboard generates only uppercase characters. This request must be invoked before the access requests (e.g., login) can be successfully issued.

NOTES

Once the request has been issued, the system changes the translation tables used by the terminal control software so that all uppercase alphabetic characters are translated to lowercase. The user still needs to use the special escape conventions to represent the ASCII graphics that are not on the uppercase-only terminal keyboard. Uppercase alphabetic characters also require the escape conventions. After the MAP request is given, the user may log in normally.

This request must be used for 150-, 300-, and 1200-baud terminals if their keyboards can transmit only uppercase characters; for any other terminal type, it is ignored.

EXAMPLES

The following example shows a user invoking the MAP request.

```
! MAP

! LOGIN \JONES \DEMO

  PASSWORD:
! MYPASS
```

Name: modes

SYNTAX

```
modes {mode_string}
```

FUNCTION

sets the terminal modes before login. Accepts the same mode_string as the set_tty -modes control argument. Without arguments, it gives the current modes.

ARGUMENTS

mode_string
is a list of modes to be set.

Name: noecho

SYNTAX

```
noecho
```

FUNCTION

allows you to turn off the echoplex mode which may be on by default. This command is equivalent to the "modes ^echoplex" preaccess command line.

Name: slave

SYNTAX

```
slave {-control_args}
```

FUNCTION

changes the service type of the channel from login to slave for the duration of the connection.

CONTROL ARGUMENTS

-authorization ACCESS_CLASS, -auth ACCESS_CLASS
specifies the AIM level and category, or categories, of the data that will be transmitted to and from the system in this session. If you don't provide it, the default authorization from the PNT is used. You can only supply it with -user.

-no_print_off, -npf
overtypes a string of characters providing a black area for you to type the password. You can only give it with -user. (Default: depends on the terminal type)

-print_off, -pf
suppresses the overtyping of the password. You can only provide it with -user. (Default: depends on the terminal type)

-user Own_person_id, -user Own_user_id
specifies a User_id to give when validating access to the communications channel. Own_person_id is your registered personal identifier; Own_user_id is your Person_id.Project_id. If you supply no Project_id, the default project associated with the Own_person_id is used.

NOTES

The slave command enables a privileged process to request the answering service to assign the channel to it, and then attach it (see the dial_manager_ subroutine for an explanation of the mechanism for requesting channels from the answering service).

This request is administratively restricted. The project administrator must register you and your project if you want dialed terminals. The system administrator must register the dial_id if you want dialed terminals without Own_user_id and can restrict your access to a login service communications channel. Give -user with your User_id. This request then asks for your password ensuring its nonvisibility. When your identification and permission to use the channel are verified the dial request is processed.

Name: terminal_id, tid

SYNTAX

```
tid {STR}
```

FUNCTION

sets your terminal identification to STR. Without arguments, it gives the current terminal_id. This command is illegal if your site has specified answerback checking.

Name: terminal_type, ttp

SYNTAX

```
ttp {terminal_type_name}
```

FUNCTION

sets the terminal type prior to login. Without arguments, it gives the current terminal_type.

ARGUMENTS

terminal_type_name
    is the name of a system-defined terminal type.

MULTICS

COMMANDS AND ACTIVE FUNCTIONS ADDENDUM B

SUBJECT

Additions snd Changes to the Manual

SPECIAL INSTRUCTIONS

This is the second addendum to AG92-06, dated February 1985. Insert the attached pages into the manual according to the collating instructions on the back of this cover. Marginal change indicators (change bars and asterisks) indicate technical changes.

See "Significant Changes" in the Preface.

Note: Insert this cover behind the manual cover to indicate the updating of this document with Addendum B.

SOFTWARE SUPPORTED

Multics Software Release 12.1

ORDER NUMBER

AG92-06B

November 1987

51321
1088
Printed in U.S.A.

Honeywell Bull

COLLATING INSTRUCTIONS

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

Remove	Insert	Remove	Insert
Front cover, blank	Front cover, blank	3-395, 3-396	3-395, 3-396
Title page, Preface	Title page, Preface		3-396.1, blank
iii through xiv	iii through xiv	3-405, 3-406	3-405, 3-406
3-3 through 3-6	3-3 through 3-6	3-431, 3-432	3-431, 3-432
3-14.1, 3-14.2	3-14.1, 3-14.2	3-432.1 through 3-432.18	3-432.1 through 3-432.18
3-14.3, blank	3-14.3, blank	3-539 through 3-542	3-539 through 3-542
3-35, 3-36	3-35, 3-36	3-577, 3-578	3-577, 3-578
	3-36.1, blank	3-673, 3-674	3-673, blank
3-147 through 3-150	3-147, 3-148		3-673.1, 3-674
	3-149, 3-149.1	3-739 through 3-748	3-739, 3-740
	3-149.2, 3-150		3-740.1, 3-740.2
3-153, 3-154	3-153, blank		3-741 through 3-745.1
	3-153.1, 3-154		3-745.2, 3-746
3-157, 3-158	3-157, 3-158		3-747, 3-748
3-165 through 3-168	3-165, 3-166		3-748.1, blank
	3-166.1, 3-166.2	3-791, 3-792	3-791, 3-792
	3-167, blank	3-821 through 3-824	3-821 through 3-824
	3-167.1, 3-168	3-859, 3-860	3-859, 3-860
3-235, 3-236	3-235, 3-236	3-919 through 3-924	3-919, 3-920
3-241, 3-242	3-241, 3-242		3-921, 3-921.1
3-256.1, blank	3-256.1, 3-256.2		3-922, 3-922.1
3-291, 3-292	3-291, 3-292		3-923, 3-924
3-303 through 3-308	3-303 through 3-308	3-937 through 3-940	3-937 through 3-940
3-311, 3-312	3-311, 3-312	4-1 through 4-4	4-1 through 4-4
		i-1 through i-55	i-1 through i-33
		blank, rear cover	blank, rear cover

Honeywell Bull disclaims the implied warranties of merchantability and fitness for a particular purpose and makes no express warranties except as may be stated in its written agreement with and for its customer. In no event is Honeywell Bull liable to anyone for any indirect, special or consequential damages.

The information and specifications in this document are subject to change without notice. Consult your Honeywell Bull Marketing Representative for product or service availability.

11/87

Copyright © Honeywell Bull Inc., 1988

File No.: 1L13

AG92-06B

MULTICS

COMMANDS AND ACTIVE FUNCTIONS ADDENDUM A

SUBJECT

Additions and Changes to the Manual

SPECIAL INSTRUCTIONS

This is the first addendum to AG92-06 dated February 1985.

Insert the attached pages into the manual according to the collating instructions on the back of this cover. Change bars in the margins indicate technical changes; asterisks denote deletions.

Refer to the Preface for "Significant Changes."

Note: Insert this cover behind the manual cover to indicate the manual is updated with Addendum A.

SOFTWARE SUPPORTED

Multics Software Release 12.0

ORDER NUMBER

AG92-06A

November 1986

47143
287
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

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

Remove	Insert
Title page/Preface	Title page/Preface
iii through xiv	iii through xiv
1-1 through 1-4	1-1 through 1-4 1-4.1, blank
2-1 through 2-4	2-1 through 2-4
2-7 through 2-11	2-7 through 2-11
3-1 through 3-14	3-1 through 3-14 3-14.1, 3-14.2 3-14.3, blank
3-27, 3-28	3-27, 3-28
3-33, 3-34	3-33, 3-34 3-34.1, blank
3-89, 3-90	3-89, 3-90
3-97, 3-98	3-97, 3-98 3-98.1, blank
3-105, 3-106	3-105, 3-106 3-106.1, blank
3-117, 3-118	3-117, 3-118 3-118.1, blank
3-123, 3-124	3-123, 3-124 3-124.1, blank

The information and specifications in this document are subject to change without notice. Consult your Honeywell Marketing Representative for product or service availability.

©Honeywell Information Systems Inc., 1986
File No.: 1L13, 1U13

AG92-06A

Remove	Insert
3-129, 3-130	3-129, 3-130
3-149, 1-150	3-149, 1-150 3-150.1, blank
3-153, 3-154	3-153, 3-154 3-154.1, blank
3-171 through 3-174	3-171 through 3-174
3-179, 3-180	3-179, 3-180 3-180.1, blank
3-189 through 3-192	3-189 through 3-192
3-231, 3-232	3-231, 3-232 3-232.1, blank
3-235, 3-236	3-235, 3-236
3-239 through 3-244	3-239 through 3-244 3-244.1, blank
3-249, 3-250	3-249, 3-250 3-250.1, blank
3-253 through 3-256	3-253 through 3-256 3-256.1, blank
3-261, 3-262	3-261, 3-262
3-269 through 3-276	3-269 through 3-276 3-276.1, 3-276.2
3-285, 3-286	3-285, 3-286 3-286.1, blank
3-295 through 3-298	3-295 through 3-298 3-298.1, blank
3-315, 3-316	3-315, 3-316 3-316.1, blank
3-325, 3-326	3-325, 3-326
3-345 through 3-350	3-345 through 3-350 3-350.1 through 3-350.6
3-357 through 3-360	3-357 through 3-360
3-399, 3-400	3-399, 3-400 3-400.1, blank

Remove	Insert
3-405 through 3-408	3-405 through 3-408 3-408.1, blank
3-429 through 3-432	3-429 through 3-432 3-432.1 through 3-432.30
3-439 through 3-444	3-439 through 3-444 3-444.1, blank
3-451 through 3-454	3-451 through 3-454 3-454.1, blank
3-459 through 3-466	3-459 through 3-466 3-466.1, blank
3-477 through 3-482	3-477 through 3-482
3-491, 3-492	3-491, 3-492 3-492.1 through 3-492.4
3-501 through 3-504	3-501 through 3-504 3-504.1, blank
3-511, 3-512	3-511, 3-512 3-512.1, blank
3-527 through 3-530	3-527 through 3-530 3-530.1, blank
3-555 through 3-560	3-555 through 3-560 3-560.1, blank
3-573, 3-574	3-573, 3-574 3-574.1, blank
3-595 through 3-602	3-595 through 3-602
3-605 through 3-616	3-605 through 3-616 3-616.1 through 3-616.10 3-616.11, blank
3-621 through 3-632	3-621 through 3-632 3-632.1 through 3-632.6
3-659, 3-660	3-659, 3-660
3-663, 3-664	3-663, 3-664 3-664.1, blank
3-665 through 3-670	3-665 through 3-670
3-673, 3-674	3-673, 3-674 3-674.1, 3-674.2

Remove	Insert
3-679 through 3-682	3-679 through 3-682 3-682.1, blank
3-689, 3-690	3-689, 3-690
3-705, 3-706	3-705, 3-706
3-731 through 3-734	3-731 through 3-734
3-747, 3-748	3-747, 3-748
3-791, 3-792	3-791, 3-792 3-792.1, blank
3-811, 3-812	3-811, 3-812 3-812.1, blank
3-847, 3-848	3-847, 3-848 3-848.1, blank
3-853 through 3-858	3-853 through 3-858
3-861 through 3-864	3-861 through 3-864 3-864.1, blank
3-869 through 3-876	3-869 through 3-876 3-876.1, 3-876.2
3-883 through 3-886	3-883 through 3-886
3-905, 3-906	3-905, 3-906
3-913, 3-914	3-913, 3-914
3-919 through 3-924	3-919 through 3-924
3-931 through 3-940	3-931 through 3-940 3-940.1 through 3-940.4
3-945 through 3-950	3-945 through 3-950 3-950.1, blank
3-1007 through 3-1012	3-1007 through 3-1012 3-1012.1, blank
3-1013, 3-1014	3-1013, 3-1014
3-1045, 3-1046	3-1045, 3-1046 3-1046.1, blank
3-1049 through 3-1052	3-1049 through 3-1052
3-1057 through 3-1060	3-1057 through 3-1060

Remove	Insert
3-1069, 3-1070	3-1069, 3-1070
3-1077 through 3-1080	3-1077 through 3-1080 3-1080.1, 3-1080.2
3-1095	3-1095
4-7, 4-8	4-7, 4-8
4-11 through 4-14	4-11 through 4-14 4-14.1, blank
4-19, 4-20	4-19
i-1 through i-55	i-1 through i-58

INDEX

D

E

F

G

H

M

N

O

P

Q

T

U

V

W

Y

Z

HONEYWELL BULL
Technical Publications Remarks Form

TITLE: MULTICS COMMANDS AND ACTIVE FUNCTIONS ADDENDUM B

ORDER NO. AG92-06B

DATED NOVEMBER 1987

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. ☐

PLEASE FILL IN COMPLETE ADDRESS BELOW.

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

CUT ALONG LINE

PLEASE FOLD AND TAPE–
NOTE: U.S. Postal Service will not deliver stapled forms

FOLD ALONG LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 39531 WALTHAM, MA

POSTAGE WILL BE PAID BY ADDRESSEE

Honeywell Bull Inc.
200 Smith Street MS486
P.O. Box 9199
Waltham, Massachusetts, 02254-9832

FOLD ALONG LINE

Honeywell Bull

Honeywell Bull

Corporate Headquarters:
3800 West 80th St., Minneapolis, MN 55431
U.S.A.: 200 Smith Street, MS 486, Waltham, MA 02154
Mexico: Hamburgo No. 64, Col. Juarez Delegacion Cuauhtemoc, 06600 Mexico, D.F.
U.K.: Great West Rd., Brentford, Middlesex TW8 9DH, England **Italy**: 32 Via Pirelli, 20124 Milano
Canada: 155 Gordon Baker Road, North York, Ontario M2H 3P9 **New Zealand**: 14/16 Liverpool Street, Auckland 1
Asia: 4/F, Shui on Centre, 6-8 Harbour Rd., Wanchai, Hong Kong **Australia**: 124 Walker Street, North Sydney, N.S.W. 2060

42781, 1088, Printed in U.S.A.

AG92-06