
mv.ENTERPRISE

Online User Reference

Manual, Chapter 4



Release 4.0
August 1997

This edition of the mv. ENTERPRISE manual set is to be used with all mv. ENTERPRISE systems operating under release 4.0 or higher. This electronic document combines the four hard copy volumes into a single file to be viewed with Adobe's Acrobat Reader, 3.01 or later. Content is identical between online and hard copy editions of this manual set.

It is the policy of PICK Systems, Inc. to improve products as new technology, components, software and firmware become available. PICK Systems, therefore, reserves the right to change specifications without prior notice.

Copyright © 1998, 2000 by PICK Systems, Inc.

Irvine, CA 92606

All rights reserved. Printed in U.S.A.

mv. ENTERPRISE is a trademark of PICK Systems, Inc.
MultiValue is a trademark of Spectrum International, Inc.
UNIX is a registered trademark in the USA and other countries, licensed exclusively through X/Open Company, Limited.

SCO and UnixWare are registered trademarks of Santa Cruz Operation, Inc.

AIX is a trademark of International Business Machines Corp.

MP-RAS is a licensed product of NCR Corporation.

PICK is registered trademark of PICK Systems.

All other brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Part Numbers: This online document
combines the hard copy
manuals with these part
numbers:

84-00014A01

84-00014A02

84-00014A03

84-00014A04

Software Release: 4.0

Document Release: AA

The material contained in this document is furnished for customer reference only, and is subject to change without notice. PICK Systems, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. PICK Systems assumes no responsibility for any errors that may appear in this document and makes no commitment to update nor to keep current the information contained in this document.

The techniques described here are proprietary and should be treated accordingly. No part of this document may be reproduced in any form or by any means without the prior written consent of PICK Systems.

Note that PICK Systems appreciates receiving suggestions and comments on its publications. Comments may be sent to the publisher:

PICK Systems, Inc.
1691 Browning
Irvine, CA 9606
Tel (800) 367-7425
E-mail: documentation@picksys.com
ATTN: Manager, Technical Publications

This page intentionally left blank.

Chapter 4 Table of Contents

Editor	4-1
Introduction to the Editor	4-1
Overview of the Editor Operation	4-3
EDIT Verb	4-5
Text and Data Items	4-6
Editor Command Syntax	4-8
Strings	4-8
Colon (:) Delimiter	4-8
Up-arrow Wildcard Character	4-9
Unprintable Characters	4-9
Line-pointer Controlling Commands	4-10
BOTTOM Command	4-10
GOTO Command	4-10
LIST Command	4-10
NEXT Command	4-11
NULL Command	4-11
TOP Command	4-11
UP Command	4-11
WINDOW Command	4-12
String Match Locating Commands	4-14
LOCATE Command	4-14
AGAIN Command	4-14
INPUT Command	4-16
INPUT Command	4-16
Data Inserting Commands	4-19
INSERT Command	4-19
MERGE Command	4-20
MERGE Command Extended Syntax	4-21
MERGE Command Defaults	4-22
Data Deleting Commands	4-23

DELETE Command (Simple)	4-23
DELETE Command (String-Search)	4-23
Data Replacing Commands	4-26
REPLACE Command	4-26
REPLACE ALL Command (String-Search)	4-26
Universal String-Search	4-27
Column Specifications	4-27
Multiple Replacements Within a Line	4-28
Replacement After Multiple-line Replacement	4-29
Multiple Replacements After the MERGE Command	4-29
Creating Null Lines	4-30
Item Manipulating Commands	4-31
EXIT Command	4-31
FILE DELETE Item	4-31
FILE ITEM Command	4-32
FILE SAVE Command	4-33
FLIP BUFFER Command	4-34
Data Formatting Commands	4-36
APPEND LINE Command	4-36
BREAK LINE Command	4-36
HEX OUTPUT Command	4-37
JOIN LINES Command	4-37
SUPPRESS Command	4-38
TAB Command	4-38
ZONE Command	4-39
Assembler Formatting Commands	4-41
ASSEMBLY FORMAT Command	4-41
MACRO EXPANSION Command	4-42
PRESTORE Command	4-43
PRESTORE Command	4-43
PRESTORE CALL Command	4-45
PRESTORE DISPLAY Command	4-46
PRESTORES in Procs	4-47

Miscellaneous Commands	4-49
CANCEL Command	4-49
COLUMNAR POSITIONS Command	4-49
CURRENT LINE Command	4-49
ITEM SIZE Command	4-50
WILDCARD TOGGLE Command	4-50
Editor Messages	4-51
Summary of Editor Commands	4-53

This page intentionally left blank.

Chapter 4: Editor

Introduction to the Editor

The Editor is a Processor that permits online, interactive modification of any item in the database.

The Editor is used to create and/or modify mv. ENTERPRISE BASIC programs, Procs, paragraphs, sentences, phrases, menu items, assembly source, data files, and file dictionaries. The Editor uses the current line concept, that is, at any given time there is a current line that can be listed, altered, deleted, etc. The Editor includes the following features:

- Two variable-length temporary buffers
- Absolute and relative current line positioning
- Line number prompting on input
- Merging of lines from the same or other items
- Character string location and replacement
- Conditional and unconditional line deletion
- Input/Output formatting
- Prestoring of commands

✓NOTE In this chapter, several example sessions are shown to illustrate how certain commands and Editor operations work. Text that is user-supplied is highlighted with a preceding asterisk.

Convention	Description
UPPER CASE	Characters printed in uppercase are required and must appear exactly as shown.
lowercase	Characters or words printed in lowercase are parameters to be supplied by the user (i.e., line number, data, etc.).
{ }	Braces surrounding a parameter indicate that the parameter is optional.
"string"	A string is a sequence of characters delimited by any nonnumeric character (except a blank or a minus sign) that does not appear within the body of the string itself. (See "Editor Command Syntax.")

Table 4-1: Conventions Used in Editor Command Formats.

Overview of the Editor Operation

The Editor uses two data areas (buffers) to edit an item. The item is copied into one buffer from which updates are assembled in the other. An **F** command merges the updates with the item and then toggles the function of the buffers.

The Editor uses two variable-length temporary buffers (buffer 1 and buffer 2) to create or update an item. When the Editor is entered, the item to be edited is copied from the file to buffer 1 (the current buffer). Each line (attribute) of the item is associated with a line number. A current line-pointer points to the current line of the item, and an EOI (End-Of-Item) pointer points to the last line of the item. Editor operations are performed on one line at a time (the current line) in an ascending line number sequence from TOP (line 0) to EOI. As an Editor operation is performed on a line, the modified line and all previous lines are copied to buffer 2 (update buffer).

The editing process continues working on buffer 1. As lines in the item are changed (or lines are inserted or deleted), the Editor builds a new updated version of the item in buffer 2. Thus, updating must continue in an ascending line number sequence until an **F** command is entered. The **F** command merges the updates with the previously existing item, and an automatic resequencing of the item takes place.

The **F** command does not permanently file an item. It completes the copy to the update buffer, causing all lines to be resequenced and the EOI pointer to be repositioned. It then switches (toggles) the function of the buffers, so that buffer 1 becomes the update buffer and buffer 2 becomes the current buffer. Editing then occurs in buffer 2 with new modifications assembled in buffer 1. This toggling of buffers can go on indefinitely until the item is permanently filed away by a file-item (**FI**) or filesave (**FS**) command.

Overview of the Editor Operation

This editing process is examined in Figure 4-1 which shows a four-line item in buffer 1 (current buffer) with the current line-pointer positioned at line 2 in the top half. Two lines (1234 and 567) are then inserted after line 2 (See Figure 4-1, buffer 2, below). When an F command is issued, the buffers are toggled and the situation is depicted in the bottom half, in which buffer 2 becomes the current buffer. Further modifications made to the item are assembled in buffer 1, which now becomes the update buffer.

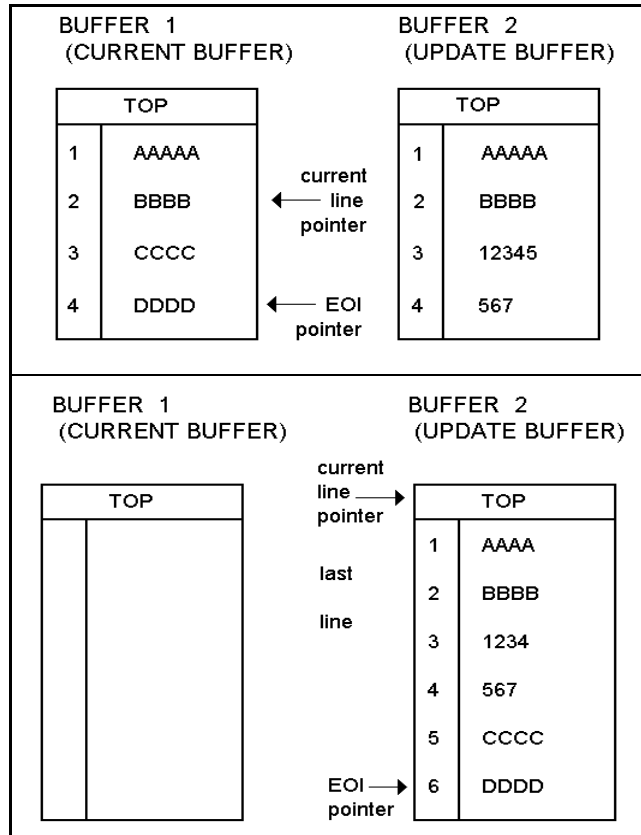


Figure 4-1: Editing example before and after F command.

EDIT Verb

The EDIT verb invokes the Editor. Use the EDIT verb at TCL for entering the Editor to create new items, edit existing ones, or to delete items.

Format

ED{IT} {**DICT**} file-name {item-list} {(options)}

- DICT** The DICT modifier specifies that specified items in the dictionary portion of the specified file are to be edited. If omitted, the specified items in the data section of the specified file are edited.
- file-name** Specifies the name of a file where the items currently exist or are to be stored.
- item-list** Consists of one or more item-IDs separated by blanks. If the item-list is null, or is replaced by an asterisk (*), all items in the file are specified. If multiple item-IDs are specified, the first item specified is edited first.
- options** Options are specified as a single character; multiple options may be separated by commas:
- A** Turns on the Assembler code formatting option (See the topic "ASSEMBLY FORMAT Command.")
 - I** Allows the use of record locking while an item is being edited.
 - M** Turns on macro expansion flag (See the topic "MACRO EXPANSION Command.")
 - P** Sends all system output to the line-printer.
 - S** Suppresses line numbers, and with the AS Assembler verb, also suppresses object-code (See the topic "SUPPRESS Command.")
 - Z** Suppresses TOP and EOI messages.

Description

The EDIT verb invokes the Editor, allowing new file items to be created and existing file items to be displayed, edited, or deleted.

Items in the item-list are edited one at a time. The first item in the list is edited first, followed by the second, and so on. When the first item is exited, the Editor is automatically reinvoked for the next item in the list. This cycle continues until the item list is exhausted, or until the commands EXK, FIK, or FDK are entered to terminate the list. If a select-list is in effect (by using a SELECT, SSELECT, or GLIST), and the item-list is omitted, item-IDs are obtained from the select-list.

✓NOTE Modify D-pointers, using the ACCOUNT-MAINT, FILE-MAINT, or RENAME-FILE utility (See “File Management” and “System Maintenance” chapters.)

When the Editor is entered, this prompt displays:

Top

.

The current line-pointer is set to line zero, and the Editor waits for a command to be entered (the period prompt (.) indicates that an Editor command is to be entered). If the specified item does not already exist on file, the message **New Item** displays above the **Top** message. If multiple item-IDs are specified, the item-ID of the item currently being edited displays.

Text and Data Items

As noted in the “File Structure” topic of the “File Management” chapter, the elements subsidiary to files are items. Structurally, they are made of attributes, while functionally they all are seen by some processor as data. But intuitively, one may consider items to be of two types, text and data.

In a data item the meaning of a data string depends on which attribute it is in. Data items are of two types, attribute defining items (located in dictionaries) and data file items (processed by ACCESS, user exits,

EDIT Verb

or mv.ENTERPRISE BASIC). In both of these types of data items individual lines are referred to as attributes.

A text item is a sequential string that uses the attribute-mark-count only to delimit substrings. Text items consist of lines, which are structurally the same as the attributes of data items, but which have no meaning because of their attribute location. Text items include mv.ENTERPRISE BASIC and Assembly Language programs, Procs, and documents processed by Runoff or a word processor.

The Editor can create, modify, and delete data and text items anywhere in the system, constrained by the user's account privilege level and update lock codes, yet disregarding the type of item or its end use.

The Editor displays attributes as lines, so that the attribute-mark count within the item and the line number displayed by the Editor are the same. Note that attribute zero is the item-ID. The following is an example of the EDIT verb.

```

:ED F1 I1 I2 I3 <----- ED verb with multiple item-IDs
I1             <----- File F1, Item I1 is edited first
Top
.EX           <----- Exit command (exits Editor)
"11" exited.
I2           <----- EDITOR automatically re-entered to edit next item
(I2)
Top
.EX           <----- Exit command
"12" exited.
I3           <----- EDITOR automatically re-entered
New Item     <----- Shows that I3 is a new item
Top
.EX           <----- Exit command
"13" exited.
:            <----- Returns to TCL

```

Editor Command Syntax

This topic describes the syntax of Editor commands.

Editor commands are one or two-letter mnemonics, which must appear as the first nonblank input character. Command parameters follow the command. Blanks may be inserted between parameters for clarity, but embedded blanks in parameters are not permitted.

Editor commands can be entered either in upper or lowercase. This is especially convenient when editing text items, when the terminal may be in the lowercase mode.

Strings

Certain Editor commands use a **string**, which may be defined as a series of characters that is surrounded, or delimited by, a pair of identical, nonnumeric characters that do not appear within the string itself. Lowercase alphabetic characters are not valid as delimiters.

The following are examples of valid strings:

```
/123 MAIN ST./  
.abc 123 DEF.  
AThis test stringA  
" $ 9876.54 "
```

For convenience, the closing delimiter of the string is necessary only if further parameters follow the string specification, or if trailing blanks are to be included as part of the string.

Colon (:) Delimiter

A string is used in Editor commands that specify a search for matching data in the item. The colon (:) is a reserved delimiter. If used, it indicates that a column-dependent correspondence between string and line characters is necessary for a match. The following example attempts to find the matching characters LOOP in columns 1 through 5 of the line.

Editor Command Syntax

:LOOP :

However, this next string attempts to find the matching characters LOOP anywhere within the line.

/LOOP /

Up-arrow Wildcard Character

The up-arrow (^) is a wildcard character used with L(ocate) and R(eplace) Editor commands. A wildcard is a reserved character within the body of the string. It indicates that any character in the corresponding position in the line is acceptable as a match. Note that this feature may be nulled by using the ^ command. For example, the following string attempts to find in the line the matching characters AB, then any character at all, then CD:

/AB^CD/

This feature may be deactivated by using the ^ character alone at the command prompt. Entering it again toggles the feature back on. Accordingly, the Editor outputs:

/^ON

to indicate the wildcard feature is deactivated, or

/^OFF

to indicate the wildcard feature is on.

Unprintable Characters

Unprintable characters include the control characters between X'00' and X'1F', inclusive. The Editor marks control characters by inserting a period (.) where the control character stands in the text line.

However, it does not indicate what the character is. It may then be removed by replacing a unique string, which includes the control character with the string of choice. The control character should be marked with a ^ in the first string in the replacement.

Line-pointer Controlling Commands

These commands control the current line-pointer and list the item being edited. For line-pointer commands, the message TOP is printed if the current line-pointer is set to zero. The message EOI *m* (where *m* is the last line number of the item) is printed if the pointer is set to the End-Of-Item (EOI).

BOTTOM Command

Bottom sets the current line-pointer to EOI.

Format

B

GOTO Command

Positions current line-pointer and lists the line.

Format

Gn or n

Description

Either of these two commands positions the current line-pointer to line *n* and lists the line.

LIST Command

This command lists *n* lines, starting from the current line plus one.

Format

L{n}

Description

If *n* is omitted, only one line is listed. If *n* is greater than or equal to the number of lines from the current line to the EOI, all the lines down to the EOI are listed.

Line-pointer Controlling Commands

If a LIST command is issued when the current line-pointer is at the EOI, the next n lines starting from line 1 are listed. The LIST command positions the current line-pointer at the last line listed.

NEXT Command

Increments current line-pointer.

Format

N{n}

Description

This command increments the current line-pointer by n lines (one line if n is omitted), then lists the new current line.

NULL Command

The NULL command executes by pressing ENTER only.

Format

ENTER

Description

This command is the same as a LIST command where n is omitted. The next line is listed and the current line pointer is advanced one line. This command is included for convenience when stepping through lines in an item.

TOP Command

TOP sets the current line-pointer to zero.

Format

T

UP Command

The UP command decrements the current line-pointer by n lines and then lists the new current line.

Format

U{n}

Description

If *n* is omitted or is zero, the current line is listed.

WINDOW Command

The *W* command displays the specified line and a number of preceding lines. The number of preceding lines is controlled by the page depth specified in the **TERM** settings. The total number of lines to display is (page depth -2) lines.

Format

W{n}

where: *n* is the number of the line to terminate display. If *n* is 0 (zero) list the first (page depth -2) lines.

Description

If there are fewer than (page depth -3) lines preceding *n*, lines 1 through *n* are displayed. If at the top of the item, *W* displays the first (page depth -3) lines.

The following provides an example of line control commands:

Line-pointer Controlling Commands

```
:ED FILE1 ITEM
Top
P <----- P0 --- Prestored command (lists 22 lines)
001 AAAAA ---|
002 BBBBB |
003 CCCCC | <----- Item consists of only 6 lines, so entire item is listed.
004 DDDDD |
005 EEEEE |
006 FFFFF ---|
Eoi 006
. <----- NULL command (since current line-pointer is at EOI,
the 1st line is listed)

Top
001 AAAAA
. <----- NULL command (lists next line)
002 BBBBB
.N2 <----- NEXT command (goes down 2 lines and lists line)
004 DDDDD
.U2 <----- UP command (goes up 2 lines and lists line)
002 BBBBB
.N9 <----- NEXT command; since the item has only six lines,
the EOI is listed

Eoi 006
.L <----- LIST command (since current line-pointer is at EOI,
the 1st line is listed)

Top
001 AAAAA
.L3 <----- LIST command (lists next 3 lines)
002 BBBBB
003 CCCCC
004 DDDDD
.T <----- TOP command (goes to line 0)

Top
P <----- Prestored command (list 22 lines)
001 AAAAA ---|
002 BBBBB |
003 CCCCC | <----- Item consists of only 6 lines, so entire item is listed.
004 DDDDD |
005 EEEEE |
006 FFFFF ---|
Eoi 006
.G5 <----- GOTO command (lists line 5)
005 EEEEE
.B <----- BOTTOM command (goes to EOI)
Eoi 006
.L <----- LIST command (since current line-pointer is at EOI,
the 1st line is listed).

Top
001 AAAAA
.3 <----- GOTO command (lists line 3)
003 CCCCC
.T <----- TOP command (goes to line 0)
Top
```

String Match Locating Commands

The **LOCATE** command searches for characters that match a specified string. The **AGAIN** command repeats the last issued **LOCATE** command.

LOCATE Command

This command searches for characters matching the string.

Format

```
L{n}"string"{p{-q}}
```

Description

The search is restricted to column **p**, or columns **p** through **q**, if specified. If **q < p**, **q = p** is assumed. If the delimiter used in the **LOCATE** command is a colon (:), only matching strings starting in the first column specified (**= p**) are located.

If **n** is not specified, the next occurrence of "string" is located, and that line is listed. At the listed line the current line-pointer is set. If **n** is specified, **n** lines, starting from the current line plus one, are scanned for the occurrence of "string". All lines are listed in which the string is found. Since the current line-pointer is incremented by **n**, it might not be located at the last line listed.

The scan always begins from the current line plus one.

AGAIN Command

The **AGAIN** command repeats the last **LOCATE** command issued.

Format

```
A
```

The following provides an example of the **LOCATE** and **AGAIN** commands.

String Match Locating Commands

```
:ED F1 ABC
Top
.P
001 ABCDEFG ---|
002 12ABCDEFG |
003 BCDEFG | <----- This is what item ABC looks like.
004 ABC |
005 ABCDEFG ---|
Eoi 005
.T
Top
.L"ABC <----- LOCATE command (locates next line with ABC)
001 ABCDEFG <----- Line 1 located
.T
Top
.L5/ABC/ <----- LOCATE command (scans 5 lines and locates lines
                    containing ABC)
001 ABCDEFG ---|
002 12ABCDEFG |
004 ABC | <----- Lines 1, 2, 4, and 5 located
005 ABCDEFG ---|
Eoi 005
.T
Top
.L5<A<3-4 <----- LOCATE command (locates A in columns 3 thru 4)
002 12ABCDEFG <-----Line 2 located
Eoi 005
.L5:ABCD: <----- LOCATE command (locates ABCD column
                    dependent; i.e., must be in columns 1 thru 4)
Top
001 ABCDEFG ---|
005 ABCDEFG ---| <----- Lines 1 and 5 located
Eoi 005
.L5:^^AB: <----- LOCATE command (locates AB in columns 3 thru 4)
Top
002 12ABCDEFG<----- Line 2 located
Eoi 005
.L:^B: <----- LOCATE command (locates next line with B
                    in column 2)
Top
001 ABCDEFG <----- Line 1 located
.A <----- AGAIN command repeats last LOCATE
004 ABC <----- Line 4 located
.A <----- AGAIN command repeats last LOCATE
005 ABCDEFG <----- Line 5 located
```

INPUT Command

The INPUT command is used for data entry. The user may create a new item, or may insert or add lines to an already existing item.

INPUT Command

The INPUT command causes the Editor to enter the input environment.

Format

I

Description

The Editor enters the input environment on the first line. All following lines input by the user are then considered as data input lines to the item, until the user exits the INPUT environment.

- If the INPUT command is issued for a new item not previously edited, the new lines are input to the item starting at line one. The Editor requests data lines by prompting with the line number followed by a plus sign (+). At this prompt, enter the data lines.
- If the INPUT command is issued for an item already containing data, the new lines are inserted following the current line. INPUT is prompted with the current line number, after which the lines are inserted, followed by a plus sign (+). If the current line-pointer is at line zero (Top), input lines are inserted before the first line of the item with a prompt of 000+.
- A null input (a carriage return only) causes the Editor to exit the input environment and await the next Editor command. (If a null line is required in the item, it is necessary to create the line with a fill character and then replace the fill character with a null by using the REPLACE command (see the “Data Replacing Commands” topic). The INSERT command can also be used to insert null lines). If there is an error in the current input line, the user can press ENTER twice, to enter the line and exit the input environment. Then the user needs to execute a REPLACE-string operation to fix the error. Finally, the user should reenter the input

INPUT Command

environment without executing an F command, except on initial input.

The user should note that when the input environment is initially exited for a new item, an automatic F command is executed by the Editor, thus toggling the function of the Editor buffers, listing the newly entered lines.

The following example illustrates using the INPUT command for a new item.

```

:ED AFILE AITEM
New Item      <----- Note that this is a new item.
Top
.I            <----- INPUT command
001+INPUT    ---| <----- Lines being input
002+DATA    ---|
003+        <----- INPUT ended
Top           <----- Automatic F command executed

.L2          <----- LIST command
001 INPUT
002 DATA
Eoi 002
.

```

The following is an example using the INPUT command for a previously edited item.

```

:ED TESTFILE TESTITEM
Top
.P          <-----Prestored command (list 22 lines)
001 LINE 1  ---|
002 LINE 2   | <-----This is what item currently contains.
003 LINE 3  ---|
Eoi 003
.T          <-----TOP command
Top
.I          <-----INPUT command
000+ NEW LINE A<-----New line input
000+      <-----INPUT ended
.G2        <-----GOTO command
002 LINE 2
.I          <-----INPUT command
002+ NEW LINE B<-----New line input
002+      <-----INPUT terminated
.F          <-----F command toggles buffers
Top
.P          <-----Prestored command (list 22 lines).
001 NEW LINE A
002 LINE 1
003 LINE 2
004 NEW LINE B
005 LINE 3
Eoi 005
.

```

Data Inserting Commands

The **INSERT** command inserts one new line. The **MERGE** command inserts one or more lines.

INSERT Command

The **MERGE** command inserts one or more lines by merging lines from the same item, or from another item in the same file or different file.

Format

I data

Description

The user types an **I**, followed by one blank, followed by the data to be inserted. The specified data is inserted as a new line after the current line. Note that the data to be inserted must be separated from the **I** by only one blank. All other blanks are considered to be part of the inserted line.

The **INSERT** command is convenient for either inserting only one line of data (rather than using the **INPUT** command), or for inserting a null line. The latter is done by typing **I** and one space, followed by **ENTER**. One may also insert a string of attribute-marks to generate a string of null lines. This feature is especially useful when entering dictionary items, which use null lines within their structure.

Example

```

:ED ABC ITEM5
Top
.P
001 ABCDEFG ----| <-----This is what ITEM5 looks like.
002 HIJK      ----|
Eoi 002
.G1          <-----GOTO command
001 ABCDEFG
.I 12345     <-----INSERT command
.F          <-----F command (toggles buffers)
Top
.P
001 ABCDEFG ----|
002 12345    | <-----Here is ITEM5 after insertion.
003 HIJK     ----|
Eoi 003
.

```

MERGE Command

For merging from the same file.

Format

ME{n}/item-ID/{m}

Description

This command merges *n* lines (starting from line number *m*) of the item whose item-ID is specified by */item-ID/* into the item being edited. The lines are inserted following the current line. The item specified by */item-ID/* must be in the same file as the item being edited. A value of one is assumed for *n* and *m* if either or both are omitted. If */item-ID/* is null (*//*), lines are merged from the item being edited, as it stands in the current buffer, thus duplicating the specified lines in the item.

If the item from which lines are to be merged is not on file, the message Not on file displays.

Data Inserting Commands

Example

```

:ED FILE1 ITEM1
Top
.P
001 11111    ----|
002 22222    | <-----This is what ITEM1 looks like.
003 33333    ----|
Eoi 003
.EX          <-----EXIT command (exits Editor)
'ITEM1' exited.
:ED FILE1 ITEM2
Top
.P
001 AAAAA    ----|
002 BBBBB    | <-----This is what ITEM2 looks like.
003 CCCCC    ----|
Eoi 003
.G2          <-----GOTO command
002 BBBBB
.ME2"ITEM1"1 <-----Merge 2 lines from ITEM1 starting at line 1.
.F          <-----F command (toggles buffers)
Top
.P
001 AAAAA    ----|
002 BBBBB    |
003 11111    | <----- Here is ITEM2 after the 2 lines from ITEM1
                                are merged.
004 22222    |
005 CCCCC    ----|
Eoi 005
.

```

MERGE Command Extended Syntax

For merging from other files, the MERGE command requires an extended syntax. The extended syntax requires the use of the delimiters (“and”) instead of the / delimiter used above. Thus they become reserved when using the merge command in the sense that the colon (:) is reserved when using the LOCATE, REPLACE, and DELETE commands. In this case there is the additional peculiarity that (“and”) are not the same character, though any character may

normally be used as a delimiter, as long as all the delimiters in the same string are identical.

To merge from other files use the format:

ME{n}{**DICT** file-name {item-name}}{m}

The use of **DICT** means that the merged item comes from the dictionary of file name instead of the data section. If no item-ID is specified, the processor defaults to the item-ID of the item being currently edited. This is useful to place a copy of an item into a test file and edit it quickly, or to ensure that the item is not accidentally filed over the old copy. Combined with the **PRESTORE** command structure and the global **REPLACE** command, some very powerful editing functions can be done easily and very quickly.

MERGE Command Defaults

There are other defaults that apply to the **MERGE** command, which are carried over into this extended form:

ME{n}{**DICT** file-name {item-name}}

This does the same as above, except that the starting line number defaults to line 1 in the merge source item.

ME{**DICT** file-name {item-name}}{m}

This does the same as above, except that the starting line number is the only line that is merged into the destination item. As such, the line may then be modified by using the **REPLACE** command (see above).

The defaults all apply to the normal **MERGE** statement, leading to the minimal form 'ME/', which inserts the first line of the item currently being edited into the current location in the item. This is useful if you want to put a given line in several places in an item.

Data Deleting Commands

The DELETE command deletes one or more lines from the item.

DELETE Command (Simple)

This command deletes *n* lines (one if *n* is omitted), starting from the current line.

Format

DE{*n*}

Description

The current line pointer is set to the line after the deletion, allowing further Editor command sequences.

DELETE Command (String-Search)

The complex form of the DELETE command searches for characters matching the specified string (See “Editor Command Syntax”).

Format

DE{*n*}“string”{*p*{-*q*}}

Description

If *n* is not specified, *n* defaults to 1. If *n* is specified, *n* lines, starting from the current line, are scanned for the occurrence of **string**. All lines are then deleted in which the string is found. Deleted lines are listed. The current line-pointer is set to the line after the span of the DELETE command (or *n* lines).

The search for the specified string is column-dependent if the delimiter used in the string is a colon, or if parameters *p*, or *p* and *q* are used.

- If the colon is used, the Editor defaults to column 1 for the string match, regardless of any *p* or *q* parameters.

Data Deleting Commands

- If `p` is used by itself, the search starts in column `p` and continues scanning the remaining line for a match.
- If `p` and `q` are used, the scan matches all strings that fall inclusively. If `q < p`, then `q=p` is assumed.
- The user should note that the scan always begins from the current line.

This is similar to the simple `DELETE` command, which starts with the current line and continues for the next `n-1` lines.

The following provides an example of the `DELETE` commands.

Data Deleting Commands

```
.ED TEST ITEM.1
Top
.P
001 123XYZ    ---|
002 AAAAAAA  |
003 XYZ123   | <----- This is what item ITEM.1 looks like.
004 ABABABAB |
005 12345    |
006 AA       ---|
Eoi 006
.G5          <-----GOTO command
005 12345
.DE2        <-----DELETE command (deletes 2 lines)
Eoi 006
.F          <-----F command (toggles buffers)
Top
.P
001 123XYZ    ---|
002 AAAAAAA  | <----- Here is item ITEM.1 after lines 5 and 6 are deleted.
003 XYZ123   |
004 ABABABAB ---|
Eoi 004
.T
Top
.DE99/123   <-----DELETE command (deletes lines containing 123)
001 123XYZ   ---| <----- Deleted lines are listed
003 XYZ123   ---|
Eoi 004
.F
Top
.P
001 AAAAAAA  ---| <----- Here is item ITEM.1 after deletion.
002 ABABABAB ---|
Eoi 002
.DE2:AB     <-----DELETE command (deletes lines with B
                in column 2)
002 ABABABAB<----- Deleted line is listed
Eoi 002
.F
Top
.P
001 AAAAAAA  <-----Here is item ITEM.1 after deletion.
Eoi 001
.
```

Data Replacing Commands

The REPLACE command replaces from one character to several lines. It also executes several replacements of a single line. The U option replaces all copies of a string within a line with a specified replacement string.

REPLACE Command

The simple form enters the input environment (see the “INPUT Command”).

Format

R{n}

Description

Input is requested for data to replace n lines (one if n is omitted), starting from the current line. The input environment is exited when entering data for a specified number of lines, or when entering a null line (i.e., a carriage return only). In the latter case, the remainder of the lines (including the line that received the null input) remain unchanged. The current line-pointer points to the next line in the current buffer to be edited.

REPLACE ALL Command (String-Search)

This string-search form of the REPLACE command searches for characters matching string 1 (See “Editor Command Syntax.”)

Format

R{U}{n}/string 1/string 2/{p{-q}}

Description

If n is not specified, only the current line is scanned for string 1. If string 1 is located, it is replaced by string 2. If n is specified, n lines that include the current line are scanned. The first occurrence of

Data Replacing Commands

string 1 in each line is replaced by string 2. Lines that are changed are listed in their updated form.

Universal String-Search

The universal string search form of the REPLACE command is indicated by simply using the form RU, as noted by the {U} in the above string-search format. This universal string-search form of the REPLACE command replaces all cases of string 1 with string 2 in the line or lines specified. The U option allows multiple-line replacements using the form RUn for the form Rn, but is otherwise the same as the R format.

Column Specifications

As with DE, if the delimiter is a colon (:), the column specification defaults to column 1, regardless of any p or q parameters. If only p is used, the scan begins in column p and continues for the rest of the line until a match is found. If the RU form is used, the scan continues searching for all string matches after column p. If p and q are present, a match is made if string 1 falls inclusively between columns p and q. If $q > p$, then $q = p$ is assumed. Only one delimiter separates string 1 and string 2 in the complex form of this command. The third delimiter may be left out if the column specification is not needed. Any nonnumeric character not in either string 1 or string 2 may be used as the delimiter.

The protocols above are identical for the string locate form and the form of DELETE that deletes lines containing a given string.

Example

```

:ED F1 ABC
Top
.P
001 ABCDEF  ---|
002 ABCDEF  | <----- This is what item ABC looks like.
003 ABCDEF  ---|
Eoi 003
.T
Top
.R2          <----- REPLACE command (replaces 2 lines)
001 123ABC  ---| <----- Replacement lines being input
002 XXXXXAB ---|
.F          <----- F command (toggles buffers)
Top
.P
001 123ABC  ---|
002 XXXXXAB | <----- Here is item ABC after replacement.
003 ABCDEF  ---|
Eoi 003
.T
Top
.R3/AB/HHH/ <----- REPLACE command (replaces AB with HHH)
001 123HHHC ---|
002 XXXXXHHH | <----- The 3 lines in which replacement took place
                    are listed.
003 HHHCDEF ---|
Eoi 003
.F
Top
.R3/HHH/S/1-3<----- REPLACE command (replaces HHH in
                    columns 1 thru 3 with S).
003 SCDEF   <----- Line in which replacement took place is listed.
Eoi 003

```

Multiple Replacements Within a Line

Multiple string replacements in a single line are possible without executing an F command if the preceding update instruction was an INPUT command or a REPLACE-string command. The resulting form displays after each replacement, with the current line-pointer remaining on the last line to be edited. Re-listing the modified line before an F command displays the current form rather than the modified form.

Data Replacing Commands

The intent of multiple replacements within a line is to minimize typing and buffer switching (the **F** command). If there are several elements of a line that you want to change, you may change them one at a time, using the **R** command for each, without using the **F** command in between. On each use of the **R** command in this case, the command operates on the result of the last command. Only the first use of the **R** command operates on the original line. This means that if the **X** command is used, you move back to the original line, rather than the line as it existed before the last use of the **R** command, because the last copy is not saved.

If the replacement was a full-line replacement of the form **R**, **ENTER**, followed by the prompt, the text, and a final **ENTER**, the line may not be modified by a string replace until the buffers are exchanged with the **F** command. The premise is that the **X** command can be used, followed by another replace. If this is not satisfactory, the sequence **DE** followed by **| text** has the same result, replacing within the inserted line.

Replacement After Multiple-line Replacement

You may replace text in the last line of an **Rn** group using another **R** command without first flipping the buffers (the **F** command) in the same way you can modify text after a single-line replacement command. It is not possible to access lines other than the last without using either the **F** command or the **X** command, which cancels the **Rn** replace command.

Multiple Replacements After the MERGE Command

It is possible to merge one or more lines of text into the current location in the text, modifying only the last merged line by using the multiple replace facility. Lines other than the last cannot be modified for the reasons noted above. It is possible to do a lot of text manipulation very quickly using the **MERGE**, **DELETE**, and **REPLACE** commands.

Creating Null Lines

The REPLACE command also creates null lines. First, use the INPUT command to create lines, each containing a fill character (such as a .). Then, prior to filing the item, replace each fill character with a null by using a REPLACE command (such as R99 /. /).

Consider the following line:

084 the difference between the beginning and ending

To replace the 2nd the with any, first create a helpful column guide by typing **.C**

1	2	3	4	5
1	2	3	4	5

Then enter either of the following two commands:

.R/the/any/24

.R/the/any/24-26 (a column range example)

The above commands both yield the following:

084 the difference between any beginning and ending

Further unrelated examples of the REPLACE command column specifications:

.R5/XYZ/123/15 Replaces first occurrence of string XYZ after column 15 for the next 5 lines.

.RU7/XX/77/20-50 Replaces all occurrences of XX between columns 20 to 50 for the next 7 lines.

Item Manipulating Commands

Editor commands are provided for merging updates into the item, filing the item, deleting the item, or exiting an item.

EXIT Command

The EXIT command ends the Editor session and returns control to TCL.

Format

EX{K}

Description

The item being edited is not updated to the disk-file. Instead, the terminal displays the message item-ID exited.

Any of the above commands ordinarily return control to TCL. However, when multiple items are specified in the ED verb at TCL, the commands instead return control to the Editor so it can edit the next specified item.

The EXK (or EXT) command allows you to exit from such a situation, sending the editing process to TCL or to the Proc that called the Editor.

If an item changes during the editing session, the following message displays before the item is exited:

Item changed; exit it? (Y/N=CR)

FILE DELETE Item

The FILE DELETE command deletes the item from the disk-file and returns control to TCL.

Format

FD{K}

Description

When the item is deleted, the terminal displays the message item-ID deleted. You cannot FD any item other than the one you are currently editing. The K option returns control to TCL or to a calling Proc.

To prevent accidentally deleting an item during the editing session, the following message displays before an item is deleted:

Are you sure you wish to delete it? (Y/N=CR)

Deleting a large number of items can be accomplished easier with the DELETE verb, or by using a prestored command. The DELETE verb is faster.

Format

DELETE {**DICT**} file-name item-list

CAUTION! Do not use DELETE-FILE to delete an item!

FILE ITEM Command

The FILE ITEM command updates the edited item to the disk-file and returns control to TCL.

Format

FI{**K**}

FI{**K**}{**O**} item-name

FI{**K**}{**O**}{(**DICT**)file-name {item-name}}

Description

When the item is filed, the terminal displays the message item-ID filed.

You may file the item currently being edited back onto itself, to a different item in the current file, or to a different item in a different file.

Item Manipulating Commands

✓NOTE The delimiter (space or left parenthesis) must immediately follow the FI. Use a blank as a delimiter when only the item name is specified. The default is the currently edited file. Any item-IDs with embedded blanks may be enclosed in parenthesis. The DICT or file name, if present must immediately follow the left parenthesis, (no blanks). The edited item is updated to the appointed file. Control is returned to TCL unless a selected list is in effect, in which case the next item is entered. The K option cancels any selected list in effect and returns control to TCL or a calling Proc.

If the FI command specifies a file name or item name, the Editor checks that the specified item does not already exist in the file. This is to prevent the accidental destruction of the item. The O option causes the Editor to overwrite the old item in the file with the current edited item.

FILE SAVE Command

The FILE SAVE command updates the edited item to the disk-file and returns control to the Editor.

Format

FS

FS{O} item-name

FS{O}({DICT})file-name {item-name}

Description

The current line-pointer is set to zero. You may file the item currently being edited back onto itself, or to either a different item in the current file, or to a different name in a different file, by using the extended syntax forms of the FS command. The FS command updates to the appointed file a copy of the item being edited, then returns control to the Editor. With the exception that the Editor

retains control and you may continue editing the item, the FS command functions like the FI command.

FLIP BUFFER Command

The F command toggles the function of the Editor buffers.

Format

F

Description

Updates are merged with the previously existing item, setting the current line-pointer to zero.

The following provides an example of item manipulating commands.

Item Manipulating Commands

```

:ED AFILE ABC
Top
.P
001 AAAAAAAAAA ----| <----- This is what item ABC looks like.
002 12121212 ----|
Eoi 002
.DE <----- DELETE command (deletes line 2)
Eoi 002
.F <----- F command (toggles buffers)
Top
.P
001 AAAAAAAAAA <-----Here is item ABC after deletion.
Eoi 001
.EX
Item changed; exit it? (Y/N=CR)
Y <----- EXIT command (returns control to TCL but does not
file updated item).

'ABC' exited.

:ED AFILE ABC
Top
.P
001 AAAAAAAAAA ----| <----- Item ABC still contains 2 lines since
002 12121212 ----| EX command above did not file updated item.
Eoi 002
.DE <----- DELETE command (deletes line 2)
Eoi 002
.FI <----- FILE ITEM command (files item and returns
control to TCL).

'ABC' filed.

:ED AFILE ABC
Top
.P
001 AAAAAAAAAA <-----Here is item ABC (note that line 2 is now
permanently deleted).

Eoi 001
.FS <----- FILE SAVE command (files item and returns
control to Editor).
Top
.FD <----- FILE DELETE command (deletes item and
returns control to TCL).

Are you sure you wish to delete it? (Y/N=CR)
Y
'ABC' deleted
: <----- TCL verb prompt

```

Data Formatting Commands

The Editor formatting commands help the user arrange and display data for easy readability.

APPEND LINE Command

The APPEND LINE command appends a string to the end of one or more lines.

Format

AL{n}/string/

Description

The AL command appends a string to the end of one or more lines. The specified string is appended to n lines, beginning at the current line. If n is omitted, the string is appended to the current line only.

Example

```
001 ABC
.AL/DEF/
001 ABCDEF
```

BREAK LINE Command

The BL command divides a line in two, using a specified string as the break point.

Format

BL{n}/string/

Description

Each line of n lines, beginning at the current line, is searched for the string and divided if the string is found. If n is omitted, only the first line is searched.

Data Formatting Commands

Example

```
001 X = 1
002 Y = 0
003 IF X THEN Y
.BL/THEN/
.T
.P
001 X = 1
002 Y = 0
003 IF X THEN
004 Y
```

HEX OUTPUT Command

The HX command toggles on and off the hexadecimal output switch.

Format

HX

Description

When the switch is on, all characters are presented in hexadecimal format. When the switch is off (the default mode), characters are presented in ASCII format.

Example

```
001 ABC
002 XYZ
.HX
Hex Print ON
.P
001 414243
002 58595A
```

JOIN LINES Command

The JL command concatenates the current line to the preceding one, separating the two with a space.

Format**JL****Example**

```

001 ABC
002 DEF
003 GHI
.JL
002 DEF GHI

```

SUPPRESS Command

The **SUPPRESS** command suppresses Editor line numbers and is an alternate-action toggle switch.

Format**S**

The **S** command suppresses Editor line numbers and is an alternate-action toggle switch. The Editor accordingly responds with Suppress ON or Suppress OFF.

The **S** command takes on an additional feature when it is used with Assembly Language programs and the **AS** command (standard assembly listing format). If the **AS** command is in effect (**AS-ON**), the **S** command suppresses the object code. With **AS** disabled (**AS-OFF**), the **S** command suppresses line numbers as with a non-Assembler data item.

The suppress feature may also be enabled by using the **S** option with the **ED** verb.

TAB Command

Tabbing is invoked whenever the Editor is in the input environment and a **CTRL+I**, or **TAB**, is pressed.

Format**TB n,n,n, ...**

Data Formatting Commands

Description

Each *n* is a **TAB** setting, with up to 15 allowable settings (in ascending order) separated by commas.

TAB outputs a series of blanks, thus moving the cursor (or line-pointer) to the next specified tab stop. Tabs set by the Editor are the same as those set by the **TAB** verb.

ZONE Command

This command sets print column limits for listing output through the **LIST** command.

Format

Z{*p*{-*q*}

Description

This command sets print column limits for listing output through the **L** (**LIST**) command (i.e., only column positions *p* through *q* of each line are listed). If *p* and *q* are omitted, the zone is reset so that the entire line is listed on output. If *q*>*p*, then *q*=*p* is assumed. Setting a zone does not affect the search for a string in the **LOCATE**, **DELETE** or **REPLACE** commands.

The following example utilizes data formatting commands.

```

:ED FN5 XX
New Item <----- This is a new item.
Top
.TB 9,18 <----- TAB command (sets 2 tab stops).
.I <----- INPUT command.
001 ABC
002 ABCD EF<----- Lines being input. For line 2, CTRL+I (which does
not print) is entered after ABCD, causing the Editor
to tab over to the 1st stop.

003 123456789
004
Top
.ZZ-3 <----- ZONE commands (limits listing output to
columns 2 thru 3).

.P
001 BC ----|
002 BC | <----- Only columns 2 thru 3 are listed.
003 23 ----|
Eoi 003
.T
Top
.Z <----- ZONE command (restores full line). SUPPRESS
.S <----- command (suppresses line numbers).
Suppress ON
.P
ABC ----|
ABCD EF | <----- Line numbers are suppressed.
123456789 ----|
Eoi 003
.S <----- SUPPRESS command (restores line numbers.)
Suppress OFF
.P
Top
001 ABC ----|
002 ABCD EF | <----- Line numbers are listed.
003 123456789----|
Eoi 003
.

```

Assembler Formatting Commands

The Assembler formatting commands are used when writing and modifying Assembly Language code.

ASSEMBLY FORMAT Command

The ASSEMBLY FORMAT command formats Assembler source code in the standard Assembly Language listing format.

Format

AS

Description

The AS command acts as an alternate-action toggle switch to format Assembler program lines in Assembly Language format, or to revert to unformatted form.

The Editor responds with the message AS ON or AS OFF, depending on the previous state. This mode may also be turned on when entering the Editor by using the A option on the ED verb. Assembly code source programs contain the assembled object code and macro expansions along with the source text. If displayed in normal, unformatted form, a line might look like

```
007 L00P STORE D1 SAVE ACCUMULATOR\01B A00499
```

If the AS mode is set on, the same line displays as

```
007 01B A00499  LOOP STORE D1  SAVE ACCUMULATOR
...object code...  ...source code...  ...comment field...
```

This display format does not affect the search columns in LOCATE, DELETE, or REPLACE commands, which use the internal (unformatted) form. When the AS mode is on, the Editor S (SUPPRESS) command suppresses object code, not line numbers.

MACRO EXPANSION Command

A macro expansion is a line of code that breaks down and is defined by one or more lower machine level instructions.

Format

M

In the AS mode the M command expands macros. Normally, the macro expansion is off. Using the M command causes the Editor to respond with the message Macro ON or Macro OFF, depending on the previous state. Additional Assembler verbs are discussed in the *Sequoia Pick Assembly Language Manual*.

PRESTORE Command

PRESTORE Command

The PRESTORE facility stores a string of commands into one of ten buffers.

PRESTORE Command

Once commands are prestored, they can be executed or displayed, and are only available for the current editing session.

Format

P{n} {command-string}

Parameter(s)

n	Buffer number 0-9. If n is not specified, buffer 0 is used.
command-string	Sequence of commands assigned to the specified buffer. If command-string is not specified, the commands stored in the specified buffer are executed. Any valid Editor command can be used, including other prestore names.

Description

More than one command can be included in the command string separated by value marks. A value mark can be input into the command string by entering CTRL+] as appropriate. For example, entering the following sequence stores a string of three commands in buffer 6:

P6 T CTRL+] R/*/*COMMENT CTRL+] FI

Entering the string above stores the three commands T, R/*/*COMMENT, and FI separated by value marks into buffer 6. Entering P6 causes the three commands to be executed. As an alternative, the ESC key can be used between commands instead of the CTRL+] sequence.

Each prestored command buffer is 100 bytes in length. If a command sequence is longer than 100 bytes, the command string will automatically overflow into the next sequentially higher buffer. For example, if the command string to be stored in P1 is 150 bytes in length, the command string will overflow into P2.

Upon entering the Editor, buffer 0 is assigned the L (list) command to list the number of lines as controlled by the page depth set in the current TERM settings. The actual number of lines listed is (page depth - 2 lines). Buffers 1 through 9 are null until commands are explicitly stored into them. Attempting to execute commands from a null buffer causes the system to respond with the message String?.

Prestored commands are only available for the current editing session. They persist from item to item when editing from an explicit item-list, a select-list, or an entire file. Any prestored command can be overwritten by executing another command to store commands into the same buffer.

Example

P5 L4

The command above stores the command L4 (list four lines) in buffer 5. To execute the commands in buffer 5, type the following at the Editor prompt:

P5

The command above causes any commands stored in buffer 5 to be executed. From our example, the P5 command causes the next four lines to be displayed.

Command	Description
---------	-------------

Table 4-2: PRESTORE Command.

PRESTORE Command

P	Executes commands stored in buffer 0. Upon entering the Editor, the system always loads a list command into buffer 0. The default number of lines to list is controlled by the page depth specified in the TERM settings. The number of lines to list is (page depth - 2 lines).
P0 L10	Stores the command L10 (list 10 lines) into buffer 0. Once stored in buffer 0, entering P or P0 lists the next 10 lines.
P1 R99/DOG/CAT]FI	Stores a string made up of two commands into buffer 1. The first command replaces the string DOG with CAT in the next 99 lines. The second command files the item. Note that the two commands are separated by a value mark. Once this string is stored in buffer 1, entering P1 executes both commands.

Table 4-2: PRESTORE Command.

PRESTORE CALL Command

A prestored command can be run repetitively by letting it call itself. A PRESTORE command that calls itself ends only when it runs out of items to process. Therefore, a PRESTORE that calls itself needs an EX, FI, or FD in the command string to force item iteration. If the command string lacks such an item iteration command, the prestored command string will loop indefinitely in the current item until BREAK is pressed.

The primary use for having a PRESTORE command call itself is to manipulate many items in the same way. This is particularly useful for searching for specified strings in files and replacing them as

PRESTORE Command

necessary. Refer to Table 4-3 below, which searches the file BP for the string LEDGER.

Command	Description
ED BP *	Edit the file.
ITEMNAME	The first item.
Top	Standard mark.
.P1 L500/LEDGER]EX]P1	Define the search.
.P1	Starts the run. At this point, the Editor exhibits all lines in the current item with the requested string, and then displays.
Eoi nnn	Number of lines in item.
'ITEMNAME' exited.	Name of the item exited.
NEXT ITEMNAME	The name of the next item.
Top	The top mark.
	All lines with the string, if any, and so on, until the list is exhausted, at which time the process returns to TCL.

Table 4-3: A PRESTORE Command Calling Itself.

Refer to Table 4-3. The operation performed could have been executed to the printer by appending the P option to the EDIT verb. Thus, all information usually displayed on the terminal would be sent to the printer.

PRESTORE DISPLAY Command

Use the PRESTORE DISPLAY (PD) command to display all currently initialized PRESTORE commands.

Format

PD

PRESTORE Command

Description

Entering the command `PD` displays all prestored commands for the current session. Using the commands stored in the example above, entering `PD` displays:

```
P0 L22
P1 L500/LEDGER[EX[P1
```

The `L22` command displayed for `P0` assumes that the page depth for the current session is set to 24 lines (as set by the `TERM` verb). The `L500/LEDGER[EX[P1` is the command string defined for buffer 1.

Notice that the value marks inserted between the commands in buffer 1 are displayed as left brackets.

PRESTORES in Procs

It is possible to create `PRESTOREd` command strings in Procs in the same manner that instructions are sent to the various processors from a Proc.

<code>PQ</code>	The Proc definition.
<code>HED BP *</code>	The verb.
<code>STON</code>	Turns the stack on.
<code>HP1 L500/LEDGER<</code>	Specifies the prestore.
<code>HP1<</code>	Executes the prestore.
<code>P</code>	Executes the verb.

Table 4-4: Defining a PRESTORE in a Proc.

This table assumes that a list is in existence. Verb activation may include an explicit item-list or specify the entire file using the conventional asterisk or null item-list.

Upon entry to the first item from the `ED` verb, the `PRESTORE` is automatically set up and available for use. All ten `PRESTOREs` may be initialized this way, allowing the development of powerful customized Editor commands.

The following provides an example of the S, CURRENT LINE, and PRESTORE commands.

```

:ED CARS TEN
Top
.L99
001 A1234    ----|
002 C1234    | <----- This is what item TEN looks like
003 XXXXX1234 |
004 ABCDE1234----|
Eoi 004
.G1
001 A1234
.DE          <----- DELETE command (deletes line 1)
.X          <----- X command (cancels previous DELETE command)
L 001       <----- Message indicates that update on line 1 is cancelled
.F
Top
.L99
001 A1234    <----- Line 1 is not deleted
002 C1234
003 XXXXX1234
004 ABCDE1234
Eoi 004
.?          <----- CURRENT LINE command
CARS TEN L004 <----- Current line is line 4
.P DE3"1234"6-9<----- PRESTORE command (prestores DELETE
                        command)
.T
Top
.P          <----- PRESTORE CALL (calls DELETE into effect)
003 XXXXX1234 <----- Line 3 deleted
.F
Top
.P          <----- PRESTORE CALL command
003 ABCDE1234<----- New line 3 deleted
Eoi 003
.F
Top
.L99
001 A1234    ----| <----- Here is item TEN after deletions
002 C1234    ----|
Eoi 002

```

Miscellaneous Commands

These miscellaneous Editor commands provide helpful features when editing items.

CANCEL Command

The CANCEL command deletes the effect of the last INPUT, INSERT, DELETE, or REPLACE command issued. This is useful if one of these commands is erroneously entered.

Format

X{F}

Description

When the effect of the update command is deleted, the message **L n** is printed (where: **n** is the line number of the line whose update is deleted). The X command does not work after multiple string replacements within a single line.

The XF command reverses the effect of all updates executed since the last buffer exchange (F command).

COLUMNAR POSITIONS Command

This C command displays a list of column numbers so that the user can readily determine the columnar position of data in a line. This is particularly helpful when editing fixed-field data, or Runoff documentation.

Format

C

CURRENT LINE Command

When the CURRENT LINE command ? is entered, the Editor responds with the file name, the item-ID, and the current line number of the item being edited.

Format

?

ITEM SIZE Command

The ITEM SIZE command outputs the size of the currently edited item.

Format

S?

WILDCARD TOGGLE Command

An alternate-action toggle switch for the special effect character ^.

Format

^

Description

The ^ command is an alternate-action toggle switch to turn off or on the special effect of the ^ character within a string.

The Editor responds with the message:

/\ ON

to indicate the wildcard feature is deactivated, or

/\ OFF

to indicate the wildcard feature is on.

Editor Messages

A list of messages output by the Editor:

MESSAGE	DESCRIPTION
Cmnd?	Illegal Editor command. Error example: FP
Strng?	Illegal specification, or missing string (for example, missing required string for MERGE, or second string for REPLACE). Also occurs as a result of an illegal numeric parameter specification, which causes a part of the numeric parameter to appear as if it were a string. Error example: ME 10R5/ABC/
Col#?	Illegal characters follow the recognized end of the command, or illegal format for a column-number limit specification, or nonnumeric characters used for p and q in LOCATE, REPLACE, DELETE, or MERGE commands. Error example: L.10.23.R/ABC/DEF/XR/M/DICT/MDL, SMITH,JOHN,
Seqn?	Out-of-sequence update. Updating must be done in an ascending line number sequence until an F command is entered.
Eoi m	End-Of-Item reached at line m .
Top	Top-Of-Item (line 0) reached.
L n	Specifies that n is the current line number, or specifies that update action on line n was deleted by an X command.

Table 4-5: Editor Messages.

MESSAGE	DESCRIPTION
Not on file	Item specified in merge command is not on the disk-file.
Item-ID exited.	Item exited by an EX command.
Item-ID deleted.	Item is deleted from the disk-file by the FD command.
Item-ID filed.	Item is updated to the disk-file by the FI command.

Table 4-5: Editor Messages.

Summary of Editor Commands

The following table is an alphabetical listing of Editor commands and their format:

COMMAND NAME	COMMAND FORMAT
AGAIN	A
APPEND LINE	AL{n}/string/
ASSEMBLY FORMAT (on/off)	AS
BREAK LINE	BL{n}/string/
BOTTOM	B
CANCEL	X{F}
COLUMNAR POSITIONS	C
CURRENT LINE	?
DELETE (simple)	DE{n}
DELETE (string-search)	DE{n}"string"{p{-q}}
DELETE	DELETE {DICT} file-name item-list
EDIT	ED{IT} {DICT} file-name {item-list} {(options)}
EXIT	EX{K}
FILE DELETE	FD{K}
FILE ITEM	FI{K} FI{K}{O} item-name FI{K}{O}({DICT} file-name {item-name})

Table 4-6: Alphabetical Listing of Editor Commands.

COMMAND NAME	COMMAND FORMAT
FILE SAVE	FS FS{O}item-name FS{O}(file-name {item-name})
FLIP BUFFER	F
GOTO	Gn or n
HEX OUTPUT	HX
INPUT	I
INSERT	I data
ITEM SIZE	S?
JOIN LINES	JL
LIST	L{n}
LOCATE	L{n}"string"{p{-q}}
MACRO EXPANSION	M
MERGE (same file) MERGE (other files)	ME{n}/item/{m} ME{n}({DICT} file-name {item-name}) {m}
NEXT	N{n}
NULL	ENTER
PRESTORE	P{n} {command-string}
PRESTORE CALL	P
PRESTORE DISPLAY	PD
REPLACE	R{n}
REPLACE ALL	RU{n}/string 1/string 2/ {p{-q}}

Table 4-6: Alphabetical Listing of Editor Commands.

Summary of Editor Commands

COMMAND NAME	COMMAND FORMAT
SUPPRESS (on/off)	S
TAB	TB n, n, n, ...
TOP	T
UP	U{n}
WILD CARD TOGGLE	^
WINDOW	W{n}
ZONE	Z{p{-q}}

Table 4-6: Alphabetical Listing of Editor Commands.

This page intentionally left blank.