HEWLETT PACKARD

User's Guide

HP B1444A
Softkey Driven Editor

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Safety

Summary of Safe Procedures

The following general safety precautions must be observed during all phases of operation, service, and repair of this instrument. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the instrument. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground The Instrument

To minimize shock hazard, the instrument chassis and cabinet must be connected to an electrical ground. The instrument is equipped with a three-conductor ac power cable. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate In An Explosive Atmosphere

Do not operate the instrument in the presence of flammable gases or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

Keep Away From Live Circuits

Operating personnel must not remove instrument covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with the power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Do Not Service Or Adjust Alone

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

Do Not Substitute Parts Or Modify Instrument

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification of the instrument. Return the instrument to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

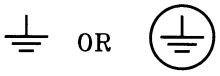
Warning Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.

Safety Symbols Used In Manuals

The following is a list of general definitions of safety symbols used on equipment or in manuals:

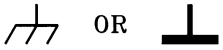
Instruction manual symbol: the product is marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.

Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).



Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating the equipment.

Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. A terminal marked with this symbol must be connected to ground in the manner described in the installation (operating) manual before operating the equipment.



Frame or chassis terminal. A connection to the frame (chassis) of the equipment which normally includes all exposed metal structures.

Alternating current (power line).

Direct current (power line).

Alternating or direct current (power line).

The Note sign denotes important information. It calls your attention to a procedure, practice, condition, or similar situation which is essential to highlight.

Caution

The Caution sign denotes a hazard. It calls your attention to an operating procedure, practice, condition, or similar situation, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product.

Warning

The Warning sign denotes a hazard. It calls your attention to a procedure, practice, condition or the like, which, if not correctly performed, could result in injury or death to personnel.

How To Use This Manual

This manual contains information about operating the Softkey Driven Editor. At the beginning of each chapter there is a summary of the information which you will find in that particular chapter. The appendices of this manual contain additional information which either supplements that in the chapters, or which helps explain the use of the Softkey Driven Editor.

When you use this manual and have a specific action you want to take, or a specific subject of interest, the following summary may be helpful. Refer to the action or subject of interest in the left-hand column. Then locate the chapter or appendix in the right-hand column for the action or information which meets your needs.

IF YOU WANT TO:	THEN REFER TO:
Learn About the Softkey Driven Editor	Chapter 1 - This chapter provides a description of the Softkey Driven Editor and its features, the relationship to your operating system and to the HP 64000 editor, as well as other information about finding answers and updating software.
Prepare to Use the Editor	Chapter 2 - This chapter tells you about procedures for installing the Editor, how to modify your .profile for easier use, how to invoke the Editor and its options, and about use of your terminal for keyboard entry.
Learn Editor Modes and Structure	Chapter 3 - This chapter has information about how the Softkey Driven Editor is structured, the levels of softkeys available, a functional description of its three modes, how shell commands are used with the Editor, and how to recover files from "crashed" edit sessions.

IF YOU WANT TO:	THEN REFER TO:
Get Started At Creating And Editing Text Files And Commands	Chapter 4 - This chapter includes descriptions of the keyboard entry, the display format, creating and editing files, and entering and editing command files.
Learn Details About Commands Available And The Syntax For Using The Commands	Chapter 5 - This chapter has descriptions, examples, definitions, and parameters which apply to use of the Editor. It includes syntax examples and detailed syntax diagrams, and cross-references to other commands.
Find Help Or Solve Problems While Using The Editor	Chapter 6 - This chapter describes some possible problems or questions which may arise while you are using the Softkey Driven Editor. It describes sources for answers, and points to specific bothersome areas.
Find The Meaning And Explanation For Softkey Driven Editor Status And Error Messages	Appendix A - This appendix contains a listing of the messages which appear during an edit session to let you know when an error has been made or when a choice is available to the User.
Use The while Command In Conjunction With Other Commands	Appendix B - This appendix contains the truth values related to executing the while command and the truth value returned at the end of that command.

Contents

6 Help and Problem-Solving

A Editor Status and Error Messages

B Editor Command Truth Tables

C Comparison with HP 64000 Editor

Index

Installation Notice

General Information

Chapter Overview

This chapter provides the following information to help introduce you to both the Softkey Driven Editor and to this manual. Here are some of the subjects which are in this chapter:

- Brief description of features and editing capability.
- A description of the relationship between the Softkey Driven Editor and your operating system.
- How this manual is organized, including how to get started. Also a reference to "Using This Manual" and about other manuals you should have access to.
- How to find answers and help when you can't find something or don't understand something.
- A reference to information about the Software Materials Subscription (SMS), which will keep you current with new software.
- Reference to information about the Response Center, a technical assistance subscription service.

Editor Description

Description of the Softkey Driven Editor

The HP B1444A Softkey Driven Editor provides text entry and text manipulation commands in a softkey-driven editor which facilitates revising source code and text files. It includes all the features found in the HP 64000 workstation editor, enhanced to take advantage of your operating system. Command files written for the HP 64000 workstation editor are compatible with the Softkey Driven Editor and it is compatible with files developed using the "vi" editor supplied with your operating system.

Displays from the editor support a virtual screen-width of 240 characters on each line. The screen is easily scrolled left and right to let you view all parts of the text. Displays are compatible with HP terminals, HP bit-mapped monitors, and your operating system window manager on the HP 9000 Series 300/400 and Series 700.

Commands, variables, and functions appear on softkeys which track the proper syntax for your next entry. For more experienced users, automatic command completion (done as partial commands are typed in, then completed with the Tab key) provides an efficient way for you to enter commands for execution. There are both standard and enhanced edit functions, including global search and replace capability. Other functions or commands include character and string "wildcards"; extract, retrieve, and merge for efficient editing of files; and split and join commands to alter lines as you wish.

Editor Relationship to Your Operating System and the HP 64000

The Softkey Driven Editor runs on the HP 9000 Series 300/400 and HP 9000 Series 700 computers with your operating system. This editor is similar to the HP 64000 Logic Development System (LDS) workstation editor, but is enhanced by your operating system compatibility. Compatibility of your shell commands and editor commands make building and execution of commands an easier task. Use of your shell variables and your .profile directory make it easy to access the Editor and to move back and forth between the Editor and your operating system shell.

Use of your operating system and the ability to create, edit, and execute command files from the editor make other parts of your development tools easier to access and use. Command files built in the editor will help to get more efficient use of emulation, analysis, and assembler/linker/compiler features. To obtain full capability for command files, logging commands, and directory changes, you also must have User Interface Software (also called "pmon") installed on your operating system; see chapter 2.

What's in This Manual?

Using This and Other Manuals

Refer to the "Using This Manual" section at the front of this manual for a guide to locating chapters and appendices which will help you with specific needs. Also refer to the table of contents and the index to locate specific paragraph topics or subjects.

Suggestions for Learning the Editor

It will depend on your previous experience with other editors and systems as to how much familiarization is needed before you begin to feel comfortable with using the Softkey Driven Editor. Looking at the table of contents for each chapter will help you to learn which areas are familiar ground and which ones may contain new information. If you study chapters 2, 3, and 4 in order, you then will be able to use the editor, having learned its modes and its structure. You should refer to the command syntax information in chapter 5 so you are familiar with the syntax examples and diagrams there, and can refer to them as questions arise.

If you have previously used the HP 64000 workstation editor and are familiar with most aspects of your operating system, you may need only to refer to the syntax chapter to refresh a concept or to look up relatively minor points about how certain commands or variables operate. Also, if you have used the HP 64000 editor, you may want to look at appendix C which describes differences between the HP 64000 editor and the Softkey Driven Editor.

Where to Find Terms and Conventions

Where possible, this manual incorporates standard terms and definitions, especially regarding your operating system. When practical, terms which may be unfamiliar to most users and terms which are unique to the Softkey Driven Editor are defined in text as close as possible to the use of the term. One important convention is that keywords which you enter from softkeys are shown in **boldface type like this** to make it easier for you to identify the commands and variables in text. Command parameters entered from the keyboard are shown in standard type like this. The syntax conventions used for commands and variables are shown near the beginning of chapter 5.

Finding Answers and Help

Looking in the table of contents and in the index will often provide the clue as to where to find answers. In addition, chapter 6 contains a summary of how you might approach finding help and solving problems. Some questions and subjects covered there may be helpful. There also is an on-line manual which has information about the editor. And, lastly, there is a Hewlett-Packard Response Center technical service to which you can subscribe for answers and problem-solving related to the HP 64000-UX software and system usage.

Determining Software Revision Numbers

You can determine the revision number, in the form "Rev YY.XX", of your software product by entering:

more /system/productname/productrev

where "productname" is actually the model number of the product (B1444, no letter suffix, for the Softkey Driven Editor).

For example, to determine the software revision number for this Editor, enter:

more /system/B1444/productrev

You then will see the following information on the screen:

B1444-19XXX SOFTKEY EDITOR HP9000/XX0 Rev YY.XX
dd/mmm/yy hh:mm:ss

The software version appears on the STATUS line when you enter the editor by way of the directory /usr/hp64000/bin. The software version appears on the STATUS line (until your first keystroke) as follows:

sk: YY.XX (c) 1986, Hewlett-Packard Company

Software Materials Subscription and Response Center

Hewlett-Packard offers a Software Materials Subscription (SMS) to provide timely and comprehensive information for your development environment. Also, there is a technical assistance subscription service available through a Response Center. SMS and the Response Center are described in the "*HP Support Services*" Manual (supplied with this manual).

Installation Notice

Preparation for Use

Chapter Overview

This chapter provides information to help you prepare for using the Softkey Driven Editor on your operating system, including the following:

- Information about procedures to install and to update software for the Softkey Driven Editor.
- Using your operating system shell and your ".profile" file to make it easier to use the editor with your operating system.
- How to invoke the editor and available options.
- What you should consider when configuring the data terminal to be used for keyboard entry with the Softkey Driven Editor.

Installing Editor Software on Your Operating System

Software for the Softkey Driven Editor will reside on the hard disk of your operating system (HP 9000 Series 700 or Series 300/400 computer which supports your terminal. Initial software and subsequent updates will be installed by your operating system System Administrator. It is recommended that the System Administrator perform the tasks outlined in a following paragraph in this chapter under the heading "System Administration Tasks". Some of the tasks are required on a "first-time-only" basis. If these are not performed, your ability to recover from an editor "crash" might not be successful.

Note

Additional software is required in order to use command files with the Softkey Driven Editor. To use command files with the editor on the operating system (installed on an HP 9000 Series 300/400 and 700 computers), the HP B1471 64000-UX Operating Environment software is required. For more information, contact your local HP Sales/Service Office.

System Administration Tasks

Involvement of the System Administrator with the Softkey Driven Editor will be minimal, usually only at installation time or for an update. Here are some tasks to be performed by the System Administrator after installing the Softkey Driven Editor software:

1. If there are enough users on the system using the Softkey Driven Editor, as well as other development tool applications, then you might find it worthwhile to add the **/usr/hp64000/bin** directory to the PATH shell variable in the **/etc/profile** file.

2. The System Administrator should also modify the **/etc/rc** file to allow for the "crash" "skrecover" command to work. The modification required is to add the line **skpreserve** in the section of the **/etc/rc** file that covers "miscellaneous housekeeping". A good place for this command is near "expreserve(1)". The "skpreserve" command moves any temporary files for users from the "/tmp" directory to the **/usr/preserve** directory, where the **skrecover command** may be used to recover them. The "skpreserve" command also sends mail to the owners of the temporary files, informing the owners that crashed edit sessions have been preserved.

".profile" and Operating System Shell Variables

Changing Your ".profile" File

You can use the ".profile" file in your $HOME directory to set options for your operation system session and to perform many functions automatically. (The following information applies only to users with "sh" or "ksh". If you use "csh", you should check your manuals to see how to accomplish the same tasks.) If you do not have a .profile, it can be created easily, or, if one exists, it can be modified using the information about shell variables in the following paragraph. After changing your .profile, you can make it active by doing one of the following: (1) Log out of the operating system and then log back in, which causes the .profile to be executed; or (2) Re-execute the .profile from

your $HOME directory (or the directory where the .profile is located) by typing ". .profile" from the shell.

Using Shell Variables

Shell variables added to your .profile will make many tasks easier. Here are some examples of what you may wish to add to your .profile, followed by brief explanations of why each is useful:

```
PATH= $PATH:/usr/hp64000/bin
HP64KPATH= .:$HOME/bin
PRINTER= lpr
MAXREC= 15
SKTOP=
EDITOR= /usr/bin/sk
export HP64KPATH PRINTER MAXREC SKTOP
export EDITOR
```

The **/usr/hp64000/bin** directory contains the "sk" editor, and it should be added to the search path you use unless the $PATH variable already contains it. This may be checked by executing "echo $PATH" from your shell and looking for this directory name.

The $HP64KPATH shell variable is used to control which directories are searched when the user tries to execute a command file. If the filename has a relative path, then the directories specified by the HP64KPATH shell variable are searched from left to right until the command file is found. The search is relative from the directory and not just within it. If no file is found, then an error is generated. If this shell variable is not defined, then all relative filenames check only the current directory.

Setting up "PRINTER= **lp** will allow the Softkey Driven Editor to use this shell variable when a **list printer** command is invoked. The shell variable should be set to some shell command which will cause the output to be spooled to the printer. Also, this command should accept standard input when no argument is given, as with "**lp**(1)".

The $MAXREC shell variable is also used by the "sk" editor in the context of putting files being overwritten into your user's recovery directory. Since the default number of files in the recovery directory is 128, you may want to select a smaller number to minimize disk usage.

The SKTOP shell variable can be used to control the placement of the softkeys, STATUS, and command lines for the Softkey Driven Editor, as well as for other shell commands. Depending on how this shell variable is set, it will cause most operating system applications to place the softkeys, STATUS, and command lines at the top of the display. This action is similar to the "-v" option of the editor as defined in the following, but the SKTOP shell variable can apply to selected operating system applications.

The SKTOP shell variable can be made to fit your individual needs with variations as follows:

SKTOP=	This will affect most operating system features available on the system.
SKTOP= sk	This will affect only the "sk" editor.
SKTOP= all	Same as SKTOP= (see above).
SKTOP= sk:pmon:	This allows setting only selected individual applications (separated by colons).

The EDITOR shell variable will set the default editor for various operating system applications, such as for "mailx(1)". If this is defined as shown above, and exported, then the "sk" editor is invoked rather than "ed(1)" or "vi(1)", for example.

It is crucial that you export the new shell variables, so other programs, such as "sk", can use them.

Other shell variables may be defined by the user as desired. The shell variables may then be accessed by the user from the command line. This is accomplished by typing a dollar sign, or '$', followed by the shell variable name. The shell variable name may also be enclosed by braces, "{}", if the name is followed by characters that are not part of it. The editor, when it sees the shell variable, will then search for an exported shell variable and, if found, will replace the dollar sign and name with the value assigned by the user. The shell variable may be used anywhere on the command line, including within filenames.

Invoking the Editor and Options

The Softkey Driven Editor is invoked as "sk" from your operating system shell. The syntax for the command, along with the explanation for its options, is:

sk [-c] [-n] [-v] [-f < NUMBER>] [-t < NUMBER>]
[-i < FILE1>] [< FILE2>]

where these definitions apply:

-c This option turns off the column number information which tracks the cursor location in the **INSERT*** and **REVISE*** modes. This accomplishes the same thing as the **colm_num off** (column_numbers off) command from the editor.

-n This option prevents the conversion of tab characters to spaces when loading the initial file. This gives the same result as the **no_tab_c** (no_tab_convert) for the **edit** command.

-v This option will locate the STATUS line, command line, and softkey labels at the top of the terminal display, instead of at the bottom by default. This option may be your own preference, or, with some terminals, it may avoid up-and-down bouncing of these lines when you use the up arrow and down arrow keys, which may occur when the STATUS, command, and softkey lines are at the bottom of the display.

-f < NUMBER>

This option allows the user to preset the tabstops that are used in the **INSERT** and **REVISE** modes of the editor. This option does not affect the conversion of tab characters when reading or writing files. Instead, it performs the same function as the **tabset fixed < NUMBER>** command, which affects which column the cursor moves to when a tab or shift-tab key is pressed. Tabstops will be positioned every NUMBER columns, starting in column 1.

-t < NUMBER>

This option controls how tab characters are converted into spaces when the initial file is read into the editor. The default action is to place tabstops every eight (8) columns and to interpret tab characters as moving the text cursor to those positions. The -t < NUMBER> option lets you enter the number which causes tabstops to be placed at fixed intervals, starting with column 1. This option is the equivalent of the **tab_conv** (tab_convert) option to the **edit** command.

-i < FILE1>

This option lets you change the name of the initial edit file after it is read from the disk (or as created if not already existing). A file could be read in with one name, but saved later under a different name. This option is the same as the **into < FILE2>** used with the **edit** command in the editor.

< FILE2>

This would be the name of a file to be read into the editor. If this name is not entered, a new, empty file is created in the editor. Tab characters will be handled as specified by the two related tab options above. If neither the "-n" nor "-t" option is used, the default action would be to convert tabs as if a "-t8" option were used. For more details, see the **edit** command syntax in chapter 5. The "< FILE2> " option acts the same as the "< FILE1> " option used with the **edit** command.

Data Terminal Configuration

In general, you should configure your data terminal being used for the Softkey Driven Editor as specified in the manuals for your operating system computer, either the HP 9000 Series 300/400 or the HP 9000 Series 700. If you experience occasional problems with the data transfer rate which creates "garbage" on the display, you can redraw the display to its last "clean" appearance by using CTRL l (lowercase L). If you have frequent problems with "garbage" being created, it may be necessary to reduce the baud rate. Depending on the number of sessions and the type of activity on the system, reducing the baud rate will solve most data transfer problems with the screen display of the Softkey Driven Editor. Terminal display problems, if they occur, are likely to be observed with large jumps in a file and with large data files, such as when you use NEXT page, PREV page, Home-Up, or Home-Down. This could also happen if you press a key repeatedly or hold it down, as with the Return, Insert Char, or **--ETC--** softkey.

Installation Notice

Modes, Structure, and Operating System Connections

Chapter Overview

This chapter provides the following information:

- A description of how the Softkey Driven Editor is structured.
- A description of the four levels of softkeys.
- Functional descriptions of the three editor modes (Command, **INSERT***, and **REVISE***).
- Information about your shell commands used for interrupts and about other operating system connections.
- Descriptions of how to recover "saved" files inadvertently overwritten and of how to recover from "crashed" edit sessions.

Editor Structure

General Description

The Softkey Driven Editor is based on the structure and operation of an editor originally developed for the HP 64000 Logic Development System (LDS). However, the Softkey Driven Editor operation has been enhanced by use of your shell commands and techniques. If you previously have used and are familiar with the HP 64000 LDS editor, you may wish to look at appendix C which describes most of the areas of difference between that editor and the Softkey Driven Editor. In general, the edit file is considered to be any desired number of lines. (The editor can display a line number up to 99999. It can still work for files with more lines than this, but only the last five digits are displayed.) Each line consists of 240 characters; normally spaces are appended to each line to equal 240 total characters.

Importance of Tab Characters

Tab characters exist within the editor only as single control characters. For normal operation, the Tab key moves the cursor to the next tabstop (as defined by the tabset command), rather than inserting a tab character. Thus, when you are editing files where tab characters are to be preserved as control characters, use care to choose the **no_tab_c** (no tabconvert) option. Or, you can choose the **tab_conv** (tabconvert) option which will expand tab characters into spaces. If this expansion is done, the tab character is expanded into the number of spaces required to move to, but not including, the next tabstop. One other important item for you to note is that the Softkey Driven Editor strips-off (removes) all trailing spaces in all cases; this could be critical if those spaces are needed, as in checksums. There is more information about tabconversion in chapter 5 under **edit** and **end** syntax.

Editor Softkey Levels

When you enter the Softkey Driven Editor, softkey labels will display the first level of commands. The furthest right label will show **--ETC--** which stands for "et cetera", simply meaning "and others". Pressing the **--ETC--** softkey will display "other" labels on successive levels. There are four levels of softkeys which are accessed in a circular fashion with **--ETC--**. Most softkeys on the four levels provide additional softkey labels for entering further commands or variables (other softkeys provide information on the STATUS line as to what you should do next). The four levels of softkeys appear as follows:

FIRST LEVEL:

INSERT REVISE delete find replace < LINE+ -> end --ETC--

SECOND LEVEL:

merge copy extract retrieve join split list --ETC--

THIRD LEVEL:

renumber repeat tabset range autotab save edit --ETC--

FOURTH LEVEL:

while insert colm_num log help (blank) **WHATCHAR --ETC--**

Operating System Interrupts of the Editor

Certain shell commands (signals) will interrupt the editor or commands being executed from the editor. This use of interrupt signals is important in such instances as a phone/modem hangup or having entered an infinite loop in the **while** command. Most editor commands will run to their completion before any interrupt signal takes effect. However, the commands which will terminate prematurely with an interrupt are:

copy	merge
edit	renumber
end	repeat
extract	replace
find	save
join	while
list	

If one of these specified commands is terminated prematurely by an interrupt, an error message appears on screen. Interrupt signals which will affect the editor are as follows:

CTRL | (requires keystrokes of "control" key and "| "). This will cause the editor to terminate whatever it is doing and to exit immediately. Any temporary files will be removed and the current text file will be lost. This signal will take effect at any time. (Using **CTRL |** is equivalent to the operating system signal SIGQUIT ; consult your operating system references for more about this signal. Also, the stty setting for 'quit' may be different from CTRL | . To see what control character 'quit' is set to, do a "stty -a" command from the shell. If the stty settings are different from CTRL | , then that control sequence should be used.)

CTRL c This signal will terminate prematurely the commands specified previously and return you to the Command mode. If you are already in the Command mode, the signal is ignored. It is also ignored if you are already in either the **REVISE***, **INSERT***, **TABSET***, or **RANGE*** mode. (**CTRL c** is equivalent to your operating system signal SIGINT; consult your operating system references for more about this signal.)

SIGHUP This signal is generated by a modem hangup and it causes the editor to terminate any current command in process. When this happens, a "crash" recovery file is created which can be used to re-create your current edit session ("skpreserve" is invoked automatically). (Consult your operating system references for more about this signal.)

SIGKILL This signal cannot be intercepted or ignored. When it occurs, the edit session is terminated immediately and no temporary files are removed. However, recovery is possible in the same way as for a "crashed" session (user needs to invoke "skpreserve"). (Consult your operating system references for more about this signal.)

Editor Functional Modes

There are three functional modes to the Softkey Driven Editor: Command, **INSERT**, and **REVISE**. Your edit session always will be in one of these three modes. Each of these modes is covered in chapter 5, Editor Command Syntax. The following paragraphs provide a brief description of the modes.

Command Mode Description

The Command mode is used to issue both editor commands and shell commands from the command line. This mode allows you to execute commands, to build commands for execution, and to log commands to a file. Comments can be added in the Command mode by preceding them with a "# " (number sign). Commands and comments are entered on the command line from the keyboard with up to 240 characters displayed on-screen as three separate lines on standard terminals. The entire command line is accessed by

using the roll-up and roll-down keys. The current line in the text (source) file is indicated by a > symbol in the left margin of the edit area. To fully utilize command files, logging commands, and directory changes, User Interface Software (also called "pmon") must be part of the system; see chapter 2.

You can enter the Command mode in several ways:

1. From your operating system shell, you can invoke the Softkey Driven Editor (refer to chapter 2) and specify a file to be edited. This places you on the command line, from which you either can invoke commands or enter the **INSERT*** or **REVISE*** mode to edit the file.

2. From the **INSERT*** or **REVISE*** mode, pressing either key the second time will put your session in the Command mode (before pressing the key the second time, STATUS line message reads "To leave, press **INSERT*/REVISE*** key again").

3. From either the **INSERT*** or **REVISE*** mode, you can press any command softkey and move temporarily to the Command mode. From there you can execute a command, which when completed will return you to your current line of text (this is true except for end, which, of course, takes you out of the edit session). However, once you are on the command line from either **INSERT*** or **REVISE*** mode, if you now clear the command line, you remain in the Command mode. (Command line is cleared either by using the Backspace key or the "kill" character from your operating system stty setting. If you cannot execute the command or if you change your mind, it will be necessary to clear the command line before you can re-enter either **INSERT*** or **REVISE*** mode.)

The Command mode is exited by pressing either the **INSERT** or **REVISE** softkey. You also have the choice from the Command mode to exit the session by pressing **CTRL d** simultaneously. This will display a message on the command line reading "Do you really want to quit?" A "yes" answer will exit you to your operating system shell. A "no" answer leaves you on the command line and the STATUS line reads "No changes lost, edit resumed".

INSERT Mode Description

The **INSERT*** editor mode is used to enter new lines of text into the current edit file, both for existing files and for files just being created. New lines are identified by a > NEW symbol rather than by a line number, and the > NEW line can be moved up and down in the file until either Return is pressed or the

INSERT* mode is exited. As each new line is created, by pressing Return, another > NEW line appears below it.

The **INSERT*** mode is entered directly by invoking the Softkey Driven Editor from your operating system shell with no file name specified. Another direct way to enter **INSERT*** mode is by pressing the **INSERT** softkey from either of the other editor modes. When you invoke the Softkey Driven Editor from your operating shell with a file name specified, it takes you to the Command mode, from which you can press the **INSERT** softkey.

You can exit the **INSERT*** mode by pressing the **REVISE** softkey, to enter that mode, or by pressing the **INSERT*** softkey to move to the Command mode. (Note, however, if you press **REVISE** from **INSERT*** and the file is empty, a message appears reading "ERROR: Unable to enter **REVISE** mode since file is empty", and you are then left in the Command mode.)

Another way to exit the **INSERT*** mode is to press simultaneously **CTRL d** keys which moves you to the command line and asks the question there "Do you really want to quit?". A "yes" response will exit you to your operating system shell, while a "no" response sends you back to the **INSERT*** mode with the STATUS line message "No changes lost, edit resumed".

You can move temporarily to the Command mode by pressing any of the command softkeys available from the **INSERT*** mode (except for **end**, which will exit you from the session to your operating system shell). As that command is completed, you are placed back in the **INSERT*** mode. (If you do not execute a command after temporarily leaving **INSERT*** edit mode --because of a syntax problem or because you changed your mind-- to re-enter **INSERT***, you first must clear the command line either with the Backspace key or by pressing the "kill" key, as defined in your operating system stty setting, and then pressing **INSERT** again.)

REVISE Mode Description

The **REVISE*** editor mode is used to enter changes in the current edit file for existing (saved or nonempty) files. The current line is indicated by the > symbol in the left margin and the cursor is positioned at the current column for text entry. You can move about in the file, making changes by typing over text or by using the various keyboard functions to modify the file.

The **REVISE*** mode is entered directly for a current edit file from either the Command mode or **INSERT*** mode by pressing the **REVISE** softkey. (If you try to enter **REVISE** mode and your **INSERT*** file is empty --that is, has no

lines/text in it-- a message appears reading "ERROR: Unable to enter **REVISE** mode since file is empty", and you are left in the Command mode.)

You exit the **REVISE*** mode either by pressing the **INSERT** softkey to enter that mode, or by pressing the **REVISE*** softkey to move to the Command mode.

Another way to exit the **REVISE*** mode is to press simultaneously **CTRL d** keys which moves you to the command line and asks the question there "Do you really want to quit?". A "yes" response will exit to your operating system shell, while a "no" response sends you back to the **REVISE*** mode with the STATUS line message "No changes lost, edit resumed".

You can move temporarily to the Command mode by pressing any of the command softkeys available from the **REVISE*** mode; if you then issue a command, as that command is completed, you are placed back in the **REVISE*** mode. (As with **INSERT*** mode, if you temporarily move to the command line but do not execute a command, to re-enter the **REVISE*** edit mode, first you must clear the command line either with the Backspace key or with the "kill" character, as defined in your operating system stty setting, and then press **REVISE** again.)

Recovery of Saved Files

If you have an operating system directory called $HOME/.recover (a directory off your user's $HOME directory), the Softkey Driven Editor will automatically protect you from overwriting a file with the **save** or **end** command. When you issue a **save** or **end** command, the editor checks to see if that file already exists. If it exists, the editor next checks to see if you have a $HOME/.recover directory, and, if so, moves the file about to be overwritten into your $HOME/.recover directory. The operating system **rcvr** command allows you to retrieve files from the $HOME/.recover directory; see operating system "**rcvr**" Command For Recovery in this chapter.

"purge" Command to Move Files

You can use the shell command **purge** to move files listed on the command line into your user's $HOME/.recover directory. If that directory does not exist, then no purge is done. (For information about $HOME/.recover directory, see "Recovery Of Saved Files" in this chapter.) If a file is placed in the $HOME/.recover directory, it is renamed as a number and added to the directory file. Also, if the $MAXREC count for the directory is exceeded, the oldest file in the $HOME/.recover directory is deleted to make room for the newest file. This command is invoked as:

purge file1 [file2 . . . filen]

Note that any links to other files will be lost at this time. The user must re-create these links if this file is to be recovered at some future time.

"rcvr" Command for Recovery

The shell command **rcvr** retrieves files from your user's $HOME/.recover directory. If more than one file has been purged with the same file name, the one purged last will be recovered first. You can use either absolute or relative file names. If the file name is relative (not starting with a " / "), the current directory is used to create an absolute path name. A search for that path name is made in the $HOME/.recover/directory file and if it is found, the sought file is moved back to its original position. The $HOME/.recover/directory entry is removed. You must use the complete file name because wild-card characters will not be expanded correctly by the shell. For example, "**rcvr** a*" would be expanded using the files starting with "a" in the current directory, not by using only those purged files that start with "a". The rcvr command is invoked as:

rcvr file1 [file2 . . . filen]

If a file is to be recovered to a directory other than the one it was purged from, or if the file is to be recovered under a different name, the following command may be used for each file to be recovered:

rcvr -f newname oldname

If multiple links to the original file exist, it may be necessary to remove the original file and all files linked to it, move or recover the backup copy to the original filename, and then relink to all other paths.

"dirrec" Command for Directory Listing

If you have a $HOME/.recover directory for use in protection against overwriting files (see Recovery of Saved Files in this chapter), you can use the shell command **dirrec** to produce a listing of that directory. This works much like the **ll**(1) shell command. The listing will appear in order from newest file to oldest file. Note that an absolute path is given for each file, which must be matched in order for a file to be recovered with the **rcvr** command.

Recovery of "Crashed" Edit Sessions

The ability to recover an edit session lost due to a variety of possible causes, usually referred to as a "crash", is always important to you as a user. Some possible causes of such a "crash" include: a telephone or modem hang-up, a system problem, a power failure, an accidental catastrophic signal which cannot be recalled or canceled, or other unanticipated reasons. The Softkey Driven Editor provides the procedures to allow recovery from a "crash". The "**skrecover**" command will allow you to recover all, or nearly all, of a session which "crashed", as well as the list of your sessions which have "crashed". This recovery requires that "**skpreserve**" be run prior to the use of the "**skrecover**" command.

Using the skpreserve" Command is Required

Having the operating system "**skpreserve**" command invoked on your system after you experience a "crash" is required if you expect to recover any files which were being edited at the time. If a computer system is recovering from a power failure or is being initialized, "**skpreserve**" will be executed automatically (see System Administration Tasks in chapter 2). When "**skpreserve**" has been invoked, it will send your mail informing each user that

a crashed edit session has been saved for them. The "**skrecover -d**" command can then be used to get a list of file names that represents these "crashed" edit sessions.

Note

Normally, to handle a catastrophic system failure, "**skpreserve**" will be invoked during the multi-tasking initialization. If a user has a "crash" due to a phone/modem hang-up, "**skpreserve**" is invoked automatically for that user only. For "crashes" due to some other reason, such as a SIGKILL signal, the user may invoke "**skpreserve**" to affect only those temporary files owned by the user. However, if the "**skpreserve**" command is executed while the user (or someone else using the same login) is in an edit session, the user will get a mail message that there has been a "crash" for the file being edited, even though that edit session itself has not "crashed".

"skrecover" Command File

If your edit session experiences a "crash" and you receive mail that a recoverable file exists, the "**skrecover**" command may be used to get a list of "crashed" recoverable files, as well as to recover them. Options to the "**skrecover**" command are:

skrecover [**-d**] [**-r**] [**-u**] [**-n** FILEID] [**-f** FILE]
[**-t** NUMBER] [FILE]

where the following definitions apply:

-d — This option MUST BE USED ALONE, with none of the other "**skrecover**" options or file names at the same time. Using the **-d** option alone provides your own user list of recoverable files with their absolute path names; the date/time of the "crash"; a unique file id for that "crashed" session; and the name, if any, of the edit file at the time of the "crash". If the file had no name, the listing uses "> > No file name < < " . The full path name must be used when recovering a file.

-r — This option may be used to cause a recovery file to be removed. This is useful if the user has no desire to recover the "crashed" edit session. If

more than one file exists with the same name, then the **-n** option should be used in conjunction with this option to make sure the correct recovery file is removed. A filename is required, unless an unnamed edit session is to be removed. Note that the edit session cannot be recovered once the recover file is removed with this option.

-u

This option will inform the "**skrecover**" command that it is okay to overwrite an existing file, if there is one. If you do not use the **-u** option, and a file with the same name as the recovery file already exists, an error message is displayed and the recover command will not work until you have done one of the following: (1) enter the **-u** option if it is okay to overwrite; (2) delete, rename, or move the existing file; or (3) use the **-f** option to rename the file being recovered. The reason this decision must be made is that part of the recovery process is to purge the file to your user $HOME/.recover directory, if it exists, or to delete the file (for information about $HOME/.recover directory, see that heading in this chapter). When this purge, delete, or rename is done, the recovery of the "crashed" edit session continues.

-n FILEID

This option identifies a specific "crashed" edit session with a unique file id number, obtained only by using the **-d** option. This would be used if more than one "crashed" session exists with the same file name in it. In that case, if the file id is not used, then the "crashed" session listed first with the **-d** option will be recovered first.

-f FILE

This option allows you to change the name under which the recovered file is to be saved. You must use this option if the file being recovered did not have a name when the "crash" occurred (the "**skrecover -d**" command will inform you of the valid names for recoverable files). If the file name

used for this option does not start with a " / ", it is taken to be relative to your current directory.

-t NUMBER — This option lets you choose to have tabconversion of the file being recovered. The default without this option is to keep all the white space in the file as spaces, that is, no tabconversion would occur. If you use this option, enter a number from 1 through 239. For more information about tabconversion, see the **end** or **save** command syntax in chapter 5.

FILE — This option refers to the name of the recoverable file. If this file is relative (does not start with " / "), the current directory is used to build an absolute path name; otherwise, it uses the absolute path. This file name is then compared with file names from the directory generated by the "**skpreserve**" command after the "crash", and a file owned by the user will be recovered. If the file did not have a name at the time of the "crash", then this file name would not be used. In other words, a missing file name matches an unnamed edit session.

Installation Notice

Getting Started

Chapter Overview

This chapter provides information to help you get started using the Softkey Driven Editor:

- A description of the typical keyboard used with the Editor.
- A description of the display format and information available during an edit session.
- Basic information to get you started using the Softkey Driven Editor to create, modify, end, and save both edit and command files.
- Explanations of editing techniques for new and existing files.
- Descriptions of how to enter, edit, and use command files.

Note Chapter 2 explains ways for you to invoke the Softkey Driven Editor.

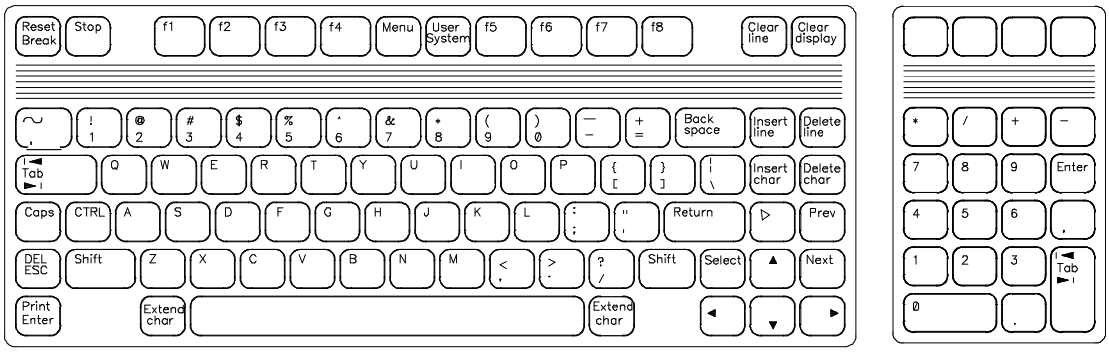
Understanding the Keyboard and Display

When you use the Softkey Driven Editor, your input from the keyboard and the output you see on the display will vary somewhat, according to the terminal keyboard and CRT display. Explanations of the keyboard functions in this manual and descriptions of the data and messages displayed will be based on a typical data terminal. Differences in keyboard labels are noted where applicable, and the only major differences in what you see on a display unit will be the number of lines or columns which are on-screen at a time. The information which follows about the keyboard and display will be helpful in familiarizing yourself with the features and operation of the Editor.

Keyboard Layout and Labels

Knowing your keyboard layout and key labels will help you make decisions about entering commands and text quickly and accurately.

As an example, the standard keyboard used with the HP 9000 Series 300/400 and Series 700 is shown in the following figure. The descriptions of Keyboard Input And Functions which follow are based on the labels and keystrokes used with that "standard" keyboard. More information about the standard keyboard is located in the *HP9000 Application Execution Environment User's Manual* and the *HP 9000 Peripheral Installation Installation Guide*. For more information about use of the converted HP 64000 development station keyboard, see the converted development station manuals.



Standard Keyboard

Keyboard Input and Functions

To enter commands or text into files with the Softkey Driven Editor, input will come from three keyboard sources: (1) standard alphanumeric character keys, such as: abc123; (2) function keys which allow modification of the text being entered; and (3) the special set of eight double-size keys, commonly called softkeys because the labels which appear on the display to identify their functions are not fixed, but rather are changed by software programs to give a variety of commands. The eight keys are normally labeled f1 f2 f3 f4 f5 f6 f7 f8 on the keyboard and provide the power of quick, easy entry of both operating system and editor commands. The following descriptions will cover primarily the keyboard functions which provide either a capability to use standard keyboard keystrokes in a specialized way or which modify the standard (expected) use of the keyboard. The way in which you can use the softkeys to provide very efficient entry and editing of text files and command files is covered in chapter 5, Editor Command Syntax.

Here are descriptions of keyboard functions which have special meaning when you use the Softkey Driven Editor (they are divided into two groups, one-key functions and two-key functions):

ONE-KEY FUNCTIONS (require pressing only one key)

CURSOR KEYS

There are two differences in how these keys operate, one which you might expect and one you might not. Here is how these operate:

Left-arrow, Right-arrow. As you would expect, these keys move the cursor left and right on either the command line (80 characters wide) or in the text file area (240 columns wide). However, since only 72 columns are displayed on the typical terminal, if you move the cursor beyond that on the right, your text will begin to shift to the left whether or not you actually enter characters beyond column 72. To return to the left-hand margin of the text, you can press Return, or use the < or Shift Tab (backtab). On the command line, the cursor moves left or right normally, shifting to the previous or next line as you reach the left or right margins.

	Up-arrow, Down-arrow. As you would expect, these keys move the cursor up or down to change your current line of text. However, in the Command mode when the command line exceeds 80 characters, the Up and Down Arrows operate differently. When the command line is under 80 characters, these keys move the current line, indicated on-screen by > , up and down in the file. When the command line exceeds 80 characters, these keys move the cursor between lines of the command line until one of its boundaries is reached, at which time they operate normally again. (This is different from HP 64000 editor; appendix C lists differences.)
Insert Char	(insert character) This inserts characters at the cursor position and pushes text (including the character under the cursor) to the right in all modes. The "I" enunciator appears at the right of the STATUS line when Insert Char is enabled. All special keys automatically turn Insert Char off. In the Softkey Driven Editor, there is no "placeholder" symbol (underlined caret) as there is on the HP 64000 LDS (see appendix C for a summary of differences).
Delete Char	(delete character) This will delete the character under the cursor, which shifts all characters following the cursor on the line to the left by one character. In the Insert Char mode, the Delete Char key turns the Insert Char off.
CLEAR TO END OF LINE	This function uses the Clear Line key in a different way, namely, to clear text from the current cursor position to the end of the line in all modes.

Delete Line	(delete line key) This function deletes the current text line and places the cursor on the next line (if there is one). In the **INSERT***, **TAB***, or **RANGE*** modes, Delete Line key only clears the line.
Insert Line	(insert line key) This will insert a new line following the current line and position the cursor at the beginning of the text line. In the **INSERT*** mode, Insert Line key acts like a Return would. It does not function in **TAB*** or **RANGE*** mode.
Tab	This key moves the cursor to the right from the current location to the next tabstop in the **INSERT*** and **REVISE*** modes. In the Command mode, Tab produces a much different result: if you enter a partial command and press Tab, the editor will attempt to complete the command entry. If the entry forms the beginning of a unique command keyword, then that entire command keyword is placed on the command line. If it is not a unique command keyword, various possible keywords are displayed on the STATUS line for your selection provided that these keywords match the text on the command thus far. Also, if all of the possible keywords share more characters, then those characters will also be placed on the command line until the keywords diverge. If Insert Char is on when Tab is pressed, Insert Char is turned off.
Next Prev	(next page, previous page) These keys remove the current page displayed and display the next or the previous "page" of lines, respectively. On standard terminals, the Softkey Driven Editor displays 19 lines per text page.

Home Up	(up-left arrow) In the **REVISE*** mode, this moves the current line to the first line in the file, with the cursor in column 1. In the **INSERT***, **TAB***, or **RANGE*** mode, the current line becomes the line just before all other lines in the file, with the cursor in column 1. In the Command mode, "home up" moves the current line to the START line; the cursor remains on the command line.

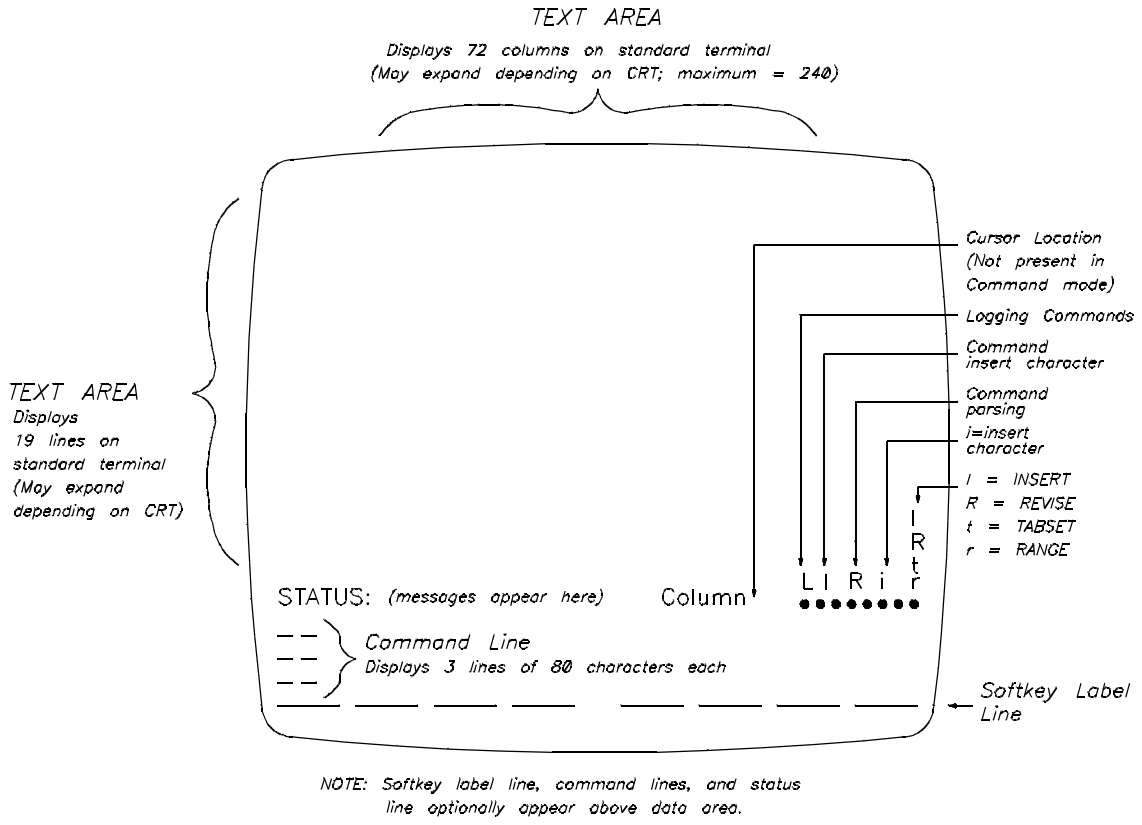
TWO-KEY FUNCTIONS (require pressing two keys)

Home Down	(Shift up-left arrow) In the **REVISE*** mode, this moves the current line to the last line in the file, with the cursor in column 1. In the **INSERT***, **TAB***, or **RANGE*** mode, the current line will follow all other lines in the file, with the cursor in column 1. In the Command mode, "home down" moves the current line to the last line of the file, or to the START line if the file is empty; the cursor remains on the command line.
Backtab	(usually Shift Tab keys) In the **INSERT*** or **REVISE** mode, this moves the cursor to the left from its present location to the previous tabstop. In the Command mode, Shift Tab moves the cursor to the start of the previous keyword (shell command or editor softkey entry). This can save you time when you need to make changes on the command line.
Clear Line	This uses the "kill" character from your operating system stty setting (typically **CTRL u**) to clear the current line, REGARDLESS OF WHETHER OR NOT THE KEYBOARD HAS A CLEAR LINE key. (See "Clear To End Of Line" under One-Key Functions earlier in this chapter, which explains that the Clear Line key clears FROM the cursor position to the END of the line and NOT necessarily the entire line.)

Recall > >	This uses **CTRL r** to recall previous commands from the newest to the oldest (in the buffer), and places the recalled command on the command line.
Recall < <	This uses **CTRL b** to recall previous commands from the oldest to the newest (in the buffer), and places the recalled command on the command line.
Exit Editor	Pressing **CTRL d** will allow you to exit a session without saving any text or commands. Pressing **CTRL d** from any editor mode refers you to the question on the command line: "Do you really want to quit?". If you answer "yes" and press Return, you will exit the Softkey Driven Editor to your operating system shell (current file contents are NOT saved). If you answer "no", the message "No changes lost, edit resumed" appears and your session continues.
Redraw Display	Using **CTRL l** (lower-case L) will refresh the on-screen display from current memory in case the display becomes scrambled ("garbage") from data transmission problems.
Control Char-acters	**CTRL v** can be used to enter control characters into the editor. A control character is entered by first pressing **CTRL v**, followed by the control character itself. For example, to enter the control character **CTRL l** (lower-case L) into the editor, you would press **CTRL v**, and then **CTRL l** (lower-case L), entered by pressing the CTRL key and l (lower-case L) at the same time. A control character is displayed as a single " . ". The **WHATCHAR** softkey is available in the **REVISE*** and **INSERT*** modes to determine whether a " . " in text is really a control character or a period. See **WHATCHAR** syntax in chapter 5 for more information.

Display Format Description

Displays on your terminal screen will provide you with a good picture of what is happening in real-time with your edit session. There are four sections in the display: (1) softkey label line, (2) STATUS line, (3) text entry area, and (4) command line. Each of these sections provides useful information about the session, and these sections are described in the following paragraphs. Refer to the following figure for all of the descriptions.



CRT Display Format - Softkey Driven Editor

Softkey Label Line

These labels correspond to the special set of eight keys on the top row of the terminal keyboard. There are four levels of softkey labels, each accessed in turn with the **--ETC--** softkey at the right end on each level. The labels are changed to display commands and variables which are entered by pressing the corresponding softkey, and other keys as required, on the terminal. Softkey, command, and STATUS lines can be put at the top of the display with **sk -v** option or with SKTOP shell variable from the operating system shell (see chapter 2). For more information, refer to Editor Softkey Levels in chapter 3 and Editor Command Syntax in chapter 5.

Status Line

The STATUS line provides you with messages related to your edit session, including entry of commands. It displays a message about your session status (such as "STATUS: Editing/users/norm/notes"); softkey prompts (such as "ENTER: An operating system file name"); and error messages (such as "ERROR: syntax error"). Editor status messages are summarized in appendix A. STATUS, command, and softkey lines can be put at the top of the display with **sk -v** option or with SKTOP shell variable from the operating system shell (see chapter 2 for more information).

Other important information also appears on the STATUS line. A "Column" label indicates the column number where the cursor is currently located in the **INSERT*** or **REVISE*** mode (column number can be turned off with the **colm_num** command).

At the right end of the STATUS line is a series of eight indicator dot positions related to the editor modes. As you change to the various editor modes, you will see enunciator letters as reminders to you of the current status of the editor. The letters appear only in the 1st, 2nd, 4th, 6th, and 8th positions of the dots. Refer to the previous figure for the following explanation of each letter.

For the Command mode:

Letter "L" in the left-hand dot position shows that commands are being logged to a file.

Letter "I" in second-from-left dot position shows Insert Char is on.

Letter "R" in fourth-from-left dot position shows that entries on the command line are syntactically correct and could be executed at that point by pressing the Return key.

For the other modes (**INSERT***, **REVISE***, **TABSET***, and **RANGE***):

Letter "i" in sixth-from-left dot position shows Insert Char is on.

Letter "I" in right-hand dot position shows session is in the **INSERT*** mode.

Letter "R" in right-hand dot position shows session is in the **REVISE*** mode.

Letter "t" in right-hand dot position shows session is in the **TABSET*** mode.

Letter "r" in right-hand dot position shows session is in the **RANGE*** mode.

Text Area

The displayed text area consists of 19 lines and 72 columns on a typical terminal. The area displayed may be expanded on bit-mapped displays of the HP 9000 Series 300/400 and Series 700 computer without the window manager environment. The first eight columns on-screen consist of blanks, a line number (as applicable), and a line label as follows:

>	Designates the current line in the file being edited.
Line #	Shown as a decimal number to indicate the sequence of lines in your file or its current sequence if a renumber command has been given.
START	Indicates the beginning of the text file; precedes first line of text.
NEW	Designates lines added during the current edit session which have not been renumbered by the renumber command.
END	Indicates the end of the current text file.
TAB	Line is displayed when the tabset command is executed. Shows T where current tabstops are set.

RANGE When **range** is executed, a line of R's (RRR...RR) indicates the current range of columns.

Command Line

The command line is accessed through the Command mode and is used to execute Softkey Driven Editor commands and shell commands. The command line shows only three lines on-screen at one time, although you can continue to enter commands far beyond those three lines of 80 columns each. Individual commands are separated by a semicolon (;), and comments are entered by using a number sign (#) preceding each comment. Command, STATUS, and softkey lines can be put at the top of the display with **sk -v** option or with SKTOP shell variable from the operating system shell. Also refer to Editor Functional Modes in chapter 3 for more information about use of the command line. The Command mode is explained in chapter 5, including syntax entries for **INSERT*** and **REVISE*** modes which temporarily access the command line when a command softkey is pressed.

Creating, Saving, Ending, and Modifying Files

No matter whether you are creating a new file, editing an existing file, saving either new or old files, or ending a file, you will be given various options on softkeys or will be asked to answer questions aimed at avoiding problems. If you will first study and learn the editor functional modes covered in chapter 3, understand the keyboard and display in this chapter, and become familiar with the editor command and variable syntax descriptions in chapter 5, your very first edit session on the Softkey Driven Editor will be both a continuing learning process and a success.

The following paragraphs describe some of the basic procedures and steps you will need to follow or observe as you create, save, end, and modify files using the Softkey Driven Editor. Because text or command file entry is an individual activity tailored to your needs, with frequent options and branches, step-by-step procedures are not possible to detail. The following are reminders and very general procedures for creating, saving, ending, and editing.

Creating a New File

When you enter the editor from the operating system shell and do not have a file name, you are placed in the **INSERT*** mode with an empty file, on a NEW line. The edit cursor is at the start of the line. Your first keystroked character will appear on the display and you can continue entering up to 240 characters on that line. Each time you press Return, another NEW text line appears. You may wish to do some formatting for your text entry with **tabset** and **autotab**, and decide whether or not to display column numbers (a convenience, but, if on, it could affect data transfer rates).

If you wish to execute an editor command during your entry of the new file, choose the command(s) from the softkeys available and press the softkey(s). You will exit temporarily from **INSERT*** mode to the Command mode, where the command keyword appears. Complete the command entry, press Return, and the command is executed (if syntax was okay) and you are put back in the **INSERT*** mode, where your text entry continues. If you also selected the command **log** (log_commands) **on**, the command which was executed is entered into the log file you designated. The executed commands can be recalled for use again with **CTRL r** (new-to-old) or **CTRL b** (old-to-new).

You are now ready to save or to end your edit session.

Saving the File

At any time during entry of the file, you can choose to save the current text file by pressing the softkey labeled **save** and giving it a file name. This will place a copy of the file on your system disk, and your edit session continues.

As part of saving the file, you have the choice to do **tab_conv** (tabconvert) or not. If you wish to save disk space or if you are editing a "make" file, then you will want to use the tabconvert option. If you need help on this, see syntax for **save** in chapter 5.

When you have completed entry into the file, it can be preserved with the **end** command (a file name must be given in order for the file to be saved).

Ending the Edit Session

When you have finished entering text or commands, you can press the **end** softkey. To keep the file just entered, it must have a file name.

As part of ending the session, you have the choice to do **tab_conv** (tabconvert) or not. If you wish to save disk space or if you are editing a "make" file, then you will want to use the tabconvert option. If you need help on this, see syntax for end in chapter 5.

After executing **end**, the edit session ends and you return to the shell.

Changing an Existing File

To make changes in an existing file, you can access it from the operating system shell or from the editor. In either case, you can enter changes by using commands and the keyboard functions defined earlier in this chapter.

From the operating system shell, you would invoke the editor and the file by entering the command "**sk**", with any options you choose, followed by the name of the file. From the editor command line, you enter **edit**, followed by the name of the file, and press Return. From the command line, you can execute commands or use the **INSERT** or **REVISE** mode. When changes are completed, you **end** the file.

Entering and Editing Command Files

Another of the useful features of the Softkey Driven Editor is command files and the logging of commands during an edit session. Command files, which are more fully described in the User's Guide, allow you to create a file of commands and invoke this file as if you had just entered them. Another facility of command files is that you can use parameter substitution to make a command file meet more general applications.

Note

You must have installed User Interface Software (also referred to as "pmon") in order to have full use of command files, logging commands, and changing directories. See "Installing Editor Software On Your Operating System" in chapter 2 for information about where the User Interface Software is available.

There are two approaches you can take toward creating command files. One approach is to use the **log** (log_commands) command to store commands executed during an edit session in a file. This log file then can be edited and modified as needed to make it into a command file. Some changes you might make are: adding comments to the command file (by prefacing the comments with "# "); and adding parameters which will make the command file more generally useful. To add parameters, a PARM line must be inserted at the start of the command file with a list of parameters to be used. Also, parameters must be inserted into the command file as you wish to use them.

The alternative approach to making a command file is to enter the commands as text during an edit session. The resulting file can be saved and executed later as a command file.

When you use either approach, some key points should be remembered:

1. Enter the entire command, not just the abbreviation on the softkey. All commands must be syntactically correct. An advantage of logging commands is that they must be syntactically correct in order to be logged.

2. Only commands which are executed on the command line are logged. Text entered or modified in the **INSERT*** or **REVISE*** mode, as well as keystrokes, such as keyboard cursor movement and delete line, will NOT be logged.

3. Some control characters cannot be generated using the keyboard. These control characters are: **CTRL s** and **CTRL q** (used for terminal/computer handshakes); and **CTRL c** and **CTRL |** (both signals). The only way these characters can be created is to use their octal value as part of a **replace** command so the character is inserted into the text. An example of creating a **CTRL c** character in a file might be:

   replace "aword" with "aword\003"

4. Other control characters can be created on the command line by using the **CTRL v** sequence before the control character itself. This will create a " . " on the command line which cannot be differentiated from a period (if you edit a command file, you can use the **WHATCHAR** softkey to find the character and its numerical equivalent). An alternative way is to use the octal equivalent for any character in a **find** or **replace** command, as in the preceding example.

5. Some strings in command files will need to delimit quote characters within the string. If the quote characters around the string are the caret (^), then any double or single quote characters within the string should be delimited with a backslash. It is not necessary to delimit quotes if the entire string is defined with single or double quote characters. Not delimiting the quote characters in a caret-quoted string can cause following lines to be concatenated to the line containing the string, which will probably result in syntax errors when the command file is executed. An example of correctly-delimited commands might be:

```
replace ^ hello^  with ^ Check the \" in this string^
insert 'This is another " character without delimiter'
```

Installation Notice

Editor Command Syntax

Chapter Overview

This chapter provides the following information:

- A description of system features which allow easier entry of commands.
- A summary of editor commands.
- Definitions of conventions used in the syntax diagrams.
- Functional descriptions, examples, and parameters for syntax of all editor commands.

Easier Entry of Commands

Several features have been designed into the Softkey Driven Editor which make it easy to use and quite compatible with your operating system. These features are listed and described in the following:

- **Softkeys.** Softkeys, with labels displayed optionally either at the top or the bottom of the screen, are used to enter nearly all commands. These defined softkeys provide fast command entry, and minimize errors.
- **Command Completion.** You need to type only the first character(s) of a command keyword (enough to uniquely identify the keyword), and press the Tab key. The system will then complete the keyword for you.
- **Command Line Recall.** Commands may be recalled from a buffer by pressing **CTRL r** (from newest to oldest command) or **CTRL b** (from oldest to newest command).
- **Command Line Erase.** Allows corrections by erasing the current command line with the "kill" character from your operating system stty setting, then entering the correct command.
- **Multiple Commands On One Line.** You can enter more than one command on the same command line by separating the commands with a semicolon (;).

- **Change Directory.** You can change your operating system directory while in the Softkey Driven Editor by using the cd command (hidden; not on a softkey).
- **Operating System Filters and Pipes.** You can specify operating system filters and pipes as the destination for information with the **list** command. See the description of **list** command in this chapter for details.

Command and Variable Summary

Editor commands and the variables used with the commands are summarized in the following table. The table also identifies those commands and variables which are "hidden", that is, accessed by typing them in from the keyboard.

Commands and Variables Summary

COMMANDS	
autotab	log_commands
cd (change directory) *	merge
column_numbers	range
copy	renumber
delete	repeat
edit	replace

Commands and Variables Summary (Cont'd)

COMMANDS	
end	retrieve
extract	save
find	split
help	tabset
insert	wait*
join	while
LINE+ -	! *
list	! ! *
VARIABLES	
!CMD!	LINE #
CMDFILE*	PARMS*
COLUMN	POINT
COUNT	SPACES
DIR	STRING
FILE	# TIMES
LIMIT	TIME*
LINE+ -	WHATCHAR
* Indicates a hidden command or variable; must be entered from keyboard	

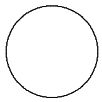
Syntax Conventions

The conventions used in the command and variable syntax diagrams in this chapter are as follows:

This symbol indicates a command keyword entered by pressing a softkey. The keyword is shown as it appears on the command line and may not be the same as the softkey label.

This symbol contains either prompts indicating that parameters must be entered from the keyboard or else it references additional syntax diagrams. Softkey prompts are enclosed by the "< " and "> " delimiter symbols and are shown exactly as they appear on the softkey label.

References to additional syntax diagrams may be shown in uppercase characters with no delimiter symbols.



A circle denotes operators and delimiters used in expressions and command lines.

Whenever keywords entered from softkeys appear in text or examples, they are shown in boldface italics, for example, **copy**. Command parameters which are entered from the keyboard are shown in standard type.

Introduction to Softkey Driven Editor Syntax

Using the Softkey Driven Editor involves a combination of entry from the keyboard of your terminal and entry from the softkeys which have different functions as their labels change. It is the syntax (that is, the structure and relationship) of the keyboard and softkey entries which enables your most efficient use of the editor.

Command Line Entries

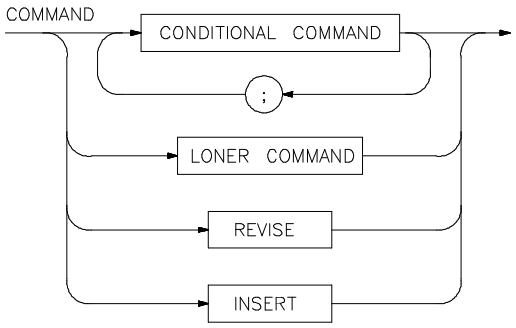
To help understand the syntax for the first entry you make on the command line, you should become familiar with the diagrams for "Command", "Conditional Command", and "Loner Command" which follow the next paragraph. Note especially that only the Conditional Commands may be used to construct multiple commands.

Syntax for Variables and Commands

Descriptions and syntax diagrams (where applicable) for commands and variables are given separately, in two groups, with each group given in alphabetical order (see group lists in the previous table). These groupings are shown after the syntax descriptions for Conditional and Loner Commands. For each group, the syntax heading appears as a major heading preceding the information associated with it.

command

Syntax



Function

The Command mode allows you to enter either loner (single) or conditional commands on the command line, which then may be: (1) invoked from the editor; (2) logged to a logfile for subsequent use; or (3) edited for desired changes.

Default Value

Waiting for softkey, keystroke, or Return.

Examples

Refer to the syntax diagrams for individual commands.

Parameters

Refer to the syntax diagrams for individual commands.

Description

Editor commands or shell commands (preceded by "!" exclamation point) may be executed from the command line. Also, comments may be entered on the command line by preceding them with a "# " (number sign). From the command line, you can enter either the **INSERT** or **REVISE** mode by pressing the corresponding softkey. For more information on the conditional and loner commands, refer to the two syntax diagrams following this one. For information about temporarily entering the Command mode from either **INSERT*** or **REVISE***, refer to the command syntax for each mode and to the "Editor Functional Modes" heading in chapter 3.

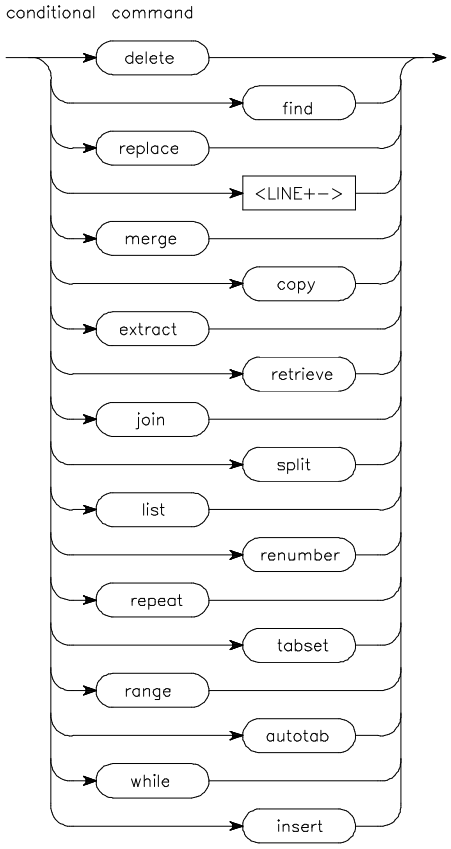
HP 64000-UX can expand shell variables on the command line and also in command files. Only those shell variables beginning with "$" followed by an identifier will be supported. An identifier is a sequence of letters, digits, or underscores beginning with a letter or underscore. The identifier may be enclosed by braces, "{}", or entered directly following the "$" symbol. Braces are required when the identifier is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. Expansion of shell variables involves the replacement of the shell variable by the definition of that shell variable. If the shell variable is not defined within the user's environment, the shell variable will be replaced by a null string.

For example, assume a directory named /users/softkeys exists, and the shell variable "S" is used to represent the word "soft". By specifying the directory as /users/${S}keys, the correct result is obtained. However, if you attempt to specify the directory as /users/$Skeys, HP 64000-UX looks for the value of the variable "Skeys". This is not the result we expected. You will not get the intended result unless "Skeys" is already defined to be "softkeys". To include "$" as part of a command, you must put a backslash (\) in front of the "$". Otherwise, the system will try to expand the shell variable which is designated by "$" and any text that follows it.

You can examine the current values of all shell variables defined in your environment with the UNIX command "env".

conditional command

Syntax



Function

A conditional command is either invoked from the editor or used to build a multiple command for execution within a while command. Conditional commands return a truth value (true or false) as summarized in appendix B.

conditional command

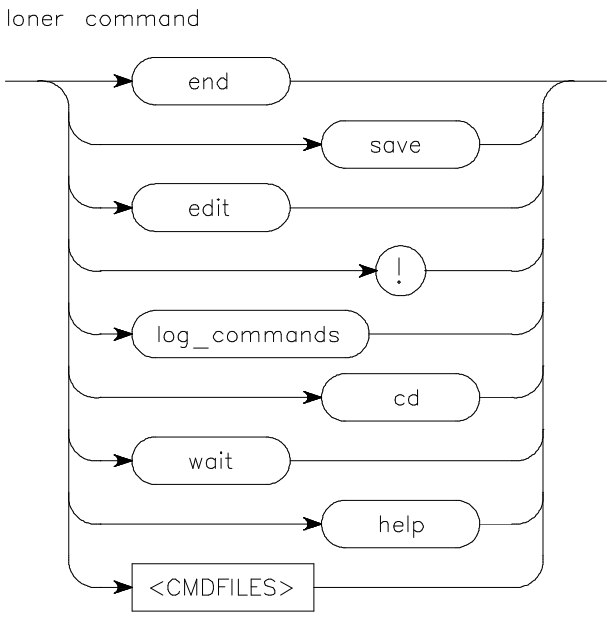
Default Value,

Examples, and

Parameters Refer to the syntax diagrams for individual commands.

loner command

Syntax



Function

Loner commands are invoked as single commands and may not be used in constructing multiple commands for execution within a while command. Loner commands do not return a truth value.

Default Value, Examples, and Parameters

Refer to the syntax diagrams for individual commands.

< !CMD!>

Syntax Example(s) **list** < **!CMD!**>

Function

< **!CMD!**> is a prompting softkey which appears with the list command. When it appears, you may enter a shell command, preceded and followed by !, which, if then executed, would list file text to that shell command. This command may be any type of input. The user should be sure that the command entered is correct. Note that this explanation applies only to the softkey labeled < **!CMD!**> and not to the invoking of a shell command by use of the exclamation point "!", which is covered in the command syntax section.

Default Value

Prompts for entry of a shell command, preceded and followed by "!".

Example(s)

list ! col -x | lpr -q !

CMDFILE

Syntax Example(s) CMDFILE (command file)

Function CMDFILE stands for "command file", and it represents a hidden command which is entered from the keyboard. You can type on the command line the name of a command file (source file) which contains a series of valid shell commands. Note that additional software is required in order to use command files with the Softkey Driven Editor; for information on this, see cmdfile command syntax Description in this chapter or the "Installing Editor Software Your Operating System" heading in chapter 2. For more information on CMDFILE variable, see Description under this same syntax heading.

Default Value Displays prompting softkey < **PARMS**> (PARAMETERS) when command file needs parameters.

Example(s)
```
GETSAMP
FIRSTTEST PARM1 PARM2 PARM3
```

Description Command files can be initiated from any of the development environment features, including the Editor. Command files do not stop processing until either: the end of the command file; the last feature is terminated; a syntax error occurs; or a signal (such as SIGINT) interrupts the command file. You can invoke a command file from within another command file, but the original command file will wait for the second command file to be completed before continuing. (This is different from the previous editor for the HP 64000 Logic Development System. Refer to appendix C for a description of these differences.) Command files may not be used to construct a complex command during an edit session.

< COLUMN>

Syntax Example(s)

split at column < **COLUMN**>
range < **COLUMN**> **thru** < **COLUMN**>

Function

< **COLUMN**> is a prompting variable for entry of any column number in the edit file related to a command to be executed. You can enter any positive integer from 1 through 240.

Default Value

none

Example(s)

split at column 5
range 5 **thru** 15

< COUNT>

Syntax Example(s) **while count < COUNT>**

Function **< COUNT>** is a prompting variable for a repetition count. It is used as a loop counter with the while command. You can enter any positive integer; if no value is used, defaults to 1.

Default Value If no value is used, defaults to 1.

Example(s)
while count 37 **do delete doend**
while count do insert "hello" **doend**

< DIR>

Syntax Example(s)

cd < **DIR**>
(Note that the shell command cd for "change directory" is a hidden command, and must be entered from the keyboard. When cd is entered, < **DIR**> appears on a softkey).

Function

< **DIR**> is a prompting softkey for a new directory name. It allows you to specify a different directory and use "cd" to move the edit session from one directory to another. The < **DIR**> softkey appears only when the hidden command "cd" is entered from the keyboard. **!**cd will NOT change the directory of the edit session, that is, where the current text file will be saved.

Default Value

Defaults to user's HOME directory.

Example(s)

cd newdir

< FILE>

Syntax Example(s) **list < FILE> help**

Function

The variable **< FILE>** is a prompting softkey for entry of a file name. Simple file names are easily parsed by the scanner of the Softkey Driven Editor. However, complex filenames, such as ones which include a hyphen (-) or certain other characters, may require use of quotation marks or an escape sequence to avoid a syntax error. For more information, see Description under this same syntax heading.

Default Value

Prompts for file name.

Example(s)

edit oldfile **into** newfile

Description

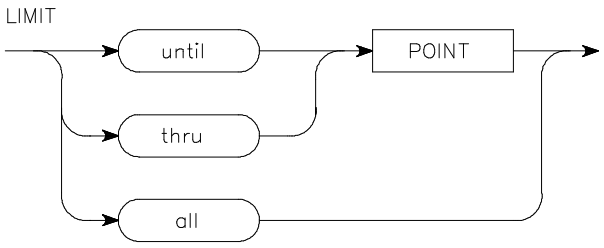
Since some complex file names are not readily parsed by the editor, you may have to either enclose the file name in quotes (as a string, with no spaces) or use the backslash (\ ) escape character. Examples of using an escape character would be \3file (to escape the 3) and a\-file (to escape the -). One way to determine if quotes or an escape may be needed is to enter the command and see if a syntax error occurs. Another way to determine this is to watch the "R" enunciator (at the right-hand end of the STATUS line, fourth dot position from the left); if that "R" is present, the editor is still parsing and the **< FILE>** should be accepted. File names may be either absolute or relative. An absolute file name begins with a "slash" character, "/", and this indicates that the path starts at the root directory. A relative file name does not begin with a "/" and the path is relative to the current directory.

As noted earlier, shell variables may be used as part of a filename if the shell variable has been defined and exported before the edit session. To use the shell variable, just enter the shell variable as part of the filename. The '/' delimiter will terminate the variable name.

If other text immediately follows the shell variable, then the variable name can be enclosed in braces, '{}'. Only the variable name will then be removed and the following text will be appended to the replacement text. Also, since the '$' is used to signal the start of a shell variable name, the user may have trouble finding filenames that include such a character within it. Such characters must be escaped with a backslash. Otherwise, the editor will try to search for a shell variable with that name and probably replace the name with a null string.

LIMIT

Syntax



Function

The variable LIMIT represents the **thru** or **until** options as used with the variable POINT, or as used with the **all** option. These options are used to define a line range over which a command is to operate. For more information, see Description under this same syntax heading.

Default Value

none

Example(s)

thru start
until + 5
thru "function"

Description

The range for LIMIT extends up or down from the current line, depending on the options used. The keyword **thru** is used to include the line indicated in the LIMIT. The keyword **until**, however, will include up to, but not including, the line indicated in the LIMIT. Also, if the indicated LIMIT exceeds the file boundaries, then either the start or the end of the file will be used for that boundary, even if the keyword **until** is used. The keyword **all** includes the entire file as the range. Refer to the variable POINT for more information about variables such as **< STRING>** and **< LINE+ ->** , which are used to complete a LIMIT statement.

< LINE+ ->

Syntax Example(s) **find** < **STRING**> **thru** < **LINE+ ->**

Function < LINE+ -> variable is a prompting softkey which allows you to define a POINT for use with LIMIT in a command being executed relative to the current line. (< LINE+ -> also appears as a command prompt softkey, allowing you to move to desired lines in a file being edited; refer to syntax for commands.) For more information, see Description under this same syntax heading.

Default Value none

Example(s)

find "hello" **thru** + 35
replace 'A' **with** 'B' **thru** 22

Description If you use **< LINE+ ->** to specify a relative line outside the file boundaries (that is, a line position which would be greater than the number of lines preceding or following the current line), then either the first (if -) or the last (if +) line of the file becomes the current line; a message tells you the number of lines moved.

If an absolute (unsigned) number greater than the current line number is specified and that line does not exist, the next higher-numbered line from the sought line is first found, and the line before that one (whether numbered or NEW) becomes the new current text line. Similarly, if an absolute number lower than the current line number is specified and that line does not exist, the next lower-numbered line from the sought line is first found, and the line after that one (whether numbered or NEW) becomes the new current text line. If the sought line is below or above the file boundary, the START line or the last line, respectively, will be used as the new current line.

< LINE # >

Syntax Example(s) **merge** < **FILE**> **from** < **LINE #** > **thru** < **LINE #** >

Function

The variable < **LINE #** > is a prompting softkey, which appears only with **merge** to provide a start or stop point to a file being merged. Any existing line in the file may be used, providing that the **from** line number is lower than the **thru** number. For line values outside file boundaries, see **merge** command syntax.

Default Value

from < **LINE #** > : Defaults to first line of < **FILE**> .
thru < **LINE #** > : Defaults to last line of < **FILE**> .

Example(s)

merge afile **from** 5 **thru** 37
merge /tmp/bfile **thru** 10

< # LINES>

Syntax Example(s) **join** < **# LINES**>

Function Variable < **# LINES**> is a prompting softkey, appearing only with the **join** command to specify how many lines after the current line are to be joined into a single line. A **join** command fails if the number of characters (including spaces) in the lines to be joined exceeds 240. Also refer to **join** command syntax in this chapter.

Default Value none

Example(s) **join** 2

< PARMS>

Syntax Example(s) "Command File" < **PARMS**>
("Command File" represents a hidden command which must be entered from the keyboard.

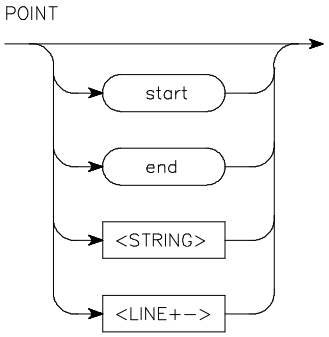
Function The variable < **PARMS**> is a prompting softkey which stands for "parameters". It is accessed only when you type in the name of any command file (source file) with a series of valid shell commands. As you type in the name of the command file, if it needs parameter(s) text, the Softkey Driven Editor will prompt you with the softkey labeled < **PARMS**> . If a space is required as part of a parameter, the parameter should be placed within quotation marks.

Default Value none

Example(s) FIRSTTEST < **PARMS**>

POINT

Syntax



Function

The variable POINT represents the options **start**, **end**, **< STRING>** , **< LINE+ ->** , used with the variable LIMIT (**thru**, **until**) to complete the definition of a range over which a command is to operate. For more information, see Description under this same syntax heading and the syntax heading for **< STRING>** .

Default Value

none (see Description below for definitions of **start** and **end**)

Example(s)

thru < STRING>
until < LINE+ ->

Description

The keyword **start** refers to the first line in the file. Similarly, **end** refers to the last line in the file.

< STRING> means any ASCII string and it is a pattern-matcher, such as that used in the **find** command. However, **< STRING>** used as a POINT is slightly different in that it will match anywhere on a line, regardless of the current range setting. The lines searched will always begin with the line following the current line and continue through the end of the file. Use of **< STRING>** as a POINT will not affect a previous **< STRING>** used in either a **find < STRING>** or **replace < STRING>** command.

< LINE+ -> as a POINT has the same meaning you will find under its separate syntax as a variable, which is: Any line number in the file or a signed offset from the current line; and the specified line may be above or below the current line. (**< LINE+ ->** also appears as a command in editing text; see its command syntax for more information.)

< SPACES>

Syntax Example(s)

edit a file **into** bfile **tab_conv** < **SPACES**>
tabset fixed < **SPACES**>

Function

The variable < **SPACES**> appears as a prompting softkey for the **tab_conv** and the **tabset** commands. Your entry is especially important for the tab-conversion option since you must choose whether or not to preserve any tab characters used as control characters. More information about tab-conversion is in chapter 3 and in the **edit** and **end** command syntax. For **tabset fixed**, < **SPACES**> (values 1 through 239) lets you specify where fixed tabstops are located.

Default Value

tabset fixed < **SPACES**> : If no entry, defaults to 16 starting at column 1 (1, 17, 33, etc.).
tab_conv < **SPACES**> : If no entry, defaults to 8 spaces per tabstop.

Example(s)

tabset fixed 4
end afile **tab_conv** 4

< STRING>

Syntax Example(s)

find < **STRING**> LIMIT
replace < **STRING**> **with** < **STRING**> LIMIT

Function

The < **STRING**> variable is a prompting softkey for you to enter a series of ASCII characters delimited on both ends by either: double quotes ("); single quotes (’); or carets (^). A string defined in this way is used with commands such as **find**, **insert**, **replace**, and **split**, and with LIMIT variables **thru** and **until**. A string can include the delimiter if it is preceded by a backslash (\ ) as an escape character. Other characters normally can be used without change in a string. The way a string is used depends on the command and on the LIMIT. For more information, refer to Description under this same syntax heading and to the syntax for **find** and **replace**.

Default Value

Depends on command usage; refer to syntax for individual commands.

Example(s)

find "ABC" **all**
replace "ABC" **with** "DEF" **thru end**

Description

When < **STRING**> is used with the **insert** command, all characters in the string are inserted into the edit file on a new line (for escaped characters, only the character itself is on the new line).

< **STRING**> , when used with a LIMIT or **find** or **replace** command, is basically a pattern-matcher which can look for a match of known, or even unknown, characters on a line. A simple example would be to change all occurrences of the word "apples" to the word "oranges" by stating **replace** "apples" with "oranges" **thru end**. Other characters used in a matching string also will be taken as literals and must match exactly in the text, as with "apples". Use of strings as pattern-matchers is made more powerful and versatile in the Softkey Driven Editor by use of "anycharacter" and "anystring". The definitions of what can be specified in a pattern-match are as follows:

anycharacter: Using anycharacter lets you match any single character. After the first delimiter of a string is entered, the anychar softkey appears.

Pressing this softkey places a ? in the string; the "?" may occur anywhere within the string. This type of match may be used where there is a fixed number of intervening characters, or if you do not care what character is matched. The replace command uses both a matching string and a replacement string. For its operation with "?", refer to the replace command syntax in this chapter.

anystring: Using anystring lets you match any number of characters, including none (the null string). The preferred match is the smallest number of characters. After the first delimiter of a string is entered, the anystring softkey appears. Pressing this softkey places a * in the string, representing anystring, which may occur anywhere within the string. An example of using anystring is to find or replace occurrences of strings which begin with the letter "a" and end with a period "." . The pattern would be "a*." . The replace command uses both a matching string and a replacement string. For its operation with "*", refer to the replace command syntax in this chapter.

In the **replace** command, it is not valid to have a pattern-matcher which is either empty or which consists entirely of anystrings (for example, "**"). However, this pattern-matcher is allowed for the **find** command since it is the equivalent of the null string (which always matches at the start of the current range).

A backslash (\ ) can be used as an escape which will allow finding occurrences of * , ? , or $ in a file. For example, if the exact string sought is a*b and this string is used for the pattern-matcher, a match would occur for any string consisting of the letter "a", followed by any number of other characters, and ending with the letter "b". However, if the string "a*b" is used, the match will be exactly a*b . If the string to be matched includes a \ , it must be escaped, as in "\\". If any other character preceded by a \ , for example, "\a", is to be matched, a second backslash must be added, making "\\a". Use of the backslash to escape characters for pattern-matching is the same for the first string used with the **replace** command; refer to the **replace** command syntax.

Using an octal number in a string is a special case which requires the backslash as an escape. The octal string can be used in either the matching or the replacement string, represented by "\nnn", where the first digit can be 0 through 3, the second and third digits from 0 through 7. All three digits must be used for recognition of the string. An example to replace ASCII character DEL (octal 177) with ESC (octal 033) is: **replace** "\177" **with** "\033". You can enter control or other nondisplayable characters in octal equivalent for **find** or **replace** strings.

< # TIMES>

Syntax Example(s) **repeat** < **# TIMES**>

Function The Variable **< # TIMES>** is a prompting softkey which appears with the **repeat** command to allow specifying the number of times the current text line is to be duplicated and inserted into the text.

Default Value 1 (that is, one time)

Example(s) **repeat** 2

< TIME>

Syntax Example(s) **wait < TIME>**
("wait" is a hidden command which must be entered from the keyboard. When wait is entered < **TIME**> appears on a softkey.)

Function The Variable < **TIME**> is a prompting softkey which appears only with the "wait" command and is used to specify how long execution of a command file is halted before it continues. Time for the temporary halt is entered in seconds. When the command file is halted, pressing CTRL c overrides the specified time, and processing starts again.

Default Value Will wait for SIGINT signal (pressing CTRL c at the same time) to continue.

Syntax Example(s) wait 10 seconds

WHATCHAR

Syntax Example(s) **REVISE***, **--ETC--**, **--ETC--**, **--ETC--**, **WHATCHAR**

Function

WHATCHAR is a softkey which can be accessed from either the **INSERT*** or **REVISE*** mode to allow you to determine quickly what character is actually located under the cursor. When **WHATCHAR** is pressed, the character under the cursor is identified on the STATUS line, along with its decimal, hexadecimal, and octal values. The first value is followed by a "D" to denote the decimal equivalent of the character. Similarly, the second value is prefixed with "0x" to denote the hex equivalent, and the third value is prefixed by a "0" to denote the octal value. For more information, see Description under this same syntax heading.

Default Value

none

Example(s)

The following examples require access from either the **INSERT*** or the **REVISE*** mode, then locating the cursor under the character, and pressing the **WHATCHAR** softkey:

Example 1. If the character under the cursor is a control character, such as CTRL d (which appears as a **.** on the display), the STATUS line reads:

Control character (4D, 0x4, 0004)

Example 2. If the character under the cursor actually is a period (a "." on the display) the STATUS line reads:

Normal character **.** (46D, 0x2e, 0056)

Example 3. If the character under the cursor is a normal character, such as a "Z" (a Z on the display), the STATUS line reads:

Normal character Z (90D, 0x5a, 0132)

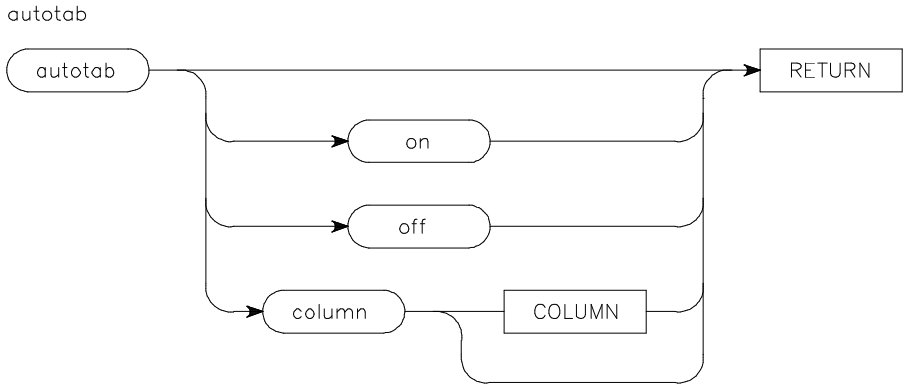
Description

Use of the **WHATCHAR** softkey to determine the character under the cursor is important since most terminals display a "." to represent control characters. Thus you can interrogate the editor to learn what a displayed "." actually represents. For instance, the first example above shows how a displayed "." represents the control character CTRL d, which was entered from the keyboard by use of: CTRL v CTRL d . (Refer to chapter 4 for information about keyboard functions, such as using CTRL v to enter control characters into the editor.)

The Softkey Driven Editor displays the same information about normal characters and may be useful at other times as well.

autotab

Syntax



Function

The **autotab** command allows automatic positioning of the cursor on the current line when using the **INSERT** or **REVISE** mode. Location of the cursor depends on whether or not a **< COLUMN>** number has been specified and on whether or not there is text on the current line, or, if the current line is blank, on previous lines. For more information, see Description under this same syntax heading.

Default Value

autotab column < **COLUMN>** : If no column number entered, defaults to the previous column selection or to 1 if no previous value was used.
autotab only: **autotab** status toggles, that is, alternates between **on** and **off** each time selection is made and "Return" is pressed.

Example(s)

autotab
autotab column
autotab column 37
autotab on (or **off**)

Description

When **autotab column** is selected and a **< COLUMN>** number is specified (1 through 240 allowed), the cursor is returned to that column. For **autotab** column numbers larger than 72, the display is shifted so the specified column and cursor are in the center of the display.

When **autotab** is turned **on** without pressing **column**, the position of the cursor depends on whether you are in the **INSERT*** or **REVISE*** mode, as follows:

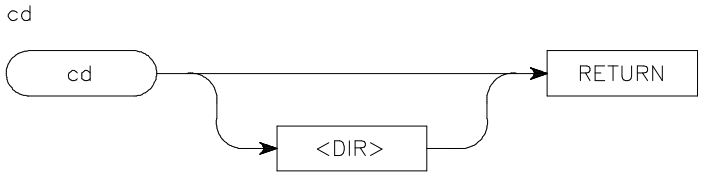
INSERT* : Cursor is placed in the same column as the first nonblank character in the next lower-numbered, nonblank line.

REVISE* : Cursor is placed in same column as first nonblank character on current line. If current line is blank, cursor is placed in same column as the first nonblank character in next lower-numbered, nonblank line.

When autotab column is selected and no **< COLUMN>** number is specified, the cursor is placed in the same column specified previously, or, if no column was specified previously, the cursor is returned to column 1 as a default. Each time **autotab** only is selected and Return is pressed, the **on**/**off** setting toggles, that is, alternates. Current setting is shown on the STATUS line.

(change directory) cd

Syntax



Function

The command "cd" is a hidden command (that is, not on a softkey) which is entered by typing it from the keyboard. When cd is entered, the softkey **< DIR>** appears as a prompt for you to enter the directory to which you wish to move during the edit session. For more information, see Description under this same syntax heading.

Default Value

If no directory specified for **< DIR>** softkey: Defaults to user's HOME directory.

Example(s)

cd newdir

Description

The cd command operates in the same way as the cd command of the operating system shell. The cd command makes the specified directory become the current directory for the rest of the edit session or until another cd command is executed. Note that doing a cd command in a shell that was invoked from the editor does not affect the current directory of the Softkey Driven Editor.

Also, observe that using the cd command in the Softkey Driven Editor does not change where the current text file will be saved. This is because the current text file is given an absolute path using the current directory at the time of executing the **edit** command. However, doing a cd command will affect **save** and **edit** commands for text files from that point on.

cmdfile (command file)

Syntax Example(s) cmdfile (command file)

Function "cmdfile" stands for "command file", and it represents a hidden command (that is, not on a softkey) which is entered from the keyboard. You can type on the command line the name of a command file (source file) which contains a series of valid shell commands. For more information, see Description under this same syntax heading.

Default Value Defaults to a prompting softkey **< PARMS>** (PARAMETERS) when a command file requires parameters.

Example(s)

```
GETSAMP
FIRSTTEST PARM1 PARM2 PARM3
```

Description

Note Additional software is required in order to use command files with the Softkey Driven Editor. To use command files with the editor on the operating system (installed on an HP 9000 Series 300/400 and 700 computers), the HP B1471 64000-UX Operating Environment software is required. For more information, contact your local HP Sales/Service Office.

Command files can be initiated from any of the operating system features, including the Editor. Command files do not stop processing until either: the end of the command file; the last feature is terminated; a syntax error occurs; or a signal (such as SIGINT or SIGQUIT) interrupts the command file. You can invoke a command file from within another command file, but the original command file will wait for the second command file to be completed before continuing. (This is different from the previous editor for the HP 64000 LDS. Refer to appendix C for a description of these differences.) Command files may not be used to construct a complex command during an edit session.

To execute a command file, you would enter a file name and any parameters the command file might require. If the file name is absolute (begins with " / "), the file name is used "as-is" to find the command file for execution. However,

if the file name is relative (does not begin with a "/"), then a more complex search is made to find that command file. If the HP64KPATH shell variable was defined and exported by the user, then each directory in that shell variable will be searched from left to right until either a valid command file is found or there are no more directories to be searched. If no HP64KPATH shell variable is defined or if it is not exported, then only the current directory is searched. To be a valid command file, the file must be readable, but not executable (that is, the execution bit is not set). If a valid command file is not found, an error message is generated. This method of searching for command files occurs for the initial command file as well as for any command file invoked from within a command file.

For example, given the following HP64KPATH shell variable:

```
HP64KPATH= /bin:.:/usr/bin
export HP64KPATH
```

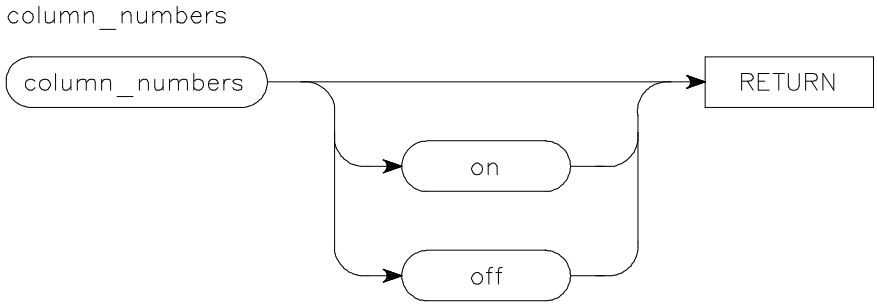
If a relative file name (for example, "cmdfile") was entered, the following search is made:

```
/bin/cmdfile
./cmdfile  < = =  Note that this file name is relative to the
                  user's current directory
/usr/bin/cmdfile
```

If a valid command file is found, the search is stopped and that command file is then executed.

(column_numbers) colm_num

Syntax



Function

The command **colm_num** is a softkey, standing for column numbers, which offers the choice of displaying or not displaying the column number at which the cursor is located during the **INSERT***, **REVISE***, **range**, and **tabset** modes. For more information, see Description under this same syntax heading.

Default Value

If **colm_num** is **off**, and no selection is made: Defaults to **on** (toggles).
If **colm_num** is **on**, and no selection is made: Defaults to **off** (toggles).

Example(s)

colm_num
colm_num on (or **colm_num off**)

Description

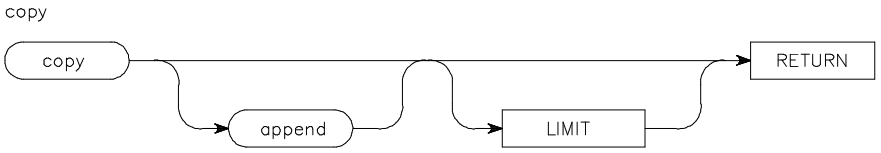
The column number is labeled "Column" and appears along the STATUS line when **colm_num** is turned **on**. If you do not need the column number information, you may wish to turn it **off** to reduce the amount of data being transferred to your terminal, which could improve the data transfer rate.

Note that if **colm_num** is pressed, but no selection of **on** or **of**f is made and Return is pressed, the display of column numbers toggles, that is, it switches to the opposite setting.

Normally, **colm_num** is **on**, but this may be changed by choosing the -c option when you invoke the Softkey Driven Editor from your operating system shell; the -c option will set **colm_num off**.

copy

Syntax



Function

The **copy** command places a copy of a line, or lines, as defined by a LIMIT, from the current edit file, into a temporary storage buffer, allowing you to choose whether the copied lines are appended (added) to the lines already in the buffer or whether the copied lines overwrite (delete) text in the buffer. Copied lines are not deleted from the current text file. (Also, see separate syntax heading for LIMIT variable.) For more information, see Description under this same syntax heading.

Default Value

If no LIMIT is specified: Defaults to **copy** of current line of text only.

Example(s)

copy
copy append thru + 7
copy append until "PRQ"
copy thru start

Description

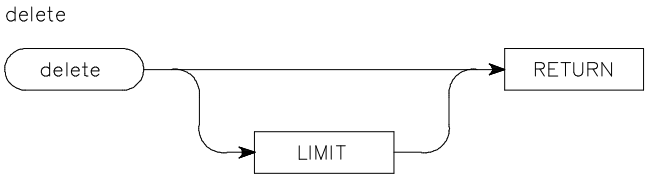
The temporary storage buffer for **copy** is shared with **extract**, and the buffer contents are not protected unless you use the **append** command. Otherwise, if the LIMIT is not found, and **append** is not used, text which was previously in the buffer is lost! The **retrieve** command (see that syntax heading) gives access to the contents of the buffer.

The range for copy includes the current text line as one boundary and the value specified by LIMIT as the other boundary.

Once text has been copied, the last line in the range becomes the new current line of text.

delete

Syntax



Function

The **delete** command allows you to remove a line, or lines, from the current text file, with a range defined by LIMIT (see separate syntax heading for LIMIT variable). The deleted text is NOT recoverable. For more information, see Description under this same syntax heading.

Default Value

If no LIMIT is specified: Defaults to **delete** of current line of text only.

Example(s)

delete
delete until "JNZ"
delete all
delete thru end

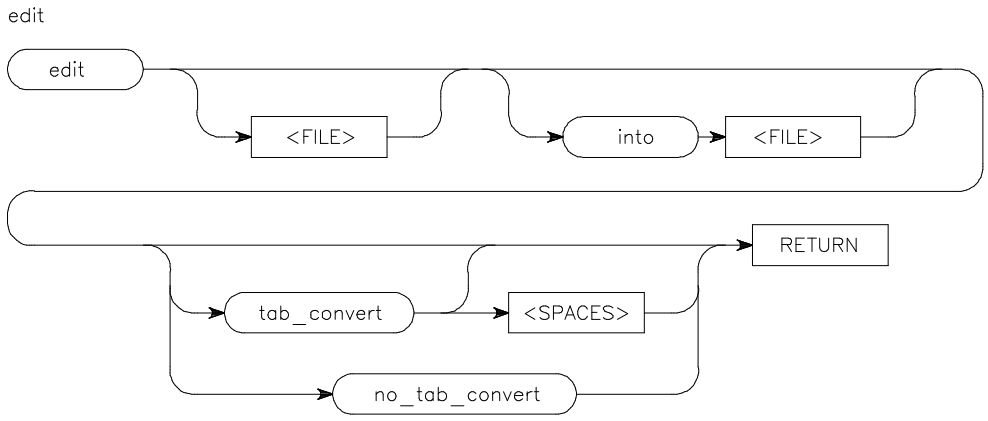
Description

If a LIMIT is specified, but LIMIT is not found, no text is deleted. The range for **delete** includes the current text line as one boundary and the value specified by LIMIT as the other boundary.

Once text has been deleted, the first line of text following the deleted line, or lines, of text becomes the new current line of text.

edit

Syntax



Function

The **edit** command allows you to continue an edit session with a new or an existing file. If **edit** is selected after the current file has been changed, you may either save that file or else specify that those changes be lost before continuing. For more information, especially about the importance of tabconversion, see Description under this same syntax heading.

Default Value

edit: Defaults to a new, empty file with no file name.
edit FILE1 **into** FILE2: No default for FILE2; must specify its file name.
edit afile **tab_conv** < **SPACES**> : If not specified, defaults to 8 spaces per tabstop.

Example(s)

edit
edit file
edit into newfile
edit infile **into** outfile
edit afile **no_tab_c**
edit bfile **tab_conv** 4

Description

When an edit session is started with no file name given, the editor begins in the **INSERT*** mode, with a new line. If the file name given for an edit session is a relative path, the current directory is searched for the file. If the given file name is an absolute path, the specified location in the file system is searched for the file. If the sought file is not found, or is in some way protected from reading, an error message is reported on-screen, and you would begin an edit session with an empty file. When a file name is given for the edit session and no option for tabconversion is selected, tab characters will be converted automatically to spaces, using the rule of one tabstop equals eight (8) spaces.

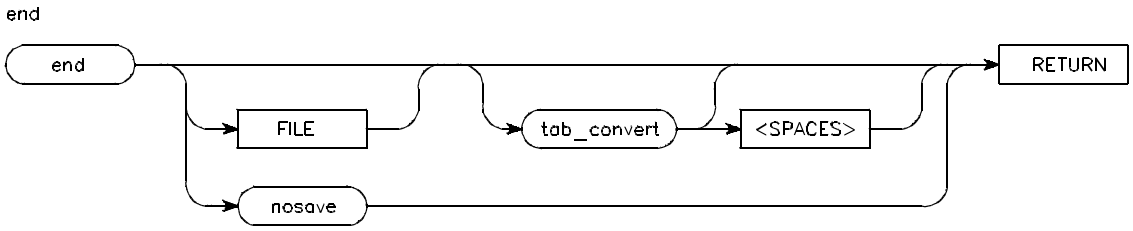
For text where a tab character is to represent a control character, you should use the **no_tab_c** (for no tab-convert) option to prevent the tab character from being converted into spaces. This option may be useful for assembly-type files with embedded tab characters.

The tab_conv option is used when tab characters represent the next tabstop position for the cursor, as determined by the **tab_conv < SPACES>** command, which defaults to tabstops in columns 1, 9, 17, etc. As a line of text is read into the editor, tab characters are converted into the number of spaces needed to move to, but not including, the next tabstop position.

The way in which a file is read into the editor does not affect use of **save** or **end** on that file. The default for writing the file is to do no tabconversion, and the resulting file will not contain tab characters unless they were entered as control characters, or if the edit command included **no_tab_c**.

end

Syntax



Function

The **end** command is used to close an edit session by placing a copy of the current edit file onto the system disk. If, however, the **nosave** option is selected, the edit session ends, without saving the current edit file, regardless of any text changes made. For more information, see Description under this same syntax heading.

Default Value

end: Defaults to destination file from edit command, with no tabconversion.
end tab_conv < SPACES> : If not specified, defaults to 8 spaces per tabstop.

Example(s)

end
end afile
end afile **tab_conv** 4
end afile **tab_conv**
end nosave

Description

When you end an edit session and the file is to be saved, a check is made to see if the destination file named from the **edit** command already exists. If a file with that name already exists, a check is made to see if you have a private recovery directory. If there is a recovery directory, the existing file is moved there before the new file is saved. Also, checks are made to see that the path to the destination file allows for the file to be written; if this is not allowable, an error is reported, and the edit session is continued.

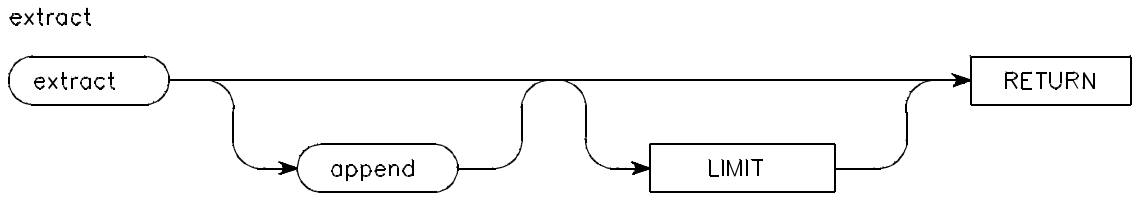
If you specify **end** to a file name other than the current file, a check is made to see if that file name already exists. If it exists, you then must specify whether or not that file should be overwritten. As noted above, if you are overwriting an existing file and a private recovery directory exists, then the existing file is moved to that recovery directory before the new file is saved. If you are in an edit session where the current file does not have a name and you try to end the edit session without specifying a destination file, an error is reported and the session continues.

An edit session will end with no tabconversion unless you specify **tab_conv**. With no tabconversion, tab characters will not be used to replace spaces even though that might save disk space. With no tabconversion, the file can be printed or edited and it will be exactly as it appears in the editor.

For "make" files, use tabconversion so the file is acceptable to the "make" command.

extract

Syntax



Function

The **extract** command removes a line, or lines, as defined by LIMIT, from the current edit file, and places the line, or lines, in a temporary storage buffer, allowing you to choose whether the extracted lines are appended (added) to the lines already in the buffer or whether the extracted lines overwrite (delete) text in the buffer. Extracted lines are deleted from the current text file. (See separate syntax heading for LIMIT variable.) For more information, see Description under this same syntax heading.

Default Value

If no LIMIT is specified: Defaults to **extract** of current line of text only.

Example(s)

extract
extract until "ADD"
extract thru 26
extract append all

Description

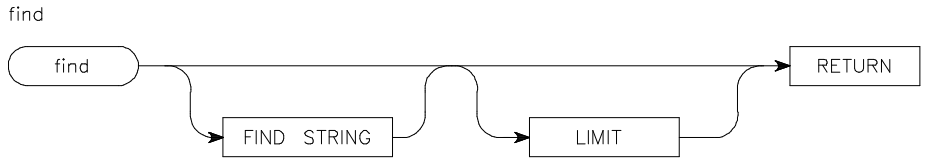
The temporary storage buffer for **extract** is shared with **copy**, and the buffer contents are not protected unless you use the **append** command. Otherwise, if the LIMIT is not found and **append** is not used, text which was previously in the buffer is **LOST!** The **retrieve** command (see that syntax heading) gives access to the current contents of the buffer.

The range for **extract** includes the current text line as one boundary and the value specified by LIMIT as the other boundary.

Once text has been extracted, the line following the last line extracted becomes the new current line of text.

find

Syntax



Function

Using the **find** command causes a pattern-matching search of the current text file for an occurrence of **< STRING>** within boundaries defined by LIMIT and by the range of columns specified by the **range** command. To be found, the first character of the string must occur within the specified range. For more information, see Description under this same syntax heading.

Default Value

find < STRING> : Defaults to string from last previous find or **replace** command, or to the null string (always matches) if no previous one is specified. **find < STRING>** LIMIT: LIMIT defaults to include first line after current line through the end of the file.

Example(s)

find
find "a*b" **thru** 35
find "he?lo" **thru end**

Description

When a defined string is found, the STATUS line displays "Found at column: " and shows the column number where the first character of the string occurs. In addition, the text line indicator appears on the line where the match was found, and in the **REVISE*** mode the edit cursor moves to the first character in the located string. In the **INSERT*** mode, a NEW line is inserted immediately after the line on which the string was found; the edit cursor is at start of that line.

After the last string within LIMIT is found, another **find** execution from that point will result in a STATUS line error message that the string has not been found. Also, find **< STRING>** locates only the first occurrence of

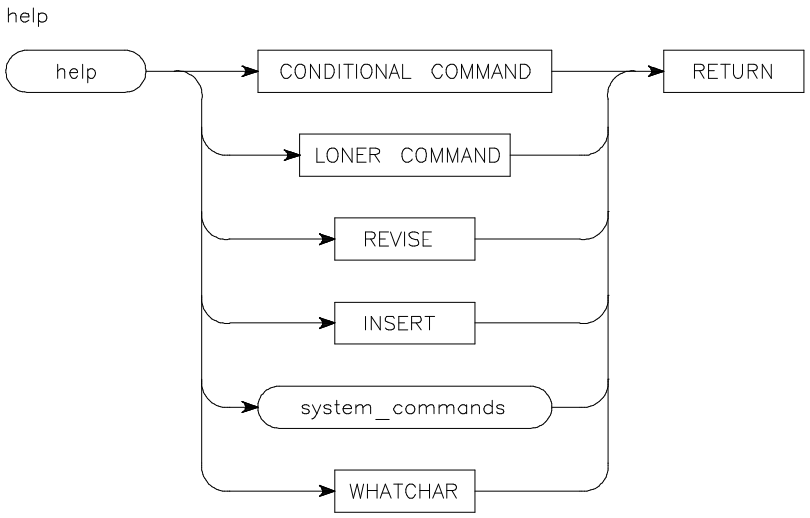
< STRING> in each line included in the LIMIT specification. (The **range** command narrows the **< STRING>** search to specific columns.)

If LIMIT is after the current text line, the **find** search begins on the line immediately following the current text line. If LIMIT is before the current text line, the **find** search begins with the line immediately preceding the current text line and proceeds backwards through the file to LIMIT (such as: **thru start**).

Since the **< STRING>** default is to the last executed **find** or **replace** command, you can make repeated searches with fewer keystrokes (by pressing Return from the command line; or by pressing **find** followed by Return from either **INSERT*** or **REVISE*** mode).

help

Syntax



Function

The **help** command provides you with on-line help information for the Softkey Driven Editor commands, variables, and modes of operation. The information appears on-screen and also may be listed to a printer (using the **list** command). For more information, see Description under this same syntax heading.

Default Value

none

Example(s)

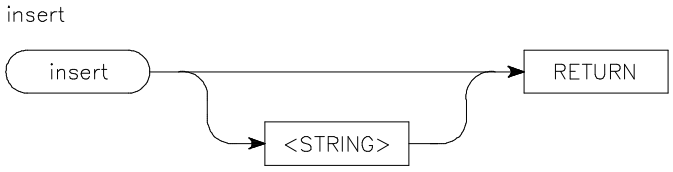
help delete
help WHATCHAR

Description

The **help** command is available by pressing either the **help** softkey, typing the word "help" on the command line, or typing "?". When either action is taken, topics on which help is available appear on softkeys (four levels, accessed in turn with **--ETC--**). When one of the softkeys is pressed, the topic will appear on the command line, and when Return is pressed, the information appears on-screen. The information is displayed by using the shell command "more(1)". The various options to this command apply, although normally you will press only the Space bar to continue paging through the information. Once the information has been displayed, the screen will be redrawn and the edit session will continue.

insert

Syntax



Function

The **insert** editor command allows you to create a new line in a text file from the command line, without actually entering either the **REVISE*** or **INSERT*** editor mode. This editor command **insert** is useful for inserting text, even blank lines, in the form of a **< STRING>** . (The editor mode **INSERT*** is a different softkey, which is used for either inserting successive lines in a file, or as the text entry mode when creating new files; it is described in the next syntax heading and in chapter 4.) For more information, see Description under this same syntax heading.

Default Value

insert < STRING> : If no string is defined, defaults to insert a blank line (same as insert).

Example(s)

insert
insert "this is a new line"

Description

A new line added by the **insert** command is placed immediately after the current text line (shown by line indicator >). An inserted string will be placed starting in column 1, regardless of whether or not **autotab** is on.

A blank line is inserted by default if no string is defined. Strings up to 240 characters are allowed (anything beyond 240 is truncated). However, to maintain compatibility with the HP 64000 LDS editor, you should limit string length to 232 characters or less (see appendix C for a description of differences between the Softkey Driven Editor and the HP 64000 LDS editor).

(MODE) INSERT*

Syntax Example(s) INSERT*

Function

The **INSERT*** editor mode is used to enter new lines of text into the current edit file, whether into an existing file or one being created. It is one of three Editor modes (the other two are Command and **REVISE***), and it is entered only by pressing the softkey labeled **INSERT**, located at the left on the first level of softkeys. (The editor command **insert** is a different softkey, which is used to create new lines, including strings, from the Command mode; **insert** is described in the immediately preceding syntax heading.) For more information, see Description under this same syntax heading.

Default Value none

Syntax Example(s)

INSERT (press softkey to enter mode)
INSERT* (press softkey to exit mode)

Description

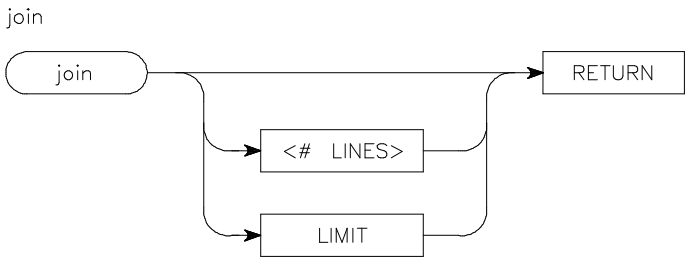
In the **INSERT*** mode, entire lines of text are added. Each new line is prefixed with NEW rather than a line number. The current text line is indicated by the symbol "> " and that line can be moved up and down in the file until either Return is pressed or the **INSERT*** mode is exited. Each time a new line is created, by pressing Return, another NEW line appears below it. When **INSERT*** is exited, if the current new line is empty, that line is removed rather than being inserted into the text file.

You can exit the **INSERT*** mode either by pressing the **REVISE** softkey to enter the **REVISE*** mode, or by pressing the **INSERT*** softkey which will take you to the Command mode on the command line.

When you are in the **INSERT*** mode, you can press any command softkey and move temporarily to the Command mode. When you then execute a command, you are placed back in the **INSERT*** mode on a new empty line. However, if you cannot execute the command or if you change your mind, it will be necessary to clear the command line before you can re-enter the **INSERT*** mode. The command line is cleared either by using the Backspace key or the "kill" character from your operating system stty setting. Once the command line has been cleared, you can re-enter the **INSERT*** mode by pressing Return or the **INSERT** key.

join

Syntax



Function

The **join** command takes a range of lines and joins them into a single line. The range is defined by using the current line as one of the boundaries and the line specified by the LIMIT or **< # LINES>** as the other boundary. For more information, see Description under this same syntax heading.

Default Value

join < LIMIT> : If no **< LIMIT>** is defined, defaults to join current line with next line.

Example(s)

join
join 2
join thru end
join thru -1

Description

When you specify a LIMIT, it is possible to join a line, or lines, either preceding or following the current line. For example, "**join thru** -1" would join the immediately preceding line to the current line of text. The **< # LINES>** option allows a number to be entered which specifies the number of lines following the current line to be joined to it. If the number of lines specified does not exist (before the end of the file), then an error is reported.

Lines are joined from the lower-numbered line (that is, uppermost on the display) to successive higher-numbered lines contained in the boundaries. The lines are joined by finding the last nonblank

character on the first line to be joined and the first nonblank character on the second line to be joined, and then inserting a single space between them as a delimiter. If a line to be joined is blank, no spaces are inserted for it, nor is a delimiter-space inserted.

Once the specified lines are joined, the resulting line appears at the lower line position, with the same line number as before the **join** command executed. All other lines in the join range are deleted.

The **join** fails if the result is a line of greater than 240 characters, or if the range specified is not found.

< LINE+ ->

Syntax Example(s)

REVISE* < **LINE**+ ->
INSERT* < **LINE**+ ->

Function

The < **LINE**+ -> prompting command softkey allows you to move immediately to either an existing numbered line or to a new current text line which is a relative number (that is, + or -) of lines away. An absolute, unsigned number may be used to select a specific numbered line in the file, while a relative, signed number may be used to specify a line above or below the current line position. For more information, see Description under this same syntax heading.

Default Value

none

Example(s)

6
-7
+ 53
0
9999

Description

If you are in either the **INSERT*** or **REVISE*** mode and press < **LINE**+ -> , you are temporarily placed in the command mode. From there you can enter either an absolute or relative number to select a desired new current line. From the **INSERT*** mode, if you press < **LINE**+ -> and move to the command line, enter 0 and press Return, a NEW line is inserted as your new current line just after the START line. On the other hand, from the **REVISE*** mode, if you press < **LINE**+ -> and move to the command line, then enter 0 and press Return, the first line of text (regardless of its number) becomes your current line.

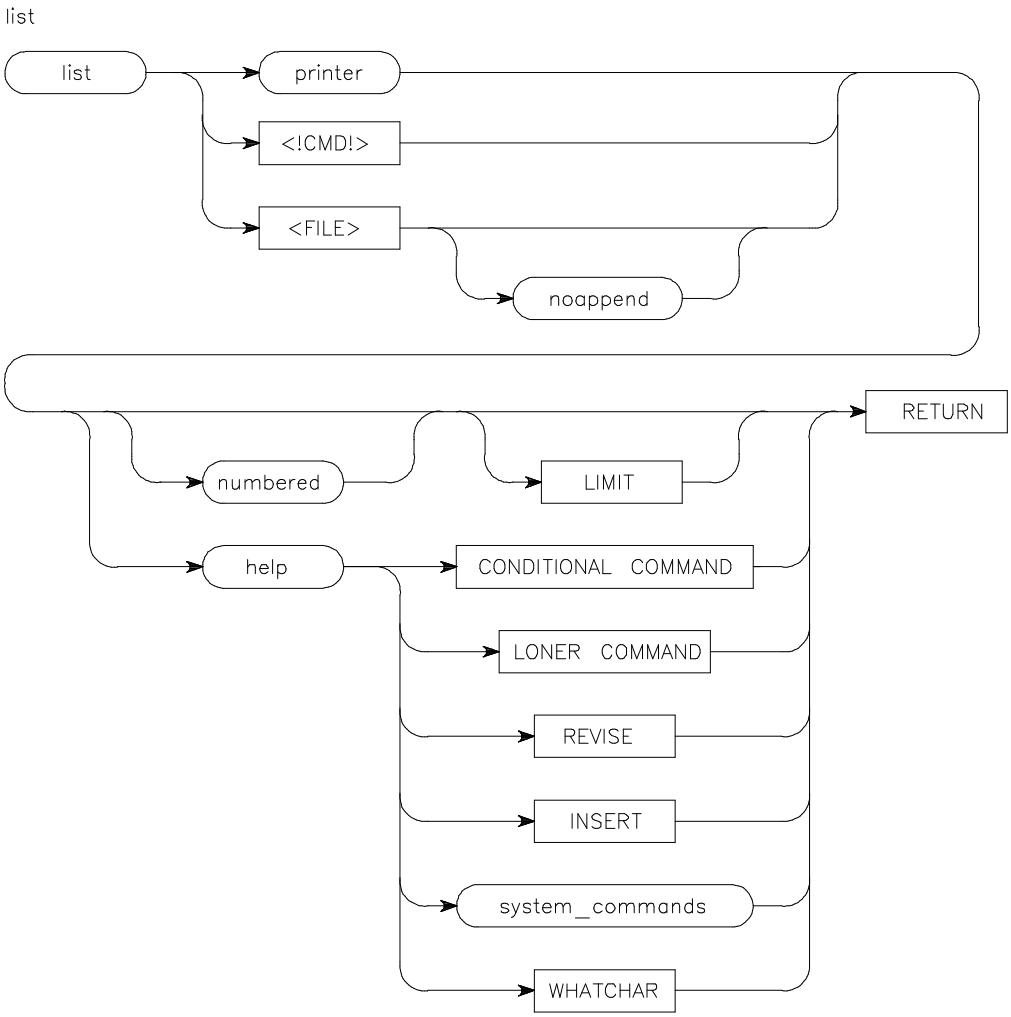
If you are in the Command mode, enter 0 and press Return, the START line will be indicated by > (cursor remains on command line). At this point, the START line is accessible only for positioning, and it cannot be modified.

When you use **< LINE+ ->** to specify a relative line outside the file boundaries (that is, a line position which would be greater than the number of lines preceding or following the current line), then either the first (if -) or the last (if +) line of the file becomes the current line; a message tells you the number of lines moved.

If an absolute (unsigned) number greater than the current line number is specified and that line does not exist, the next higher-numbered line from the sought line is first found, and the line before that one (whether numbered or NEW) becomes the new current text line. Similarly, if an absolute number lower than the current line number is specified and that line does not exist, the next lower-numbered line from the sought line is first found, and the line after that one (whether numbered or NEW) becomes the new current text line. If the sought line is below or above the file boundary, the START line or the last line, respectively, will be used as the new current line.

list

Syntax


list
list
printer
<!CMD!>
<FILE>
noappend
RETURN
numbered
LIMIT
help
CONDITIONAL COMMAND
LONER COMMAND
REVISE
INSERT
system_commands
WHATCHAR

Function

The **list** command allows you to output file text or **help** command text to a file, a printer, or to a shell command, using either numbered or unnumbered format. It includes options which let you specify whether or not the text being listed will be added to or will overwrite any text in the file being listed to (this is different from the HP 64000 LDS editor; refer to appendix C). Another option lets you define the LIMIT range over which the **list** command is to operate. For more information, see Description under this same heading.

Default Value

LIMIT defaults to current line of text.

Example(s)

list LSTFILE
list printer numbered all
list MAXINT **thru** 47
list printer help delete

Description

Doing a **list** to a file will normally cause the text to be appended to any existing text. If you wish for the entire file to be overwritten (deleted), you can specify the **noappend** option. Also, if the user has a $HOME/.recover directory, a copy of an existing file will be placed there before the listing occurs.

Since the default is to list unnumbered text, the listed text normally would have no line number information. However, if you select the **numbered** format, the output will be listed exactly as it currently exists in the editor. The **numbered** format would mean that new lines will be labeled with NEW rather than being assigned a line number.

If **help** text is listed, it will be exactly as it appears on-screen for the **help** command. Note that there is no tabconversion for a **list** command, so the output will contain only spaces unless the current edit file already has unexpanded tab characters. Also, "help" text may contain tab characters since the help information is simply a text file that exists on the disk.

list operates over a range which includes the current line of text as one boundary, and the value specified in LIMIT as the other boundary. (Refer to the LIMIT variable syntax.)

The **list printer** option depends on the $PRINTER shell variable which is initialized before the Softkey Driven Editor is invoked. This shell variable contains a shell command which can be executed to invoke the printer on your operating system. This shell command must accept standard input in order for

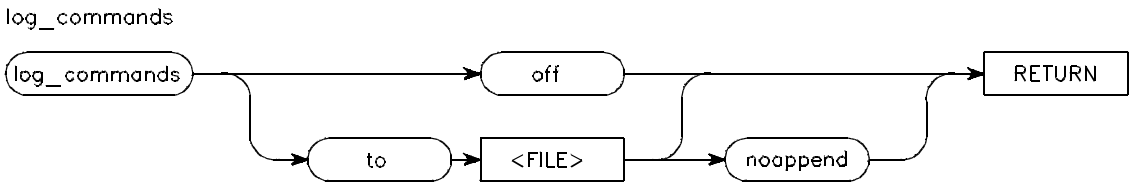
a listing to be generated. An example of setting this shell variable in your ".profile" is:

```
PRINTER = lp
export PRINTER
```

This shell variable indicates that the shell command "**lp**" is to be invoked whenever **list printer** command is executed. It is important that the shell variable PRINTER be exported. More information about this and about setting your ".profile" is located in chapter 2.

If **list** is used with an incorrect command or file which causes the screen to become filled with "garbage" characters, a CTRL l (lower-case L) will refresh the display from current memory.

Syntax



Function

log, standing for log commands, is used to control whether or not a command built and invoked from the Softkey Driven Editor is to be listed to a log file for later use. For more information, see Description under this same syntax heading.

Default Value

Defaults to append (added to) contents of designated log file.

Example(s)

log to afile **noappend**
log off

Description

Uses for log (commands) include both building command files and keeping track of commands invoked during an edit session. All commands executed from the command line will be placed in the log file. Text entered or modified in the **INSERT*** or **REVISE*** modes will NOT be logged; the use of keys, such as the cursor keys and delete line, also will NOT be logged.

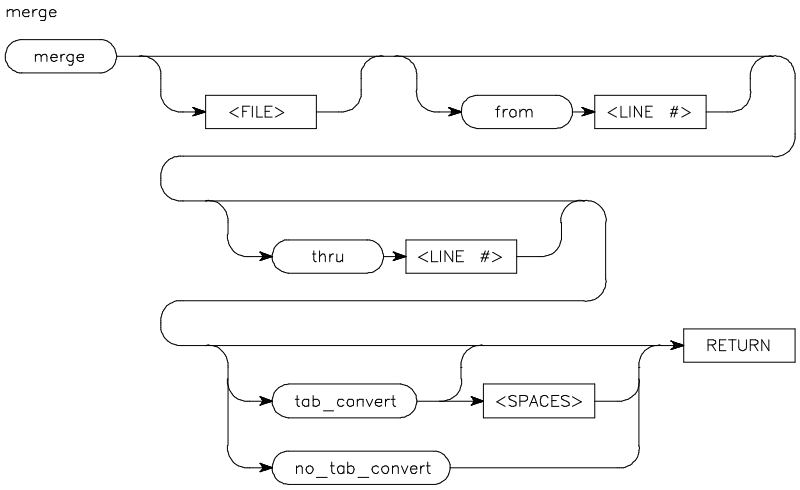
Also, remember that commands will normally be added (appended) to the designated log file. If you wish for newer commands being logged to overwrite those in the log file, you must choose the **noappend** option.

Logging of commands, as with executing command files, continues until the command either is turned off or until the last development environment feature is terminated.

The letter "L" will appear on the enunciator line while commands are being logged to a file.

merge

Syntax



Function

The **merge** command is used to combine an entire file or portions of a file into your current edit file, with options to select the range (LIMIT) over which the command is to operate and to choose whether or not tabconversion is to occur. For more information, see Description under this same syntax heading.

Default Value

merge < FILE> : Defaults to either the current file being edited or the last merge file.
merge from < LINE # > : Defaults to first line of **< FILE>** .
merge thru < LINE # > : Defaults to last line of **< FILE>** .
merge tab_conv < SPACES> : Defaults to tabstops every 8 spaces if no entry is made.

Example(s)

merge afile
merge bfile **from** 5 **thru** 37
merge /tmp/cfile **thru** 10
merge dfile **tab_conv** 4
merge efile **no_tab_c**

Description

Text merged from a file is added following the current text line, and the last line added from a file becomes the new current line.

If the line number specified in merge **from < LINE # >** is greater than the number of lines in the file, the merge is not done.

However, if the line number for **thru < LINE # >** is greater than the total number of lines in the file, the merge is done through the end of the file, providing the **from < LINE # >** used is valid (that is, less than or equal to the total number of lines in the file, as described above).

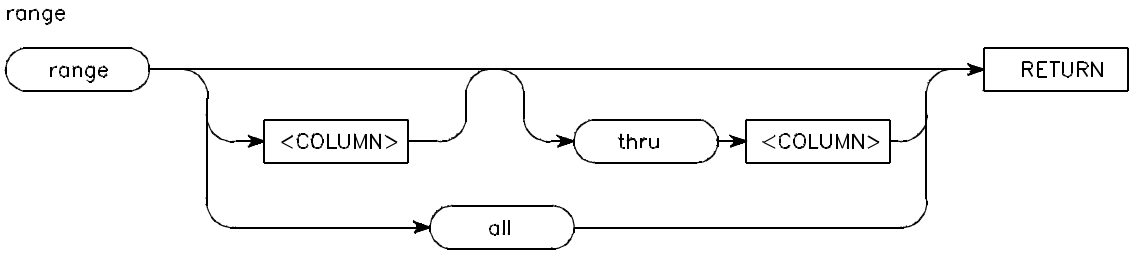
Tab characters from the text being merged are handled according to the options selected for the **merge** command. The **tab_conv** option should be selected if tab characters are used to represent movement of the cursor to the next tabstop. In this case, you can select the number of **< SPACES>** between tabstops or you can use the default action which places fixed tabstops every 8 spaces (that is, at 1, 9, 17, etc.). With **tab_conv**, when a line of text being merged is read into the editor, the tab characters in it will be converted into the number of spaces needed to move to but not including the next tabstop. A special case occurs if 1 is chosen for **< SPACES>** , as that will cause tab characters to be replaced with a single space.

If the text being merged contains tab characters that are meaningful as control characters, then the **no_tab_c** (no_tab_convert) option will prevent conversion of tab characters into spaces.

Note that the tabconvert options for the text being merged do not affect how tab characters in the current edit session will be stored; the tabconversion for saving depends entirely upon that specific **save** or **end** command.

range

Syntax



Function

The **range** command defines the width in columns of text to which all **find** and **replace** commands are restricted. (As noted under the syntax description for POINT variable, the setting for **range** command does not affect the **< STRING>** searches.) The value entered for the **thru < COLUMN>** option must be greater than or equal to the value for the **range < COLUMN>** option.

Default Value

range < COLUMN> : Defaults to 1 (one) if no entry, or if number is not within 1 to 240.
range thru < COLUMN> : Defaults to 240 if number is not within 1 through 240, or if not specified, defaults to same number as range **< COLUMN>** .

Example(s)

range
range 5 **thru** 15
range thru 200
range 16

Description

As indicated in the Default Value above, if values entered for either **< COLUMN>** are not within 1 through 240, a default will be set depending upon which column number is in error. If the first **< COLUMN>** entry is in error, it is set to 1. If the second **< COLUMN>** entry is in error, it is set to 240.

If the **range** command is executed without any options, then the **RANGE** mode is entered to allow visual setting of the range. In this mode, the current range of columns will be shown on-screen as a line of R's, that is, RRRRRRR, etc. The cursor appears in column 1, regardless of the autotab setting. The line of R's can now be moved anywhere within the file, much like in the **INSERT*** mode, and it also may be modified. Pressing either Return or the **RANGE*** softkey causes the new line settings to take effect.

When you leave the **range** mode with a blank line, then the range defaults to the maximum of 1 through 240, If the range line is not blank, the new range is set using the leftmost and rightmost nonblank characters.

renumber

Syntax

Function

The **renumber** command will renumber the text lines in a current edit file after lines have been added or deleted. Lines will be renumbered consecutively starting with 1 from the first line through the last line of text. You may wish to renumber text lines any time after you have made extensive changes by adding NEW lines or deleting text.

Default Value

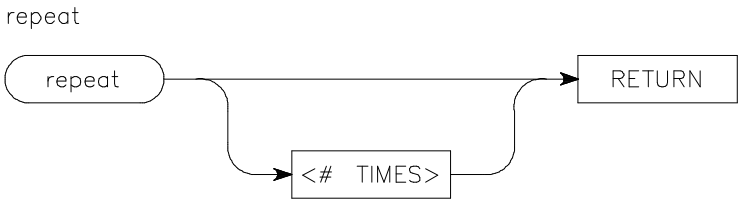
none

Example(s)

renumber

repeat

Syntax



Function

The **repeat** command will duplicate the current line of text a specified number of times and insert the duplicated line(s) immediately after the current line of text. The last inserted line becomes the new current line. Any nonzero value may be used for the number of times the line is to be duplicated. If the current line is the START line when **repeat** is invoked, then the first text line is the one which is repeated.

Default Value

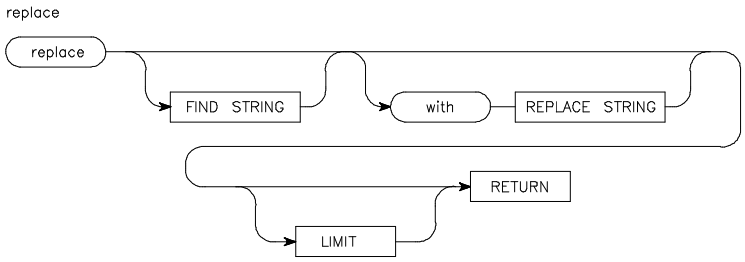
repeat < # TIMES> : Defaults to 1, that is, repeats once.

Example(s)

repeat
repeat 2

replace

Syntax



Function

The **replace** command allows searching for the first string which, if found within the range defined by LIMIT, will be replaced by the second string. It only has to find the beginning character of the first (matching) string within the range limit. It is not possible to replace a null string, so the matching string cannot be empty. For more information, see Description under this same syntax heading.

Default Value

replace < STRING> : Defaults to string used in previous replace or find command.
replace < STRING> with < STRING> : **with < STRING>** defaults to **< STRING>** from previous **replace** command, or to null string if no previous **replace** command used.
replace LIMIT : Defaults to current line if no LIMIT defined.

Example(s)

replace "hello" **with** "HELLO" **all**
replace "m?c*n" **with** "M?cn" **thru end**
replace "5 + 3 *" **with** "(5+ 3) * 7"

Description

The first string, as **replace < STRING1>** , is a matching string which is searched for with the range and LIMIT boundaries. This search is identical to the **find < STRING>** command, except that it will match multiple occurrences on the same line. The first character in this string must be found within the specified range for the string to be recognized. If this matching string

< STRING1> is found, it is deleted from the line and the **with < STRING2>** is put in its place. The search for the next possible match on that line begins with the character following the last character in the text matched by the search string.

Note that the default for **< STRING1>** in the replace command may come from the last executed **find** or **replace** command, and the default for **< STRING2>** may come from the previous **replace** command. This defaulting action allows you to alternate **find** and **replace** commands without having to define **< STRING1>** and **< STRING2>** each time.

The **replace** command makes special use of "anycharacter", represented by "?", and of "anystring", represented by "*" in the **< STRING1>** pattern-matcher and the **< STRING2>** replacement string. (Also refer to the syntax for **< STRING>** variable.) Use of anycharacter and anystring in the **replace** command is explained in the following paragraphs.

USING anycharacter: The "?" is used in **< STRING2>** to insert text matched by "?" in **< STRING1>** . If no "?" was used in **< STRING1>** , then a space is inserted at every occurrence of a "?" in **< STRING2>** (the "?" is removed in all cases). However, if "?" is used in **< STRING1>** (the pattern-matcher), then the "?" in **< STRING2>** matches the character of the "?" in **< STRING1>** on a one-on-one basis. If there are more "?" in **< STRING2>** than in **< STRING1>** , they wrap around to begin matching at the start of **< STRING1>** again.

USING anystring: The "*" is used in **< STRING2>** to insert text matched by "*" in **< STRING1>** . If no "*" was used in **< STRING1>** , then no text is inserted into **< STRING2>** , the replacement string (the "*" is removed in all cases). However, if "*" is used in **< STRING1>** , then the "*" in **< STRING2>** matches the "*" text in **< STRING1>** on a one-to-one basis. If there are more "*" in **< STRING2>** than in **< STRING1>** , they wrap around to begin matching at the start of **< STRING1>** again.

It is not valid to have a pattern-matcher **< STRING1>** which is either empty or which consists entirely of anystrings, as in **replace** "**" **with** "hello". However, this is valid for the replacement string **< STRING2>** , as in **replace** "a*b*d" **with** "**".

Here are some examples for use of "?" and "*", using "ABCDEF" as the text. Range is restricted to column 1 for the first example, and to columns 1 through 3 for the other examples.

replace "B*D" **with** "DDD"
RESULT: ABCDEF, because, due to the range, the search string "B*D" is not found.

replace "B" **with** "?"
RESULT: "A CDEF", because the "?" default, which is a blank, was substituted for B.

replace "C" **with** "*"
RESULT: "ABDEF", because the "*" default null string was substituted for C.

replace "$?_1?_2?_3$" **with** "$?_1?_2?_3?_1$"
where: $?_1$ = "A"; $?_2$ = "B"; and $?_3$ = "C".
RESULT: "ABCADEF", because only one match occurs due to the range restriction. The "extra" ? in string 2 wraps around and begins matching at the start of string 1 again.

replace "*C" **with** "**C" where: * = "AB"
RESULT: "ABABCDEF", because the additional "*" match in string 2 wraps around and reuses the "AB" from string 1.

replace "*D$?_1?_2$" **with** D$?_1?_2$*
where: * = "ABC"; $?_1$ = "E"; and $?_2$ = "F".
RESULT: "DEFABC", because of the placement of "?" and "*" characters.

A backslash (\ ) can be used as an escape which will allow finding occurrences of * , ? , or $ in a file. For example, if the exact string sought is "a*b", a match would occur for any string consisting of the letter "a", followed by any number of other characters, and ending with the letter "b". However, if the string "a*b" is used, the match will be exactly "a*b". If the string to be matched includes a \ , it must be escaped, as in "\\". If any other character preceded by a \ , for example, "\a", is to be matched, a second backslash must be added, making "\\a". Use of the backslash to escape characters for pattern-matching is the same for the first string used with the **replace** command, covered in the command syntax section.

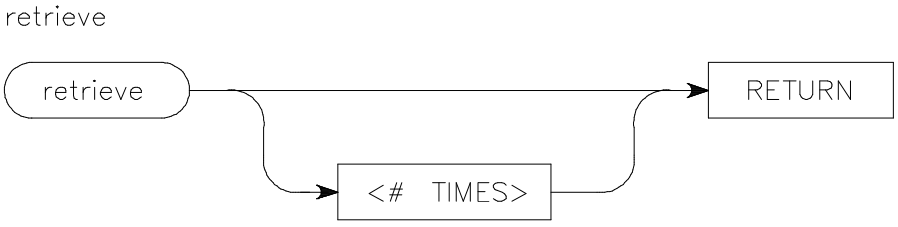
Using an octal number in a string is a special case which requires the backslash as an escape. The octal string can be used in either the matching or the replacement string, represented by "\nnn", where the first digit can be 0 through 3, the second and third digits from 0 through 7. All three digits must be used for recognition of the string. An example in a replace command to replace ASCII character DEL (octal 177) with ESC (octal 033) would be:

```
replace "\177" with "\033"
```

You can enter control characters in octal equivalent on the command line for the **find** or **replace** strings.

retrieve

Syntax



Function

The **retrieve** command provides access to the contents of the temporary storage buffer used with the **copy** and **extract** commands by placing a copy, or copies, of that storage buffer text into the current edit file, immediately following the current line of text. The last line of text retrieved from the temporary storage buffer becomes the new current line. You specify the number of copies of text with the **< # TIMES>** option, which accepts any positive integer. If no entry is made for **< # TIMES>** , the text is copied only one time.

Default Value

retrieve **< # TIMES>** : If no number entered, defaults to 1 (once).

Example(s)

retrieve
retrieve 2

REVISE* (MODE)

Syntax Example(s) **REVISE***

Function

The **REVISE*** editor mode is used to change existing lines of text. It is one of the three editor modes (the other two are Command and **INSERT***), and it is entered by pressing the softkey labeled **REVISE**, located second from the left on the first level of softkeys. For more information, see Description under this same syntax heading.

Default Value

none

Syntax Example(s)

REVISE (press softkey to enter mode)
REVISE* (press softkey to exit mode)

Description

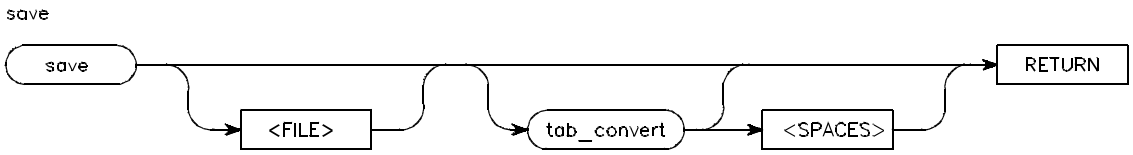
In the **REVISE*** mode, you can move about the file and modify text by writing over it or inserting text at any desired line or column in the file. The current text line is indicated by the symbol > in the left-hand column and the cursor is positioned at the current column.

You can exit the **REVISE*** mode either by pressing the **INSERT** softkey to enter the **INSERT*** mode, or by pressing the **REVISE*** softkey which will take you to the Command mode on the command line.

When you are in the **REVISE*** mode, you can press any command softkey and move temporarily to the Command mode. When you then execute a command, you are placed back in the **REVISE*** mode at the previous location in the text file. However, if you cannot execute the command or if you change your mind, it will be necessary to clear the command line before you can re-enter the **REVISE*** mode. The command line is cleared either by using the Back Space key or the "kill" character from your operating system stty setting. Once the command line has been cleared, you can re-enter the **REVISE*** mode by pressing Return or the **REVISE** key.

save

Syntax



Function

The **save** command is used to place a copy of the current edit file onto the system disk, and is similar to the **end** command except that with **save**, the edit session continues rather than returning to the shell. For more information, see Description under this same syntax heading.

Default Value

save : Defaults to current destination file name.
save tab_conv < SPACES> : If not specified, defaults to 8 spaces per tabstop.

Example(s)

save
save afile
save /tmp/bfile
save bfile **tab_conv** 4

Description

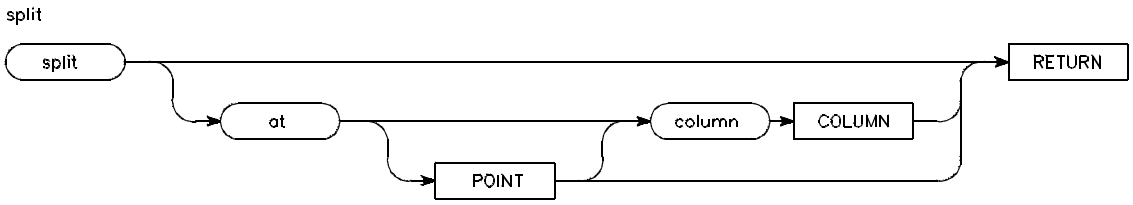
When you enter a **save** command, a check is made to see if the destination file already exists. If a file with that name already exists, and it is not the same as the current edit file name, you will be asked whether or not the file still should be saved. If your answer is "yes", and if you have created a $HOME/.recover directory, the old file copy is moved to the recovery directory before the file being saved is written to the disk.

A **save** will be done with no tabconversion unless you specify **tab_conv**. With no tabconversion, tab characters will not be used to replace spaces even though that might save disk space. With no tabconversion, the file can be printed or edited and it will be exactly as it appears in the editor.

For "make" files, use tabconversion so the file is acceptable to the "make" command.

split

Syntax



Function

The **split** command allows you to split a line of text into two lines at a specific POINT, including specification of a line number, a column number, and even a < **STRING**> . This results in two lines, with the second one being a NEW line containing the text following the POINT at which the split occurred, which now starts in column 1. For more information, see Description under this same syntax heading.

Default Value

split at POINT : Defaults to current edit line and column position.

Example(s)

split
split at column 30
split at -5
split at 5 **column** 5
split at "h*o"

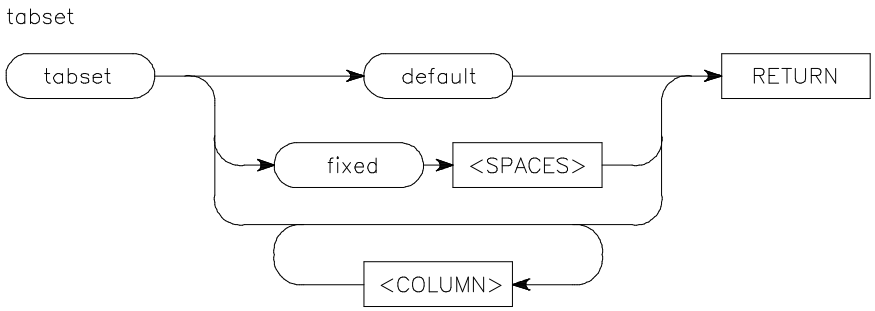
Description

If no option is specified for **split**, the current cursor position is used in **INSERT*** or **REVISE*** mode, or column 1 of the current line if **split** is invoked from Command mode (would insert blank line preceding the line used for the split).

A special case occurs when you enter **split at < STRING>** . If no column is indicated, the line split occurs at the column where the start of the string match occurs. For example, if the string were "hello", the split would be done on the line following the current line where "hello" first appears, and in the column where the "h" in "hello" occurs. If a column number is indicated, the split is done at that column, and not at the column where the string match was made. The range boundaries are not used to restrict possible matches for a **< STRING>** .

tabset

Syntax



Function

The **tabset** command is used to define the column positions the cursor moves to when Tab or Shift Tab (backtab) is pressed while you are in either the **REVISE*** or **INSERT*** mode. There is always a tabstop in column 1, regardless of other settings, and the initial (and default) setting will provide tabstops every 16 columns (that is, columns 1, 17, 33, etc.). For more information, see Description under this same syntax heading.

Default Value

tabset default : Places tabstops in every 16 columns starting with column 1.

Example(s)

tabset
tabset 5 10 15 20
tabset fixed 4
tabset default

Description

Note that **tabset** only affects how the Tab key works in **INSERT*** and **REVISE*** modes. It does not affect text or tab control characters in text.

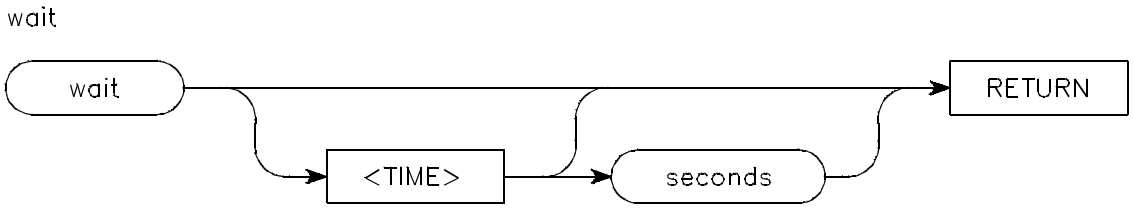
You can reset tabstops to standardize the appearance of text or to create tables. For **tabset < COLUMN>** , you can enter any number from 1 through 240 for any number of tabstops. The **tabset fixed < SPACES>** command is used to set tabstops at every **< SPACES>** columns, starting at column 1; the

valid values are from 1 through 239. The **tabset default** command resets to a default which is the same as a **tabset fixed** 16 command. Any values outside of the valid range are ignored.

When the **tabset** command has been executed and any new tabstops assigned, you are automatically placed in the **TABSET*** mode, and current tabstop positions are denoted by the letter "T" in each position. The **TABSET*** mode is similar to the **INSERT*** mode in that the line of T's can now be modified and it can be moved anywhere within the file. The final tab settings are determined by the nonblank characters on this line. Pressing either Return or the **TABSET*** softkey will remove the tabset display line and cause the tabset specification to take effect.

wait

Syntax



Function

The "wait" command is a hidden command, which only can be entered from the keyboard, and which will cause command files to be halted temporarily, either for a specified number of seconds or until CTRL c is pressed. After the wait period, the command file continues processing from that point. Note that CTRL c is only appropriate if the 'intr' stty option is set to that character. If a different character is set, then that control sequence should be used instead of CTRL c. The wait command will show the correct character to use.

Default Value

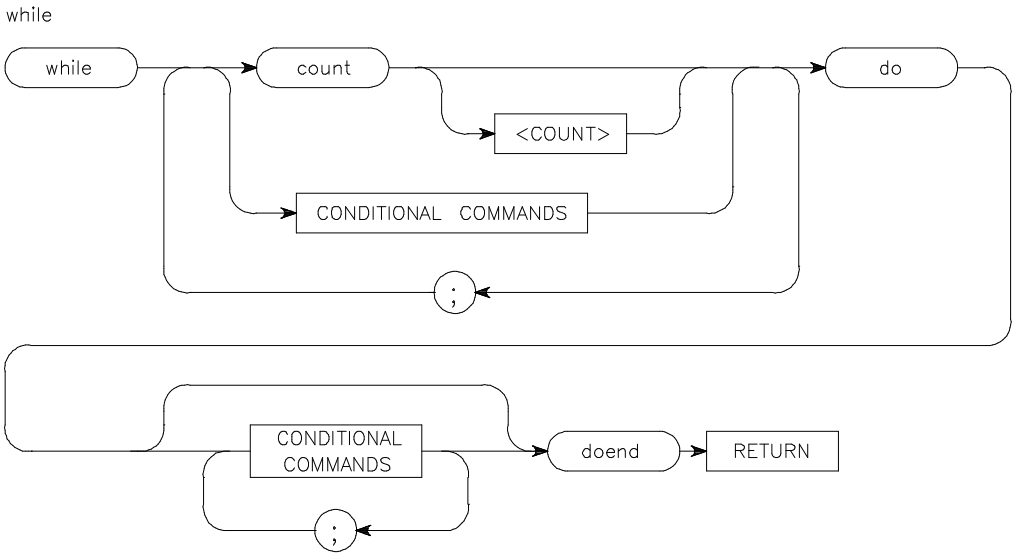
wait < **TIME**> : Press CTRL c to continue processing.

Example(s)

wait 10 seconds

while

Syntax



Function

The **while** command allows repeated execution of editor commands based on the result of a test clause. The **while** command is allowed with all editor commands EXCEPT the following (all of which could cause drastic results if allowed):

cd
CMDFILE
column_numbers
edit
end
help
log_commandsRsave
wait
!

For more information, see Description under this same syntax heading.

Default Value

< COUNT> : If no number entered, defaults to 1 (one).

Example(s)

while count 5 **do insert** "hello"; **delete thru** + 2 **doend**
while count 10; **find** "aline" **do delete doend**
while find "change" **do replace with** "newstuff" **doend**

Description

Execution of the **while** command involves two sets of commands called a "Test Clause" and a "Body Clause". The "Test Clause" consists of one or more editor commands which are all executed at the beginning of each iteration of the **while** command. The allowable commands, exceptions noted above in Function, are assigned truth values for this use. For example, **autotab**, **insert**, **range**, and **tabset** commands are always true. Truth values for the other commands are assigned based on the results when they are executed. Appendix B contains a summary of the truth values.

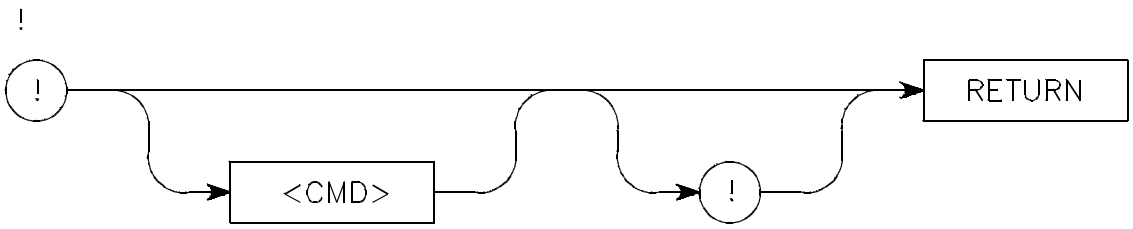
If the "Test Clause" returns a true value, meaning that execution of all commands in it returned a true value, then the "Body Clause" is executed. However, if the "Test Clause" result is false, meaning that one or more of the editor commands in it returned a false value, then the **while** command is terminated. Note that all commands in the "Test Clause" will be executed to determine its truth value.

The truth value for the **while** statement itself depends upon the truth value of the "Body Clause". Before the **while** statement begins, the "Body Clause" is assumed to have a true value. After each iteration when all the commands in the "Body Clause" are executed, the value is set depending on the result returned. If the "Body Clause" is empty or if all the statements returned a true value, the "Body Clause" value is set to true. However, if one or more of the statements in the "Body Clause" was false, the value is set to false. After further iterations, when the "Test Clause" finally fails, the value of its last "Body Clause" execution is returned.

The **count** command is used to control the number of iterations for the **while** command. In the "Test Clause", the count is evaluated as true as long as the number of iterations of the **while** command is less than or equal to the value of the **< COUNT>** variable. If no number is entered for **< COUNT>** , it defaults to 1. If **count < COUNT>** is used in any other location than mentioned here, it has no effect on the truth value.

(Shell Command) !

Syntax



Function

The "!" represents Shell Command, which is a hidden command, and it allows you to invoke shell commands from the Command mode without having to end the edit session. It does not allow you to move text from or to the editor with the "!" command. For more information, see Description under this same syntax heading.

Default Value

shell command : Will invoke a shell, using the SHELL variable, if set.

Example(s)

```
!ls /users
!sh
!sk anotherfile
!uucp anothersystem\!~ /afile ~ !
```

Description

Once the shell command has terminated, you must press a key to cause the display to be redrawn for the current edit state. Any shell valid command may be used by prefixing it with ! and following it with ! . If the "!" is a part of the shell command, it must be escaped with a backslash (\ ).

Here is a brief explanation for the examples shown above:

!ls /users: Lists directory for /users.

!sh: Moves to a new shell temporarily; exit that shell back to the editor with CTRL d.

!sk: Starts a different edit session.

!uucp anothersystem\!~ /afile ~ !: Gets a file from another computer system and moves it to the public directory. Note backslash (\) to escape ! inside command.

Installation Notice

Help and Problem-Solving

Chapter Overview

This chapter provides the following information to assist you in finding answers to both general and specific questions or problems while you are using the Softkey Driven Editor:

- What is the meaning of an error or status message displayed on my screen?
- Can I get answers by reading the operating system "man" (manual) pages and is there help built into the editor?
- What should I look at to get a better understanding about strings, especially as used in **find** and **replace** commands?
- Where do I look for information about recovering saved files, and especially what happens if my session "crashes"?
- Have some of my other questions about problems with file names, control characters, and strange on-screen displays been answered somewhere in this manual?

Error and Status Messages

While you are using the Softkey Driven Editor, messages will appear on the STATUS line and some messages or questions appear along the command line. Most of the messages and questions have sufficient information in them to let you proceed or correct a problem pointed out by the Editor.

Chapter 5, which covers editor command syntax, and chapter 3, which covers editor modes and operating system connections, will help your understanding of the terminology used in the messages and questions. Those chapters also will give you a better idea of why the message or question appears on-screen just as you have completed a particular action or keystroke.

Finally, appendix A contains a listing and explanations for all of the on-screen messages and questions which you may encounter in using the editor.

"man" Pages and Editor "help"

In addition to this printed manual, there are two other sources of information, much of which contains the same information as in this manual. You can access "man" pages from the operating system shell for the various Softkey Driven Editor shell commands such as **sk**, **purge**, **rcvr**, **dirrec**, **skpreserve**, and **skrecover**. This is usually done by entering the shell command: "man < subject> ". In addition, the editor has a **help** command which is described in chapter 5. It is accessible from a softkey in all editor modes, with the convenience of pressing any of the softkeys displayed for information about the subject represented by the label on the softkey. In other words, if you want to read an explanation on-screen about the **join** command, you would first press the **help** softkey (or just type "help", from the keyboard, on the command line), and then press the **join** softkey. Information begins appearing on-screen and you move to additional information (when the message at the bottom of the screen says "more") by pressing the Space bar.

Understanding Strings

Strings are a very powerful tool to use in checking and modifying large files. If you have questions in general about strings and their use with the editor, you can refer to the syntax for **< STRING>** as a variable in chapter 5. For further information about strings, the **find** and **replace** command syntax in chapter 5 should be helpful. In particular, the syntax for the **replace** command has some examples involving use of anycharacter ("?") and anystring ("*") to help clarify how those editor tools work.

Recovery of Files and Session "Crashes

There are two aspects of recovering files: 1.) moving files into and out of your $HOME/.recover directory; and 2.) recovering files which are being edited when some sort of "crash" occurs (such as a power failure or a phone/modem hang-up). Chapter 3 provides you with the information about using shell commands "**purge**", "**rcvr**", and "**dirrec**" to manipulate files into and out of your $HOME/.recover directory. If you should be unfortunate enough to have a "crash" occur during an edit session, you can find information in chapter 3 about what to do and what happens to allow you to recover most, if not all, of the files involved in the session. (This involves the "**skrecover**" command, run by you after "**skpreserve**" has been run as explained in chapter 2.)

Questions and Possible Answers

The following paragraphs contain some questions about subjects on which you may encounter a problem, along with some information which may help you find a solution or at least point the way to a possible solution.

Why Aren't My Control Characters Entered or Displayed?

Control characters are entered into the editor by a two-step method: You press **CTRL v**, followed by the control character. For example, to enter a **CTRL l** (lower-case L), you first type **CTRL v** and then **CTRL l** (pressing the control key and the letter at the same time). A control character is shown as a " . ". In the text area of the Softkey Driven Editor, you can use the **WHATCHAR** softkey with the cursor positioned over the "period", and find what the real character is, as well as its numerical value in decimal, hexadecimal, or octal. The subject of entering control characters is discussed in chapters 3 and 5.

Why Doesn't My File Print Okay Outside the Editor?

The problem probably is caused by conversion of tab characters. The editor normally does not have tab characters in it. When a file is sent to the printer using the **list** command, there are only spaces in the file and the printout should look exactly like the text inside the editor. The problem usually occurs when the file is saved to the disk from the editor. If the file was saved using the **tabconvert** option, then tab characters are used whenever space can be saved in the disk file. Then, when the file is sent to the printer, the tab characters can cause the output to look quite different. The solution is to either save the file without the **tabconvert** option (using the default) or to convert the tabs to spaces before sending the file to the printer using the "expand(1)" shell command. See the syntax in chapter 5 for more information.

Why Do I Get a Syntax Error When I Enter a File Name?

The likely problem has to do with the scanner of the Softkey Driven Editor. The standard file name format used by the editor is any name starting with a letter, followed by any number of letters and/or numbers. A possible problem is that the file name you entered is a "token" used by the editor, such as the

word "copy", which is also an editor command. Another possibility is that the file name has some strange characters, such as "-" or "@", or starts with a number. The solution to getting the file accepted is to escape the offending character with a backslash " \ ". For example, to edit the file named "copy", the command might look like this: "**edit** \copy". To edit a file where the name contains a "-", the command could be "**edit** the\-file". Another solution is to put the entire file name in quotes, as in "**edit** 'the-file'". However, if "$" is part of the file name, you must escape the "$" regardless of whether the file name is quoted or not. Otherwise, the "$" will be taken as part of a shell variable and will be replaced by any text that it matches.

Why Doesn't My "Make" File Work Any More?

One of the requirements for a "make" file to work is that tab characters must be used in front of all commands that are to be executed as part of the make(1) process. The problem can occur in using the editor because the default action for saving a file is to save with only spaces and no tab characters. In this way, the "make" file no longer will be acceptable to make(1). The solution is to re-edit the file using the editor, and to save the file using the **tabconvert** option, thus putting tab characters back into the file.

Why Do I Sometimes Get Strange Characters Displayed On-Screen?

This may occur when you are scrolling up and down in a file or when you make a large jump from one place in the file to another. It may be caused by the terminal being used. In some cases, the terminal is not able to keep up with the baud rate of transmission of the computer. When this happens, the data is scrambled on the display into what is commonly called "garbage". The best solution for this is to lower the baud rate for your terminal and the computer it is networked to. If this change in baud rate appears to solve the problem, you should inform your System Administrator so the baud rate is changed as far as the computer is concerned. This should involve changing the "getty" file in the "/etc/inittab" and "/etc/gettydefs" files to reflect the new baud rate. The change procedure is documented in the manual available to the System Administrator. The change made does not take effect until the computer is brought back into the multi-user state.

Installation Notice

Editor Status and Error Messages

Introduction

The Softkey Driven Editor displays error and status messages on the STATUS line and on the command line. The messages and questions relate to editor modes, to editor command syntax, and to various shell commands. The messages are summarized in this appendix, along with explanations of the meaning or purpose for the message or question.

COMMAND

MESSAGE

EXPLANATION

Invoking Editor status/error messages

display size is NUMBER lines by NUMBER columns.
It must be at least 24 by 80.

The Softkey Driven Editor will only work on displays that are at least 24 lines and 80 columns in size. The editor will stop if you attempt to run it in a window that is smaller than 24 lines by 80 columns.

terminal initialization failed

Problem with window system.

usage: sk [-c] [-n] [-v] [-f number] [-t number] [-i filename] [filename]

Bad option or filename while invoking editor from shell.

$HOME/.recover directory not accessible

Protection on $HOME/.recover directory has been set so user can't modify it, or the $HOME/.recover directory doesn't exist.

Maximum number of recoverable files available is NUMBER

MAXREC shell variable set outside of limits from 1 to 128.

syntax error

Command not formed correctly. If an 'R' appears in the right corner of the STATUS line, then the command is syntactically correct.

Keyword completion not possible

No valid keywords are possible given the current command line.

Possible tokens: COMMAND_LIST

A list of valid keywords given the current command line.

Response pipe create

An internal problem occurred while trying to create a pipe to the feeder.

Data pipe create

An internal problem occurred while trying to create a pipe to the feeder.

exec of feeder

An internal problem occurred while trying to exec the feeder.

Cannot status input

The feeder does not respond to the editor.

No homedown in shell escape

The terminal is not able to do a homedown.

yacc stack overflow

Command too complex or too large to be parsed.

yacc reduce stack overflow

Command too complex or too large to be parsed.

Disk full, line truncated

System disk is full, editor cannot write data.

Disk full, some data was lost

System disk is full, editor cannot write data.

Disk error, line truncated

Error occurred on system disk, editor cannot write data.

Number overflows a 32-bit integer

Entered number is too large.

Command status/error messages

autotab

Autotab is now off

Autotab mode is now off.

Autotab is now on

Autotab mode is now on.

Autotab in columnar mode, column: NUMBER

Autotab mode to a specific column is now on.

cd

Working directory is DIRECTORY

The change directory command worked.

< CMDFILE>

Can’t execute command file - No feeder

The command file cannot be executed since the file $HP64000/lib/feeder does not exist.

Can't execute commandfile - Feeder rev. NUMBER != Feature rev. NUMBER

The command file cannot be executed since the file $HP64000/lib/feeder revision number does not match the editor internal revision number.

Define parameter

If the user did not specify enough parameters, then the editor will prompt for them.

Parameter > 130 characters

The size of the parameter is greater than maximum of 130 characters.

Parameter name is invalid or longer than 30 chars

The parameter name given in the command file is too long or incorrect.

Parameter must be specified

A parameter must be entered.

Missing parameters in nested invocation

When invoking a command file from another command file, all parameters must be defined.

CMDFILE :not a text file

The command file to be executed is not a normal text file.

column_numbers

Column number updating on

Column number updating to the status line during the REVISE, INSERT, range, and tabset modes is turned on.

Column number updating off

Column number updating to the status line during the REVISE, INSERT, range, and tabset modes is turned off.

copy

String not matched; no lines copied

The match string used as part of the limit was not found.

Lines copied: NUMBER

The indicated number of lines have been copied to the internal buffer.

delete

String not matched; no lines deleted

The match string used as part of the limit was not found.

Lines deleted: NUMBER

The indicated number of lines have been deleted.

edit

Invalid tab conversion; using default

The tab conversion value entered was less than 1 or greater than 239, so default of 8 was used.

Do you want to lose changes?

Occurs if the current edit file has been modified. If the answer is anything but 'yes' or 'y', then the edit session continues without losing those changes. If the answer is 'yes' or 'y', then the modified current file is lost and the new file is edited.

Answer is too long

The answer to the question was too long.

No changes lost, edit resumed

The answer to the question was something other than 'yes' or 'y'.

file FILE is read protected

The file to be edited is read-protected, so it cannot be read.

edit (Cont'd)

file FILE is not ordinary

The file to be edited is not an ordinary text file, so it cannot be edited.

file FILE is not accessible

The file either does not exist or the path to that file is protected. As a result, the file cannot be edited.

Loading FILE

File is being loaded into editor.

Editing new file

An edit session is begun without any data or file name. The user is placed in the INSERT* mode.

Editing into FILE

An edit session is begun without any data, but is given a filename. The user is placed in the INSERT* mode.

Editing FILE

This file is been loaded into the editor.

Editing FILE, truncation

This file has been loaded, but some lines were truncated due to being over 240 characters long.

Interrupt, editing FILE

This file has been loaded, but loading of the file was interrupted, so some data may be missing.

Interrupt, editing FILE, truncation

This file is being edited, but some lines were truncated due to being over 240 characters long. Also, loading of the file was interrupted, so some data may be missing.

Editing FILE into FILE

The first file has been loaded into the editor and its current name is the second.

Editing FILE into FILE, truncation

The first file has been loaded into the editor and its current name is the second. Some lines were truncated due to being over 240 characters long.

Interrupt, editing FILE into FILE

The first file has been loaded into the editor and its current name is the second. Also, loading of the file was interrupted, so some data may be missing.

Interrupt, editing FILE into FILE, truncation

The first file has been loaded into the editor and its current name is the second. Some lines were truncated due to being over 240 characters long. Also, loading of the file was interrupted, so some data may be missing.

end

Invalid tab conversion; using default

The tab conversion value entered was less than 1 or greater than 239, so default of 8 was used.

Can't update recoverable files directory

Protection on $HOME/.recover/directory has been set so user can't modify it.

Can't set permissions on recoverable list

Protection on $HOME/.recover/directory has been set so user can't modify it.

No destination file

No file name was given to the current edit file nor was one used as part of the command.

file FILE already exists, delete old?

If writing to an existing file with a name other than the current edit file, then this question is asked. If the answer is 'yes', or 'y', then the existing file is either put in the recovery directory or removed. The file is then written to disk. If the answer is anything else, then no action occurs and the edit session continues.

Answer is too long

The answer to the question was too long.

File FILE not deleted, edit resumed

The answer to the query was something other than 'yes' or 'y'.

end (Cont'd)

Unable to purge FILE, no save made

The purging of the file to the recovery directory failed, so the current edit file could not be written. The recovery directory is possibly read- or write-protected.

File FILE is a directory or special file, edit resumed

The file to be saved to is an existing directory or special file, which is not allowed.

File FILE not accessible, edit resumed

Part of the path for the file is protected against writing.

writing FILE

The file is being written to disk.

error opening file FILE

Some internal error occurred while opening the file for writing.

Interrupt, wrote NUMBER lines

The writing of the file was interrupted, so it may not be completely written.

extract

String not matched; no lines extracted

The match string used as part of the limit was not found.

Lines extracted: NUMBER

The indicated number of lines have been extracted and placed in the internal buffer.

find

Can't have two '*'s next to each other; no find made

Illegal match string in find command.

String not matched; no find occurred

The match string used as part of the limit was not found.

Finding ^ MATCHSTRING^

Indicates that a find command is executing.

find (Cont'd)

Invalid limit

The limit specified for the find command was illegal or invalid.

Interrupt, ^ MATCHSTRING^ not found

The find command was interrupted before the pattern could be found.

Found at column: NUMBER

Indicates the column position of the found pattern.

^ MATCHSTRING^ not found

The string could not be found in the given limit and range margins.

NUMBER: Too many ? or * in matching string

More than 240 ? and/or * used in the pattern.

NUMBER: matching string overflow

The pattern was too large for internal storage.

help

Finding help text

The help file is being searched.

Help softkeys are not defined properly

The help file contains lines that are not in the proper format.

$HP64000 is the only environment variable allowed

The help file may only contain the $HP64000 shell variable as part of the file names.

(MALLOC) unable to allocate memory

No memory was available for the help command.

Help file did not close

The editor was not able to close the help file for some internal reason.

insert

Data lost due to truncation

The inserted line was greater than 240 characters, so some characters were lost.

INSERT

To leave, press INSERT key again

Indicates how to leave the INSERT* mode to return to the command mode.

Null character not allowed

User cannot enter a null character as text.

Text line number or + /- offset from current line

User temporarily leaves INSERT* mode. Enter a line number and press Return. The editor will move to the line and resume editing the file in the INSERT* mode.

join

String not matched; no strings joined

The match string used as part of the limit was not found.

Join: range error

Bad limit for join.

Interrupt, no join occurred

An interrupt occurred before the join could complete.

Join: line too long

The resulting line was longer than 240 characters.

Number of lines joined: NUMBER

Indicates the number of joined lines.

< LINE+ ->

Line not present

An absolute line value was given, so the editor uses the nearest existing line number for positioning.

< LINE+ -> (Cont'd)

Line not present, plus NUMBER

Given offset would go outside of the current file, so the editor uses the end of file.

Line not present, minus NUMBER

Given offset would go outside of the current file, so the editor uses the START line.

Jumping to NUMBER

Going to a specific line.

Jumping to line 0

Going to the START line.

Jumping to new line

Going to a line without a line number.

list

String not matched; no lines listed

The match string used as part of the limit was not found.

Listing ...

Listing is proceeding.

Invalid limit

A bad range was given for listing.

Tried to list to directory or special file

The file name given was a directory or special file.

Cannot access file

The path to the given file is write-protected.

Unable to purge FILE

The editor was unable to purge the existing file so it could be listed to.

Can't open FILE

The editor was not able to open the designated file for writing.

list (Cont'd)

Interrupt, listed NUMBER lines

An interrupt occurred during listing.

Listed NUMBER lines

Indicates the number of lines listed.

PRINTER environment variable not defined

In order to use the 'printer' option, the PRINTER shell variable must be defined and exported.

PRINTER environment variable must not be blanks

The PRINTER shell variable must be assigned a shell command.

Command must not be blanks

When using a shell command as a destination for the list command, it must not be left blank.

Hit return to continue

After using a shell command as a destination for listing, the user must press a key to cause the display to be redrawn in case some output was generated.

Listed help text

Help text was listed.

Help softkeys are not defined properly

The help file contains lines that are not in the proper format.

$HP64000 is the only environment variable allowed

The help file may only contain the $HP64000 shell variable as part of the file names.

(MALLOC) unable to allocate memory

No memory was available for the help command.

Help file did not close

The editor was not able to close the help file for some internal reason.

error is NUMBER

Some error occurred while listing.

list (Cont'd)

error in closing file; FILENAME

Some error occurred while closing the file that was listed to.

error in closing file; error is NUMBER

Some error occurred while closing the file that was listed to.

log_commands

Appending commands to LOGFILE until "log off"

Logging to a file, appending to its contents.

Logging commands to LOGFILE until "log off"

Logging to a file, overwriting the current file contents.

Appended NUMBER lines to LOGFILE

When logging is turned off, indicates the number of lines appended to the log file.

Logged NUMBER lines to LOGFILE

When logging is turned off, indicates the number of lines written to the log file.

No log file active

When using the 'log_commands off' command and no logging is occurring.

merge

Invalid Line specification

Bad limit given for merge command.

No merge file name given

No file name was specified nor does the current edit session have a file name.

file FILE not accessible

The file to be merged either does not exist or a part of its path is read-protected.

file FILE read protected

The file to be merged is read-protected.

merge (Cont'd)

file FILE not ordinary

The file to be merged is either a directory or a special file.

error opening file FILE

The editor was not able to open the file.

Merging ...

A merge is occurring.

Interrupt, merged NUMBER lines, truncation

The merge command was interrupted, having merged some lines, with some of those lines being over 240 characters in length and requiring truncation.

Merged NUMBER lines, truncation

The merge command merged some lines, with some of those lines being over 240 characters in length and requiring truncation.

Interrupt, merged NUMBER lines

The merge command was interrupted, having merged some number of lines.

Merged NUMBER lines

The merge command merged some number of lines.

Merged NUMBER lines, NUMBER nulls

Nulls are removed from merged text.

range

Range column NUMBER

The current range is for the specified column.

Range columns NUMBER thru NUMBER

The current range is from the first column to the second column.

Insert line key not implemented

The 'insert line' key is not defined for the range mode.

range (Cont'd)

Range is now columns NUMBER thru NUMBER

The resulting range after using the range mode is from the first NUMBER column to the second NUMBER column.

renumber

Renumbering

Renumbering is occurring.

Lines of text: NUMBER

The number of lines in the current file.

repeat

Repeat: range error

A bad limit for the repeat command.

No repeat made, repeat value 0

A value of 0 was used, so no repeat was made.

Repeating

Repeating is occurring.

Lines repeated: NUMBER

The number of lines actually repeated.

replace

Can't have two '*'s next to each other; no replacement made

Illegal match string in replace command.

String not matched; no strings replaced

The match string used as part of the limit was not found.

Illegal range - No replacements made

Invalid limit was given for replace command.

No replacement string available

No replacement string was given and no previous replacement string was available.

replace (Cont'd)

Null matching string - No replacements made

The first pattern-matching string cannot be the null string.

replacing ^ MATCHSTRING^ with ^ REPLACESTRING^

Replacement is occurring.

Interrupt, no replacements made

The replace command was interrupted before any replacements could be made.

Interrupt, replaced NUMBER occurrences, truncation occurred

The replace command was interrupted. The replace command had replaced some strings, the result of which was that some lines were truncated when their length exceeded 240 characters.

Replaced NUMBER occurrences, truncation occurred

The replace command had replaced some strings, the result of which was that some lines were truncated when their length exceeded 240 characters.

Interrupt, replaced NUMBER occurrences

The replace command was interrupted, having replaced some number of strings.

Replaced NUMBER occurrences

The replace command replaced all occurrences of the string in the given limit and range.

No replacements made

The replace command was not able to find any occurrences of the pattern in the given limit and range.

NUMBER: Too many ? or * in matching string

More than 240 ? and/or * used in the pattern.

NUMBER: matching string overflow

The pattern was too large for internal storage.

retrieve

NUMBER lines retrieved

Indicates the number of lines retrieved from the internal buffer.

REVISE

To leave, press REVISE key again

Indicates how to leave the REVISE* mode to return to the command mode.

Unable to enter REVISE mode since file is empty

If there are no lines in the current file, then the user may not enter the REVISE* mode.

Must exit REVISE mode since file is empty

Since there are no lines in the current file, the user must leave the REVISE* mode.

Null character not allowed

User cannot enter a null character as text.

Text line number or + /- offset from current line

User temporarily leaves REVISE* mode. Enter a line number and press Return. The editor will move to the line and resume editing the file in the REVISE* mode.

save

Invalid tab conversion; using default

The tab conversion value entered was less than 1 or greater than 239, so default of 8 was used.

Can’t update recoverable files directory

Protection on $HOME/.recover/directory has been set so user can’t modify it.

Can’t set permissions on recoverable list

Protection on $HOME/.recover/directory has been set so user can’t modify it.

No destination file

No file name was given to the current edit file nor was one used as part of the command.

save (Cont'd)

file FILE already exists, delete old?

If writing to an existing file with a name other than the current edit file, then this question is asked. If the answer is 'yes' or 'y', then the existing file is either put in the recovery directory or removed. The file is then written to disk. If the answer is anything else, then no action occurs and the edit session continues.

Answer is too long

The answer to the question was too long.

File FILE not deleted, edit resumed

The answer to the query was something other than 'yes' or 'y'.

Unable to purge FILE, no save made

The purging of the file to the recovery directory failed, so the current edit file could not be written. The recovery directory is possibly read- or write-protected.

File FILE is a directory or special file, edit resumed

The file to be saved to is an existing directory or special file, which is not allowed.

File FILE not accessible, edit resumed

Part of the path for the file is protected against writing.

writing FILE

The file is being written to disk.

Saved FILE

The file has been written to the disk.

error opening file FILE

Some internal error occurred while opening the file for writing.

Interrupt, wrote NUMBER lines

The writing of the file was interrupted, so it may not be completely written.

split

String not matched; no split occurred

The match string used as part of the limit was not found.

Split: line NUMBER not found

The split command was not able to find the line that was to be split (as in an empty file).

Split: illegal column NUMBER

A column value was given that was less than 1 or greater than 240.

tabset

Using default value

The tab value entered was less than 1 or greater than 239, so default of 16 was used.

Insert line key not implemented

The 'insert line' key is not defined for the tabset mode.

Too many variable tab positions

Too many tab positions were entered for the tabset command.

wait

Waiting for SIGINT (ctrl-c) or NUMBER seconds

To continue, either a CTRL c must be pressed or some number of seconds must pass.

Waiting for SIGINT (ctrl-c)

To continue, a CTRL c must be pressed.

Not able to get system time

The editor was not able to determine the system time used for waiting.

!

Executing shell command ...

The shell command is being executed.

Hit return to continue

After a shell command has completed, this message informs the user that a key must be pressed to continue. After the key is pressed, the display is redrawn and the edit session continues.

Can't homedown when escaping to the shell

The terminal is not able to do a homedown before executing the shell command.

WHATCHAR

Control character CONTROLCHARACTER (nnD,0xnn,0nnn)

The character under the cursor is a control character, with the given decimal (nnD), hexadecimal (0xnn), and octal (0nnn) values.

Space character (nnD,0xnn,0nnn)

The character under the cursor is the space character, with the given decimal, hexadecimal, and octal values.

Normal character CHARACTER (nnD,0xnn,0nnn)

The character under the cursor is a normal character, with the given decimal, hexadecimal, and octal values.

ESC character (nnD,0xnn,0nnn)

The character under the cursor is the ESC character, with the given decimal, hexadecimal, and octal values.

DEL character (nnD,0xnn,0nnn)

The character under the cursor is the DEL character, with the given decimal, hexadecimal, and octal values.

Unknown character (nnD,0xnn,0nnn)

Editor cannot display type of character under cursor if greater than 127 decimal (7F hex, 177 octal).

CTRL d

Do you really want to quit?

This query occurs when a CTRL d is pressed. If the user wishes to leave the editor, a 'yes' or 'y' answer is needed. Otherwise, the edit session continues.

Answer is too long

The answer to the question was too long.

No changes lost, edit resumed

The answer to the question was something other than 'yes' or 'y'.

CTRL r or CTRL b

Recall buffer empty

No commands have been placed in the recall buffer, so nothing can be recalled.

Recall buffer broken

Internal problem with the recall command.

skpreserve status/error messages

usage: skpreserve

The skpreserve command does not allow any options to be used in invoking it.

Unable to open temporary directory DIRECTORY

The directory where the crashed edit sessions reside could not be opened. This may be due to a protection problem.

Couldn't copy file TMPFILE to PRESERVEFILE

Due to being on different file systems, an attempt was made to copy the temporary file to the preserve directory. This was not possible for some internal reason.

Couldn't link file TMPFILE to PRESERVEFILE

Some internal error occurred while trying to link the temporary file to the preserve file.

Couldn't unlink TMPFILE

Some internal problem has prevented the link from being removed.

Error while trying to open pipe for mailing

Some system error caused the mail message to not be sent.

Error on closing pipe

Some system error caused the pipe, used to mail the message, to close improperly.

skrecover status/error messages

usage: skrecover [-d] [-r] [-u] [-n fileid] [-t number] [-f file] [file]

User did not follow the syntax for the command. For example, the '-d' option may not be used with any other option.

No filename available, use -f option

Recovery of an unnamed edit session requires that the -f option be used to give the recovered file a name.

Can't find recovery file for an unnamed file

The user is trying to recover an unnamed file and that file cannot be found.

Can't find recovery file for FILE

The user is trying to recover a file and that file cannot be found.

unknown option: NUMBER

The recovery file has bad data in it, so complete recovery may not be possible.

Recovery file incomplete, some data may be lost.

The recovery file is missing some data, so complete recovery may not be possible.

copy: error
delete: error
extract: error

An error occurred during the recovery process. Some data may be lost.

File FILE exists, no save made

The -u option was not used, so an existing file will not be overwritten. $HOME/.recover directory not accessible.

Protection on $HOME/.recover directory has been set so user can't modify it, or the $HOME/.recover directory doesn't exist.

Maximum number of recoverable files available is NUMBER

The shell variable MAXREC was set to a bad value or a number greater than 128.

Can't update recoverable files directory

Protection on $HOME/.recover/directory has been set so user can't modify it.

Can't set permissions on recoverable list

Protection on $HOME/.recover/directory has been set so user can't modify it.

Unable to open temporary directory DIRECTORY

The temporary directory where the crashed edit sessions originally resided could not be opened. This may be due to a protection problem.

Can't unlink FILE

Some internal problem has prevented the link from being removed.

Unable to write file FILE

The skrecover command is not able to write the file due to either an existing file or not being able to open a file to write to.

Unable to purge existing file FILE, no save made

The purging of an existing file did not work, so the edit session could not be recovered.

Disk full, some data was lost

The system disk being written to is full, so the recovery of the edit session is not possible.

Purge status/error messages

no $HOME shell variable

The user does not have a $HOME shell variable defined and exported.

Maximum number of records available is NUMBER

The shell variable MAXREC was set to a bad value or a number greater than 128.

Can't open recover file directory

Protection on $HOME/.recover directory has been set so user can't modify it, or the $HOME/.recover directory doesn't exist.

Can't copy file across file system

An error occurred while trying to copy the file between file systems.

Can't link to file

Some internal problem has prevented the link from being made.

Can't unlink old file

Some internal problem has prevented the link from being removed.

Can't update recoverable files directory

Protection on $HOME/.recover/directory has been set so user can't modify it.

Can't set permissions on recoverable list

Protection on $HOME/.recover/directory has been set so user can't modify it.

FILE purged

The purge to the $HOME/.recover directory worked.

Can't purge FILE

The purge command was unable to place the file in the $HOME/.recover directory.

rcvr status/error messages

usage: rcvr [-f newfile] file [file2 ...]

Indicates when bad options are used.

Only one file may be recovered when using -f option

Recovery of more than one file was attempted while using the -f option, which is not allowed.

no $HOME shell variable

The user does not have a $HOME shell variable defined and exported.

file FILE not found

The file does not exist in the recovery directory. This may be due to the fact that an absolute path must be matched. If a relative file name is entered as part of the rcvr command, then the current directory will be used as part of the absolute path to that file.

File FILE already exists

The rcvr command will not overwrite an existing file.

Can't open recover file directory

Protection on $HOME/.recover directory has been set so user can't modify it, or the $HOME/.recover directory doesn't exist.

Not able to copy file across file system

An error occurred while trying to copy the file between file systems.

Not able to link to old file

Some internal problem has prevented the link from being made.

Can't unlink recoverable alias

Some internal problem has prevented the link from being removed.

Can't update recoverable files directory

Protection on $HOME/.recover/directory has been set so user can't modify it.

Can't set permissions on recoverable list

Protection on $HOME/.recover/directory has been set so user can't modify it.

FILE recovered

The recovery worked for the given file.

FILE not recovered

The rcvr command was unable to recover the file.

dirrec status/error messages

No personal recovery directory

The user does not have a $HOME/.recover directory.

Installation Notice

Editor Command Truth Tables

Introduction

Editor truth values in the following table are related to the "Test Clause" which is executed at the beginning of each iteration of the while command. It also affects the "Body Clause" as to what truth value is returned at the end of the while command. For more information on the while command, refer to the syntax in chapter 5.

Editor Command Truth Values

Command	Value
autotab	true
copy	true if lines copied false if 0 lines copied
delete	true if lines deleted false if 0 lines deleted
extract	true if lines extracted false if 0 lines extracted
find	true if string found false if string not found
insert	true
join	true if lines joined false if no lines joined
list	true if lines listed false if 0 lines listed

Command	**Value**
< LINE+ ->	true if line is found false if line not found
merge	true if file merged false if no lines merged
range	true
renumber	true if lines renumbered false if 0 lines renumbered (empty file)
repeat	true if lines repeated false if 0 lines repeated
replace	true if strings replaced false if no strings replaced
retrieve	true if lines retrieved false if 0 lines retrieved
split	true if line split false if no split occurs
tabset	true
while	true if body is not empty and all statements in body true during last loop; or true if body is empty false if body is not empty and one or more statements were false during last loop

Introduction

The following editor commands have no truth value and may NOT be used in multiple constructs:

cd
< CMDFILE>
column_numbers
edit
end
help
log_commands
save
wait
!

Installation Notice

Comparison with HP 64000 Editor

Introduction

The Softkey Driven Editor is based on the HP 64000 workstation editor, but has been enhanced to take advantage of your development environment. If you are familiar with and have used the HP 64000 workstation editor, you will note some differences between that editor and the Softkey Driven Editor. These differences and others of less significance are referenced where appropriate throughout this manual. The following is a summary of some differences and a description of how this may affect your approach to using the Softkey Driven Editor.

List Command

In the HP 64000 workstation editor, the **list** command writes to a file with the "listing" extension, which file would always be overwritten. On the operating system, these extensions are no longer separate from the file name. For user file protection, on the Softkey Driven Editor, the **list** command appends to existing text unless the user specifically directs the command to overwrite the file; this is intended to minimize text loss. Also, if a $HOME/.recover directory exists, a copy of an existing file will be placed there before the listing takes place.

Control Characters

In the HP 64000 workstation editor, control characters are entered with a special key and are displayed with a special character on-screen. With the Softkey Driven Editor, control characters are entered with a two-step sequence. First you press **CTRL v**, followed by the control character. For example, with the Softkey Driven Editor, you would enter **CTRL L** by pressing the following sequence of keys: **CTRL v CTRL L**. This will generate a "." which represents the control character. If this "." is on the command line, then the user will be unable to identify whether it is a period or a control character. If the "." is in the text area, the **WHATCHAR** softkey may be used from the **REVISE*** or **INSERT*** mode to identify it.

Command Separators and Comments

In the HP 64000 workstation editor, multiple commands are separated by a backslash " \ " and comments are started with a semicolon " ; ". Because the operating system uses the backslash as an escape character, multiple commands on the Softkey Driven Editor are separated by semicolons " ; ". On the Softkey Driven Editor, comments start with a number symbol " # ". Both editors allow comments anywhere on the command line and they continue until the end of the line.

Command Files

Command files on the HP 64000 editor were chained, whereas on the Softkey Driven Editor, command files are nestable. This means that on the Softkey Driven Editor, when a command file is invoked from another command file, the original command file waits for the second command file to finish before it continues. If the command file call is the last line in the command file, then it will "chain" like the HP 64000 editor. On the HP 64000, the call to the second command file causes the first command file to terminate. Because of this

difference, it may be necessary to do some editing of HP 64000 command files to ensure that they will run with the Softkey Driven Editor.

Find and Replace Commands

In the HP 64000 workstation editor, the anychar and anystring softkeys produce an enhanced "c" and "s", respectively, on the command line. These are, in reality, **CTRL X** and **CTRL V** characters.

In the Softkey Driven Editor, the anycharacter and anystring softkeys still exist, but they produce the characters '?', for anycharacter, and '*', for anystring.

Editing of existing HP 64000 command files will be necessary to convert these control characters to their new equivalents. Also, if '*' or '?' characters were used in the old replace or find strings, then they will need to be escaped with the ' \ ' backslash character so the character '*' or '?' will be matched, rather than the anystring or anychar. The following command file that operates on the current edit file might be useful (these commands might cause unexpected results if the **find** or **replace** commands are on the same line as other editor commands):

```
#  escape '?' and '*' characters in find commands
0
while find 'find' do replace "\?" with "\\\?"; replace "\*" with "\\\*"; -1 doend
#  escape '?' and '*' characters in replace commands
0
while find 'replace' do replace "\?" with "\\\?"; replace "\*" with "\\\*"; -1 doend
#  convert ^ x and ^ v to ? and * in find commands
0
while find 'find' do replace "\030" with "?"; replace "026" with "*"; -1 doend
#  convert ^ x and ^ v to ? and * in replace commands
0
while find 'replace' do replace "\030" with "?"; replace "\026" with "*"; -1 doend
```

Other Differences

There are some other minor differences between the Softkey Driven Editor and the HP 64000 workstation editor. The following is a list of some other differences, which may be helpful reminders if you have previously worked with the HP 64000 editor.

- Operation of the up-arrow and down-arrow keys. In the Command mode of the Softkey Driven Editor, these keys operate differently. Refer to "Keyboard Input And Functions" in chapter 4.
- Use of a "placeholder" symbol for Insert Char: There is no "placeholder" symbol in the Softkey Driven Editor, whereas there was an underlined caret symbol for this function in the HP 64000 editor.

Index

Notice

Hewlett-Packard 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. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

UNIX is a registered trademark of AT&T.

TORX is a registered trademark of Camcar Division of Textron, Inc.

Hewlett-Packard Company
Logic Systems Division
8245 North Union Boulevard
Colorado Springs, CO 80920, U.S.A.

Printing History

New editions are complete revisions of the manual. The date on the title page changes only when a new edition is published.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual revisions.

Edition 1	64790-90901 E0686, June 1986 (Preliminary)
Edition 2	64790-90901 E0587, May 1987
Edition 3	64790-97000, July 1989
Edition 4	64790-97001, July 1990
Edition 5	64790-97002, February 1991
Edition 6	**B1444-97000, November 1991**