UniFLEX® SCREEN EDITOR

technical systems consultants, inc.

Please note that terminals from Southwest Technical Products Corporation in the CT8200 and X12 series have a key labeled TAB which, rather than producing a control-I, produces the escape sequence, control-[. As a convenience to our users with these terminals, USE treats two sequential escape sequences as a tab sequence.

On the word-processing versions of the CT8200 series, the key labeled NEXT PARA produces a control-I; on X12 terminals, the arrow that points to the right on the numeric pad does.

UniFLEX® SCREEN EDITOR

MANUAL REVISION HISTORY

Revision	Date	Change
A	9/85	Original Release, UniFLEX Screen Editor

DISCLAIMER

The supplied software is intended for use only as described in this manual. Use of undocumented features or parameters may cause unpredictable results for which Technical Systems Consultants, Inc. cannot assume responsibility. Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible, Technical Systems Consultants, Inc. will not assume responsibility for any damages incurred or generated by such material. Technical Systems Consultants, Inc. reserves the right to make changes in such material at any time without notice.

Contents

Contents

Contents

Preface

The design of the UniFLEX® Screen Editor (USE) is based on that of the Programma Improved Editor (PIE). It provides a versatile means of editing both source code and documentation and supports a variety of features, which enable the user to configure the editor most suitably for the task at hand. Technical Systems Consultants would like to acknowledge the help of Kenneth R. Lewis and Shawn J. Morrisey in the design and implementation of this product.

The editor is available for computers using either the 6809 or 68xxx UniFLEX Operating System. This manual documents all versions of the editor as the differences between them are slight. The first chapter of the manual contains instructions for the system manager. The second chapter is a tutorial designed to introduce users who have little or no experience with any screen editor to this particular one. The third chapter is a reference guide to the editor. It includes an overview of the features of the editor and detailed documentation of each command the editor supports. Certain conventions are used throughout this manual in the descriptions of these commands and of the editor itself. These conventions are explained in Appendix A.

The UniFLEX Screen Editor is in its own right a powerful, stand-alone piece of software. The user can, however, create an even more powerful tool by using it in conjunction with our text-formatting program, "pr".

Chapter 1

System Manager's Guide to the UniFLEX Screen Editor

1.1 Installing the Editor

Follow the instructions for installation on the yellow sheet of paper that was included with the disk. This procedure installs the editor as the file "/bin/use" with permissions rwxr-x. The owner of the file is "bin". Because of the way the permissions are set on the directory "/bin", only the system manager can delete the editor from the system.

1.2 Sample File

The installation procedure puts a file named "usesample" in the root directory. This file is for use with Chapter 2 of this manual, "A USE Tutorial". The tutorial instructs the user to copy this file into her or his home directory and to use only the copy for practice with the editor. Thus, the original file should remain intact.

1.3 Terminal Capabilities

The UniFLEX Screen Editor is compatible with most ASCII terminals with the capability to address the cursor directly. The number of lines displayed in the edit window may vary from terminal to terminal, but the editor always displays a total of eighty columns. If the terminal cannot accept eighty characters in one line, the results are unpredictable.

The editor uses the file "/etc/termcap", which defines the capabilities of each terminal on the system, in establishing communication with any given terminal. The editor cannot function if this file is nonexistent or if the terminal from which it is invoked is not described in the file.

All versions of the UniFLEX Operating System from Version 12 (released August 1, 1985) on include the utility "crt_termcap", which creates the necessary file.

Chapter 2

A USE Tutorial

2.1 Introduction

This chapter is an introduction to the UniFLEX Screen Editor. It is designed for people who have not used a screen editor before. It does assume, however, some basic familiarity with the UniFLEX Operating System. If you are not familiar with it, you should read either the introduction in the manual for the operating system or, for 6809 systems, the manual entitled 6809 UniFLEX: A Tutorial.

The "use" command invokes the UniFLEX Screen Editor, which allows you to create and put information in a new file or to modify an existing file. What is a file? In simple terms a file is a collection of information or data that is kept on the system and given a name for later reference. It is readily accessible because the operating system recognizes its name and knows where to look for the contents of any particular file.

A file may contain binary or textual data. A file which contains textual data is a "text file". Such files may be used to store things like documents and letters.

Letters and documents which are typed on paper may be edited in several ways, including the use of correction fluid or of scissors and paste. A screen editor is a sophisticated tool which allows you to perform similar functions on a text file easily and efficiently. The simplest way to learn about the editor is to use it. This chapter will lead you through some simple manipulations in a text file in order to acquaint you with some of the features of USE. You will find more detailed explanations of the capabilities of the editor in Chapter 3.

2.2 Syntax Statements

Each explanation of a command is accompanied by a generalized illustration of its use known as a syntax statement. Syntax is the structure the command must adhere to in order for the editor to recognize it. Throughout this document certain conventions are used in syntax statements. You must be familiar with these conventions in order to interpret syntax statements correctly.

2.2.1 Control Characters

The commands supported by the editor are based on "control characters", that is, they are defined by keys which produce nonprintable characters. Many control characters are not represented by a single key on the keyboard. Rather, you enter such a character by depressing the key marked "control" (or "ctrl") and holding it down while you type another character. In this document such a control sequence is referred to by the string

control-<char>

where <char> is the name of the key to press while holding down the control key. Alphabetic control characters are shown with uppercase letters because that is what is usually on the keyboard. You may however type either the upper- or lowercase letter.

The more commonly used control characters--such as the escape, carriage return, and tab keys--usually do have individual keys on the keyboard. In this document whenever a control sequence is referred to by a name such as "tab" in a syntax statement or an example, the name appears entirely in uppercase letters. This convention tells you to type the key corresponding to the name, not the name itself.

Your keyboard may also have keys labeled with the names of other commands--such as "home" and "replace". These keys may or may not send the same control character as the commands of the same name documented in this manual. You will have to experiment with your terminal or look in the manual that came with it in order to determine what control character a given key represents.

2.2.2 Syntax Conventions

You must type any characters other than space characters and strings representing control characters that appear in a syntax statement and that are not enclosed in angle brackets, '<' and '>', or square brackets, '[' and ']', exactly as shown.

Characters that appear in angle brackets are not meant to appear as is when you use the command. Instead, you must replace them with a specific example of the kind of argument they describe. For instance, the command you will use to copy the sample file to your home directory contains the argument <your_name>. In such a case, what you should

actually type is your user name--without the angle brackets.

Characters enclosed only in angle brackets are an essential part of the command. Characters enclosed in square brackets are optional. The underscore character, '_', is used to link separate words that describe one entity, such as "your" and "name".

Finally, most of the syntax statements in this document contain space characters in order to make them easier to read. However, when you are actually invoking a command, you should not type any space characters between the parts of the command. If you do, your results may differ from those described in the manual.

2.2.3 Remembering Syntax Statements

Each time this document introduces a command, it gives the appropriate syntax statement. The next time the command is used outside of the section in which it was introduced, the syntax statement appears a second time. After that, the text assumes that you remember the syntax of the command. This assumption is not entirely fair, but it does simplify the writing of the manual. If you find the manual telling you to invoke a command but you do not remember the syntax for it, refer to Appendix C, which lists all the commands supported by the editor in alphabetical order, accompanied by their syntax statements.

2.3 Invoking the Editor

The UniFLEX Screen Editor comes with a sample file that is used for examples throughout this chapter. The complete file specification for this file is "/usesample". You should make a copy of this file in your home directory for your personal use. You can do so by issuing the following command:

```
copy /usesample /usr/<your_name>/sample
```

This command creates a file in your home directory named "sample". The file is now identical to the sample provided with the system, but you will soon begin to change it.

The simplest form of the syntax for invoking the editor is

use <file_name>

Now that you have a file to work on, you can invoke the editor with the following command:

use sample

This command tells the operating system to invoke the editor and to prepare the file named "sample" for editing. The next section begins to explain the display that should now be on your screen.

2.4 The Screen Display

When you invoke the editor, it writes as much as possible of the file you specify into a place in memory called the edit buffer and displays a screenful of information from the beginning of the file. Although the number of lines displayed in the window may vary (it is typically 21), the editor always displays a total of eighty columns (including one column for the left-hand boundary of the window and one for the right-hand boundary). If the terminal cannot accept eighty characters in one line, the results are unpredictable). The editor tries, based on the information in the file "/etc/termcap", to display as much text as possible on whatever terminal you use.

Because the text of this document contains only seventy-two columns, a slight discrepancy between the figures and the image you will see on your screen may occur. The discrepancy involves only the top line of the edit window and the number of blank columns at the right-hand edge of the text. It does not affect any of the manipulations of text undertaken in the tutorial.

2.4.1 The Edit Window

The information that the editor displays is surrounded by a rectangular box known as the edit window. In this case each side of the box is composed of a series of vertical bar characters, '|'. The bottom is composed of minus signs, '-'. The top may contain four different characters. A plus sign, '+', indicates a column containing a tab stop. A vertical bar, '|', indicates the column in which the warning bell

sounds (like the warning bell on a typewriter) if that column does not also contain a tab stop. A number sign, '#', indicates the column in which the warning bell sounds if that column does contain a tab stop. Each remaining column contains a minus signs.

2.4.2 The Cursor

The cursor is a character which indicates where on the screen the next character you enter directly from the keyboard will appear. When the editor first displays the contents of a file, it positions the cursor at the first column of the first line of the display. The line containing the cursor is referred to as the current line; the column containing the cursor, as the current column.

The shape of the cursor varies from terminal to terminal. Most often it is a block or an underscore character, '_'. Your terminal can superimpose the cursor over text so that you can see the character beneath it even if the cursor is a block character. A printer, however, cannot display both a block character and the character underneath it. This document, therefore, uses the underscore character to indicate the cursor so that it does not obliterate the character at the same position in the text.

2.4.3 The Status Line

Below the edit window is one more line--the status line. By default, this line contains the strings "Col" and "Line", each followed by a number. The numbers indicate the position of the cursor in the file. Both lines and columns are numbered starting with 1.

The editor also uses the status line to display informative messages, prompts, and error messages.

2.5 Positioning the Cursor

The capabilities of the screen editor center around the position of the cursor in the edit buffer. It is therefore crucial for you to learn how to move the cursor about. We will begin the simplest of the cursor movements, those that are governed by the keys which are collectively

referred to as the cursor-positioning keys. These keys change the position of the cursor without affecting the text of the file.

When you invoke the editor for the first time, calling on it to edit the sample file (see Section 2.3), your screen should look like Figure 2-1 (assuming twenty-one lines and seventy-eight columns and allowing for the discrepancy described in Section 2.4).

```
                                                    111 Providence St.
                                                    Chapel Hill, NC
                                                    27514

Your name
Your address
City, State  Zip

Dear So-and-So:

     Congratulations on your decision to purchase our
new screen editor, "use", for your Operating System.
The acronym "use" stands for the UniFLEX Screen
Editor.  You can use the editor in conjunction with
our text-formatting program, "pr", or as a
what-you-see-is-what-you-get editor.  We hope that by
the time you finish editing this letter you will feel
-------------------------------------------------------------------------
 Col 1                                                          Line 1
```

Figure 2-1. Sample file.

The cursor is at line 1, column 1. For the time being, while you are learning about the cursor-positioning keys, avoid moving the cursor below line 21 or past column 78.

2.5.1 The "home" Command

The "home" command performs two separate tasks, depending on the position of the cursor when you invoke the command. If the cursor is not in the first column of the first line in the edit window, the "home" command puts it there. If the cursor is already in the first column of

the first line in the edit window, "home" moves it to the first column of the last line of the window. Thus, a sequence of "home" commands moves the cursor back and forth between the two positions. It is said to "toggle" from one interpretation of the command to another. Several other commands supported by the editor behave in a similar manner.

The syntax for the "home" command is

control-D

Now type this command several times until you are satisfied that you understand its behavior. Notice that the line number in the status line changes when you invoke the "home" command. Leave the cursor in column 1, line 1. Your screen should now look as it did when you first invoked the editor (see Figure 2-1).

2.5.2 The "return" Command

The "return" command positions the cursor at the beginning of the line following the current line. Most keyboards have a key labeled "return", "carriage return", or "enter", but you can generate this command by typing control-M. Type "return" several times, but be careful not to go beyond line 21 for the time being. Each time you type "return" not only does the cursor move to the next line but also the number in the status line indicating the number of the current line changes.

2.5.3 The "right" Command

The "return" command always positions the cursor at the first column of a line. You can move the cursor to the right one column at a time with the "right" command. The syntax for this command is

control-F

Try it. As you can see, the text under the cursor is not affected. Experiment with this command, but do not try to move the cursor beyond column 78. Notice that the column number in the status line changes each time you execute the "right" command.

2.5.4 The "left" Command

Analogous to the "right" command is the "left" command, which moves the cursor to the left one column at a time. The syntax for this command is

control-S

Do not, for the time being, try to move the cursor to the left of column 1.

2.5.5 The "rtab" Command

Sometimes it is convenient to move the cursor more than one column at a time. The "rtab" command moves the cursor to the next column that contains a tab stop. As mentioned previously the top margin of the edit window contains a plus sign, '+', in each column containing a tab stop. The syntax for the "rtab" command is

control-G

Again, experiment with this command, but do not try to move the cursor beyond column 73 (the last column containing a tab stop before column 78).

One of the more useful features of the editor is its ability to respond differently to the "rtab" command. If the editor is in word-tab mode, the command moves the cursor to the first character of the next word rather than to the next tab stop. By default, the editor considers that a word is entirely composed of alphanumeric characters (the letters of the alphabet and the digits '0' through '9'). It considers all other printable characters to be punctuation. Later in the tutorial you will use the "word_def" command to modify this definition of a word.

The "word_tab" command, like the "home" command, toggles between two functions--putting the editor into and taking it out of word-tab mode. Its syntax is

TAB w control-A

This command is typical of several of the commands that USE supports in that it consists of two control sequences. In order to invoke the "word_tab" command you must first type the tab key, or if your terminal

does not have one, control-I. When you do so, several things happen at once: an "at" symbol, '@', replaces the cursor; the string "Arg: " replaces the column and line indicators in the status line; and the cursor moves to the status line. The argument for this particular command is a 'w'. Because the 'w' is not enclosed in square brackets in the syntax statement, you know that it is not an optional argument, and because it is not enclosed in angle brackets, you know that you must type it as is (see Section 2.2.2). The 'w' should be immediately followed by a control-A. Remember not to type any space characters between the parts of the command. Those shown in the syntax statement are there purely to make it easier for you to understand the syntax.

If you have not already typed the command, do so now. Notice that when the editor is in word-tab mode, the message "Word Tab" appears in the status line. Now, if necessary, move the cursor so that it is on a line of the file that contains some text. Experiment with the "rtab" command until you become familiar with its behavior in word-tab mode as well as in the default mode.

When you are ready to continue, type the "word_tab" command again. The message "Word Tab" disappears from the status line, indicating that the editor is now in the default mode for interpreting the "tab" commands.

2.5.6 The "ltab" Command

Analogous to the "rtab" command is the "ltab" command, which moves the cursor to the left until it reaches a column containing a tab stop. The syntax for this command is

control-A

Experiment with the "ltab" command a bit. Once you are satisfied that you understand its behavior, use the "word_tab" command to put the editor in word-tab mode:

TAB w control-A

The "word_tab" command and the "ltab" command are based on the same control character, control-A. The unmodified control character is the simple "ltab" command. The "word_tab" command (and several others) use an argument to modify the function of the control-A. All these commands are related to the tab functions in some way. Similarly, other groups of commands which are related to each other in function share the same

control character. Appendix D lists all the commands supported by the editor grouped according to control character.

Experiment with the "ltab" command until you are comfortable with its use. Notice that whether or not the editor is in word-tab mode, you cannot "ltab" to the left of column 1.

If you are using the editor to write source code, you will probably prefer to leave it in the default mode. If, on the other hand, you are using it to write text, you will usually find it more convenient to leave it in word-tab mode. Since the tutorial is concerned with editing a letter, leave the editor in word-tab mode.

2.5.7 The "up" Command

When you typed the "return" command, you moved the cursor down several lines in the edit window. You can move it towards the top of the edit window one line at a time with the "up" command. The syntax for this command is

control-E

Use the "up" command repeatedly until the cursor is back at line 1. Unlike the "return" command, the "up" command does not affect the horizontal position of the cursor.

2.5.8 The "down" Command

Analogous to the "up" command is the "down" command, which moves the cursor down one line without altering its horizontal position. The syntax for this command is

control-C

Practice the "down" command, but, for now, do not try to move the cursor beyond line 21.

2.5.9 What's That Command?

You have just been introduced to eight separate commands and may well be wondering how you are going to remember which key performs which command. Because most keyboards have a "return" key, remembering that one should not be a problem. The other seven are strategically positioned on the keyboard so that they are easy to remember. With control-D at the center, they form a cross in which the position of the alphabetic key corresponds to its function. Thus, the control key above control-D (control-E) moves the cursor up, while the key below control-D (control-C) moves the cursor down. The keys to the right move the cursor one column and one tab stop to the right while those to the left behave correspondingly.

2.5.10 The "app" Command

One of the cursor-positioning commands, the "app" (append) command, does not fit this easy-to-remember pattern. The syntax for this command is control-]. The "app" command, like the "home" command, toggles between two functions. If the cursor is not in the column that contains the last printable character of the current line, it moves to that column. If it is already there, it moves to column 1 of the current line.

It is not always obvious how the editor determines which column contains the last character in a line. When you enter any printable character (including the space character) in a line, the editor enters a space character in each column between the character you entered and the column which previously contained the last character. If you subsequently erase the character you entered, the space character preceding it becomes the last character in the line. However, when you save a file, the editor truncates any trailing blanks (space characters between the last printed character in in the line and the carriage return).

Now practice with the cursor-positioning keys until you feel comfortable with them. Remember that to avoid confusion you should try to stay within the confines of the edit window. If you do accidentally go beyond its boundaries, you should be able to move the cursor so that it is back within the original boundaries, but in the meantime things that have not yet been explained will be happening to the display on your screen.

2.6 The "goto" Command

Your cursor is currently wherever you left it when you decided you had mastered the cursor-positioning keys. In order to proceed, we have to get everyone who is using the tutorial back to the same place in the file. We could have you laboriously use the newly mastered cursor-positioning keys to get to a particular line and column. However, it is simpler to use the "goto" command. The "goto" command allows you to position the cursor anywhere in the edit buffer. The editor supports four variations of this command, three of which move the cursor vertically through the file. The fourth moves it horizontally. We will begin with the simplest one.

2.6.1 The "top" Command

The simplest form of the "goto" command, "top", positions the cursor at the first column of the first line of the edit buffer and places that line at the top of the edit window. (Because the sample file is a short file, it all fits in the edit buffer at one time. Thus, the first and last lines in the edit buffer are, in this case, also the first and last lines of the file. For a large file, things are different [see the description of the "flush" command in Chapter 3]). The syntax for the "top" command is

control-T

Try it. The cursor should now be in column 1, line 1. Your screen should look as it did when you first entered the editor (see Figure 2-1).

2.6.2 The "bottom" Command

The next form of the "goto" command, "bottom", positions the cursor at the first column of the last line of the edit buffer and places that line at the bottom of the edit window. The syntax for this command is

TAB control-T

First, type the tab key. When the prompt for an argument appears in the status line, type control-T. The editor adds the message

Busy

to the status line while it executes the command, as it does when it executes any of its more time-consuming tasks. When it completes the command, your screen should look like Figure 2-2.

```
the time you finish editing this letter you will feel
fairly comfortable with many of the screen-editing
features of "use".  It really is not hard for to learn
to use!  We also hope, So-and-So, that it will serve you
very well in your work from day to day.

     The object of this tutorial is to familiarize an
inexperienced user with the more commonly used features of
inexperienced user with the more commonly used features of
editor.

     The reference guide (which is in the second chapter
chapter of the manual) contains more detailed
information about the Screen Editor and the commands
it supports.

                                                Sincerely yours,



                                                TSC
------------------------------------------------------------------
 Col 1                          Word Tab              Line 41
```

Figure 2-2. Effect of the "bottom" command.

2.6.3 The "line" Command

A slightly more complicated form of the "goto" command allows you to position the cursor at the first column of any line in the edit buffer. We will use this command to position the cursor at the beginning of line 15. The syntax for the "line" command is

TAB <line_number> control-T

First of all, type the tab key. When you see the cursor in the status line, type the number of the line you wish to go to (in this case 15). If you make an error entering the number of the line, use the backspace key (control-H) to erase the incorrect portion of the number and retype

it. Then type control-T. In response to this command, USE moves the cursor to the first column of line 15 and places line 15 approximately one third of the distance down the edit window (in our case, at the eighth line).

Be careful not to type a space character after the number you use as an argument to the command. If you do, USE returns the following message in the status line:

Invalid parameter

If this message appears, simply start the command over again by typing the tab key. When you do so, the message disappears. You will get the same message if you use a negative number or 0 as the line number. If, however, you use a number greater than the line number of the last line in the edit buffer, USE behaves as if you had executed the "bottom" command.

Now practice moving the cursor to various lines in the file until you are confident you understand all the "goto" commands introduced so far.

2.6.4 The "column" Command

The fourth form of the "goto" command is a variation of the "right" command which allows you to place the cursor in any column of the current line. The format for this command is

TAB <column_number> control-F

Let's position the cursor at column 55 of line 37. First, use the "line" command to get to line 37:

TAB 37 control-T

Now use the "column" command to get to column 55:

TAB 55 control-F

The information in the status line tells you whether or not you have positioned the cursor correctly. Your screen should now look like Figure 2-3.

```
editor.

    The reference guide (which is in the second chapter
chapter of the manual) contains more detailed
information about the Screen Editor and the commands
it supports.

                                                Sincerely yours,



                                                TSC
```

```
Col 55                          Word Tab            Line 37
```

Figure 2-3. Positioning the cursor at column 55 of line 37.

You can use the "column" command to move the cursor to the right or to the left. Again, if you use a negative number or the number 0 as the argument, you receive an error message from USE in the status line. Now practice with all four "goto" commands: "top", "bottom", "line", and "column". When you are comfortable using them, position the cursor at column 6 of line 15, so that your screen looks like Figure 2-4.

```
-+------+------+------+------+------+------+------+------#------
|                                                               |
|Your name                                                      |
|Your address                                                   |
|City, State  Zip                                               |
|                                                               |
|Dear So-and-So:                                                |
|                                                               |
|     Congratulations on your decision to purchase our          |
|new screen editor, "use", for your Operating System.           |
|The acronym "use" stands for the UniFLEX Screen                |
|Editor.  You can use the editor in conjunction                 |
|with our text-formatting program, "pr", or as a                |
|what-you-see-is-what-you-get editor.  We hope that by          |
|the time you finish editing this letter you will feel          |
|fairly comfortable with many of the screen-editing             |
|features of "use".  It really is not hard for to learn         |
|to use!  We also hope, So-and-So, that it will serve you       |
|very well in your work from day to day.                        |
|                                                               |
|     The object of this tutorial is to familiarize an          |
|inexperienced user with the more commonly used features of     |
---------------------------------------------------------------------
 Col 6                              Word Tab            Line 15
```

Figure 2-4. Positioning the cursor at column 6 of line 15.

2.7 Moving through the File

The edit window shows you only a portion of the file you are working on. You can think of it as a magnifying glass which you can move about the file in order to look at any particular part of it in detail. However, because of the standard terminology which is in use, it is better to think of the edit window as staying in one place and of the file as a scroll, which can be pulled in either direction beneath the window. The editor supports several commands which allow you to "scroll" through the file.

2.7.1 Scrolling Up

If you keep clearly in mind the image of the file as a scroll moving beneath the edit window, you will soon become used to the terminology associated with the scrolling commands, which can be confusing at first. Scrolling up means that you are pulling on the top part of your

imaginary scroll. Thus, the text that appears in your edit window is text nearer to the end of the file. Actually, playing with the scrolling commands is easier than trying to read about them, so let's get started.

2.7.1.1 Full page

The "page_up" command scrolls the file up one screenful. The syntax for this command is

control-V

The cursor should now be in column 6 of line 15 (the eighth line in the edit window), with line 8 at the top of the edit window (see Figure 2-4). Now type the "page_up" command. If your terminal is standard, after you type "page_up", line 29 appears at the top of the edit window, as shown in Figure 2-5.

```
inexperienced user with the more commonly used features of
editor.

     The reference guide (which is in the second chapter
chapter of the manual) contains more detailed
information abut the Screen Editor and the commands
it supports.

     _
                                                Sincerely yours,



                                                TSC
```

Col 6 Line 36

Figure 2-5. Effect of the "page_up" command.

The position of the cursor within the edit window does not change when you invoke the "page_up" command. It is still in the sixth column of the eighth line in the edit window, but the eighth line is now line 36.

Try typing "page_up" again. This time you are met by a blank screen, and the status line tells you that the cursor is at column 6 of line 49 (the eighth line in the edit window). Obviously, the edit window did not move as much as before. Why not? The "page_up" command moves the text past the edit window a screenful at a time unless it encounters the end of the file. In that case it positions the text so that the first line in the window is the line immediately following the last line of the file. You can verify this by typing "home" (control-D), which puts the cursor in the first column of the first line of the edit window, followed by "up" (control-E), which moves the cursor up one line. You should now see the last line of the file, line 41, at the top of the edit window.

2.7.1.2 Partial screen

At times you may wish to scroll through the file less than a full screen at a time. The "sup" (scroll up) command moves the text in the same direction as the "page_up" command, but it moves only one third of a screenful at a time. In our case, it moves the text seven lines at a time.

The "sup" command handles the placement of the cursor a bit differently than does the "page_up" command. If the current line of text remains on the screen after the execution of the "sup" command, the cursor remains on that line of text. Thus, its position with regard to the edit window changes. If, on the other hand, execution of the "sup" command forces the current line off the screen, the editor places the cursor on the line at the top of the edit window. The "sup" command always maintains the horizontal position of the cursor.

Before trying the "sup" command, use the "top" command (control-T) to position the cursor at the top of the file. Now use the "down" command (control-C) to position it at line 13. (You could more efficiently position the cursor with the "line" command, but that would put line 13 in the eighth line of the edit window and it would scroll to the top of the screen at the first invocation of the "sup" command, thereby lessening the visual distinction between the two different ways in which "sup" handles the positioning of the cursor.) Finally, use the "column" command to move the cursor to column 6.

TAB 6 control-F

Your screen should look like Figure 2-6.

```
-+------+------+------+------+------+------+------+------#------
|
|
|
|
|                                                  111 Providence St.
|                                                  Chapel Hill, NC
|                                                  27514
|
|Your name
|Your address
|City, State  Zip
|
|Dear So-and-So:
|
|     Congratulations on your decision to purchase our
|new screen editor, "use", for your Operating System.
|The acronym "use" stands for the UniFLEX Screen
|Editor.  You can use the editor in conjunction with
|our text-formatting program, "pr", or as a
|what-you-see-is-what-you-get editor.  We hope that by
|the time you finish editing this letter you will feel
-------------------------------------------------------------------
 Col 6                              Word Tab          Line 13
```

Figure 2-6. File before using the "sup" command.

The syntax for the "sup" command is

control-N

Type the command and watch as the editor scrolls the file up seven lines so that line 8 is now at the top of the page. The cursor is still in column 6 of line 13 as shown in Figure 2-7.

```
-+------+------+------+------+------+------+------+------#------
|
|Your name
|Your address
|City, State  Zip
|
|Dear So-and-So:
|
|      Congratulations on your decision to purchase our
|new screen editor, "use", for your Operating System.
|The acronym "use" stands for the UniFLEX Screen
|Editor.  You can use the editor in conjunction
|with our text-formatting program, "pr", or as a
|what-you-see-is-what-you-get editor.  We hope that by
|the time you finish editing this letter you will feel
|fairly comfortable with may of the screen-editing
|features of "use".  It really is not hard for to learn
|to use!  We also hope, So-and-So, that it will serve you
|very well in your work from day to day.
|
|     The object of this tutorial is to familiarize an
|inexperienced user with the more commonly used features of
-------------------------------------------------------------------
 Col 6                          Word Tab          Line 13
```

Figure 2-7. Effect of the "sup" command.

Once again, type the "sup" command. This time, scrolling up seven lines causes line 13 to be pushed out of the edit window. The editor, therefore, positions the cursor at column 6 of the line at the top of the window, in this case line 15. Subsequent invocations of the "sup" command continue to scroll through the file. In each case the current line scrolls off the screen when you invoke the command, and the cursor is positioned in column 6 of the top line of the edit window. Thus, once the cursor reaches the top of the edit window, it does not change position relative to the window.

The "sup" command behaves similarly to the "page_up" command when it passes the end of the file: it continues scrolling the text normally as long as the last line of the file has not passed out of the edit window. When the last line does pass through the top of the window, USE places the line immediately following the last line at the top of the edit window. This is easier done than said, so continue typing "sup" and watch what happens when you get to the end of the file. You see that the last time "sup" has any effect it puts line 42 at the top of the window. Line 41 is the last line of the file.

2.7.2 Scrolling Down

Scrolling down means that you are pulling on the bottom part of your imaginary scroll. Thus, the text that appears in your edit window is text nearer to the beginning of the file.

2.7.2.1 Full page

Use the "bottom" command

TAB control-T

to position the cursor at the last line of the file. Now, use the "column" command or the "rtab" command (control-G) to move the cursor to column 55, the column containing the 'T' of "TSC". If you have trouble positioning the cursor, go back and review Sections 2.5 and 2.6.

The syntax for the "page_dwn" command is

control-R

Invoke the "page_dwn" command and watch as it scrolls the text down one screenful. The behavior of "page_dwn" is analogous to that of "page_up". The only difference occurs at the boundary of the edit buffer. When its normal behavior would cause the beginning of the file to scroll below the first line of the edit window, "page_dwn" puts the first line of the edit buffer at the top of the edit window. Subsequent invocations of the command have no effect, as you can see by typing "page_dwn" again.

2.7.2.2 Partial screen

As you have probably guessed by now, USE supports a command which is the counterpart of the "sup" command. This command, "sdwn" (scroll down), moves the text in the same direction as the "page_dwn" command, but it moves only one third of a screenful at a time. In our case it moves the text seven lines at a time.

The "sdwn" and "sup" commands handle placement of the cursor in a similar fashion. If the current line remains on the screen after the execution of the "sdwn" command, the cursor remains on that line of text. Thus, its position with regard to the edit window changes. If,

on the other hand, execution of the "sdwn" command forces the current line off the bottom of the screen, the editor positions the cursor at the line at the bottom of the edit window. When line 1 reaches the top of the edit window, the "sdwn" command has no effect. The "sdwn" command always maintains the horizontal position of the cursor.

Use your newly gained expertise to position line 41 (the last line of the file) at the bottom of the edit window with the cursor in column 56 of line 27. Execute the "sdwn" command, whose syntax is

control-Y

repeatedly until you understand its operation. If you run out of room in the file, reposition the cursor at the bottom of the file with the "bottom" command or by practicing the "page_up" (control-V) and "sup" (control-N) commands.

2.7.3 Remembering the Scrolling Commands

The keys which scroll text upward move you deeper into the file you are editing. In this sense they are akin the the "down" command, and, in fact, they are located on the same row of the keyboard. Similarly, the keys which move you closer to the beginning of the file are on the same row as the "up" command.

2.7.4 Homing a Line

One form of the "sup" command has a different effect on the text of the file. Rather than scrolling the text, the "home_line" command positions the current line at the top of the edit window. This command is one which you will undoubtedly use over and over. Its syntax is

TAB control-N

Position the cursor anywhere you like and type "home_line". Notice that the editor maintains the horizontal position of the cursor. It simply scrolls the text in whichever direction is necessary until the appropriate line reaches the top of the edit window.

2.8 Exiting from the Editor

At this point you might like to take a break, but it is wise to stop editing your file before you do so. Then if anything happens to the system while you are not using it, your file is likely to remain intact. The editor supports several different versions of the "exit" command. We will discuss only one of them here, but you will use two others during the course of the tutorial. Before discussing them, however, it is necessary to explain what a backup file is.

2.8.1 Backup Files

When you issue the "use" command with an existing file as its argument, the editor makes a copy of that file. Any changes you specify during the editing session are made on this copy. Thus, while you are editing, two copies of the file exist. The editor leaves the original one intact and makes changes in the copy.

At the end of an editing session the editor renames the original file by appending the characters ".bak" to its name. If this addition would result in a name that is longer than the operating system allows (fourteen characters for 6809 UniFLEX; fifty-five, for 68xxx), the editor shortens the original name before adding the suffix. The file in which the editor has made changes is then given the name of the original file. You now have two files: <file_name> and <file_name.bak>.

If a file named <file_name.bak> already exists, the editor asks for permission to delete that file. In response to this question you must type either 'y' for "yes" or 'n' for "no". If you type 'y', the editor deletes the file whose name is <file_name.bak>, gives <file_name> the name <file_name.bak>, and names the copy of <file_name>, to which the changes were made, <file_name>. If you type 'n', the editor deletes the original <file_name>, names the edited copy <file_name>, and leaves the file <file_name.bak> untouched.

The editor accepts no characters other than a 'y' or an 'n' at this point. You do not need to type a carriage return after your response.

In any case, when you end an editing session in the usual way, the copy of the file, to which the changes have been made, is given the name of the original file. The contents of the backup file depend on your response. If you give permission to delete the existing backup file, the original file becomes the backup file. If you deny permission to

delete the existing backup file, the editor deletes the original file and leaves the backup file intact.

2.8.2 The "save" Command

Generally, when you leave the editor, you want to save the changes you have made during the editing session. The form of the "exit" command which does that for you is is the "save" command. Its syntax is

TAB [y] control-U

Because the 'y' is enclosed in square brackets in the syntax statement, you know that it is an optional argument, and because it is not enclosed in angle brackets you know that if you use it, you must type it as is (see Section 2.2.2). If you do not use the optional 'y', the editor asks you whether or not you want to delete the existing backup file--if one exists. If you do use the 'y', it tells the editor that you want to respond to its question about the backup file with a "yes". Thus, the 'y' allows you to avoid a question which can become rather tedious.

Because you are editing the sample file for the first time, you do not have to worry about a backup file yet. However, when you exit from the editor, it will create a backup file named "sample.bak".

Now type the "save" command

TAB control-U

The following message flashes on your status line

```
Busy
```

followed by the message

```
Saving file ...
```

while the editor saves the file. Your screen then momentarily goes blank. Control returns to the UniFLEX Operating System, and the system prompt appears in the upper left-hand corner of your screen.

This is a convenient time to take a break. When you are ready to resume work with the editor, simply type

```
use sample
```

2.9 Entering Characters

So far you have learned how to move the cursor to various places in your file, but as yet you have made no changes. The editor supports two ways of entering text into a file. You can either type directly over existing text (including blanks) or insert text between existing characters.

2.9.1 Typing over Existing Text

Use the "line" command to position the cursor at the beginning of line 9 of the sample file. Now type your name. The editor replaces the characters already on the line by the ones that you type. If your name is shorter than the string "Your name" which appears in the original file, remove the extra characters by typing over them with the space character. If it is longer, continue to type over the blank spaces until you have typed your whole name or used seventy-eight columns, whichever happens first! When you have finished typing your name, type the "return" command. Enter your street address in line 10 and your city, state, and zip code (or whatever information is necessary to complete your address) in line 11. If you make a typographical error anywhere, reposition the cursor so that it is at the character in error. Retype that character and as much of the rest of the line as necessary to correct the problem. This method is not necessarily the most efficient way to correct the mistake, but it does work.

2.9.1.1 The "backspace" command

You will now use the "backspace" command to edit line 25 of the sample file. Most keyboards have a backspace key, but if yours does not, you can generate the command by typing control-H. You will change the text from

```
    very well in your work from day to day.
```

to

very well in your work.

First of all, use the "line" command to position the cursor at the beginning of line 25. Next, use the "app" command (control-]) to move it to the end of the line. The editor positions the cursor at the column following the period. Now use the "backspace" command to move the cursor until it is in column 23, the column immediately following the word "work". Each time you type "backspace", USE deletes the character immediately to the left of the cursor and moves the cursor one column to the left. When you reach column 23, type a period, '.', to complete the change.

2.9.1.2 "backspace" versus "left"

Be sure that you understand the difference between the "backspace" and the "left" commands. The "backspace" command is a destructive command; that is, it deletes text. The "left" command, on the other hand, simply moves the cursor over the text without deleting it.

2.9.2 Inserting Characters

In its normal mode of operation, the editor replaces existing text with new text as you type over it. The "char_insert" command provides a nondestructive way of inserting text. It allows you to insert new characters between existing ones without destroying any of the previously existing text.

The syntax for the "char_insert" command is

control-P

This command, like some others you have already learned, toggles between two functions. If you are in the normal mode of operation (where typing over a character deletes it), "char_insert" puts the editor in character-insert mode. As it does so, the message

Char Ins

appears in the status line. If the editor is already in character-insert mode when you type control-P, the "char_insert" command

returns the editor to the default mode, removing the message in the status line as it does so.

You will now use "char_insert" to correct an error on line 30. In order for the sentence to read correctly, you must insert the word "the" at the beginning of the line. Of course, you could retype the entire line, but using "char_insert" is much more efficient.

Use the "line" command to position the cursor at the beginning of line 30. Now type control-P and watch for the message informing you that the editor is in character-insert mode to appear in the status line. When it does, watch the screen as you type the string "the " to correct the text (be sure to include a space character). When you are in character-insert mode, any character you type is placed in the column the cursor was in. At the same time, the editor shifts the cursor and all the text to the right of it on the current line one column to the right.

2.9.2.1 The "backspace" command again

The "backspace" command behaves differently when the editor is in character-insert mode than when it is in the default mode. To illustrate this difference, let's correct the error on line 32. The reference guide referred to in that line is actually the third chapter of this manual. You need, therefore, to change the word "second" to "third". As is often the case when you are editing a document, you could make this change in several different ways.

Use the "down" command to position the cursor to line 32. Then use the "column" command to position it at column 49, the column containing the space character immediately following the word "second". Now type the "backspace" command (control-H). The character immediately to the left of the cursor disappears. The editor also shifts the text from the cursor to the end of the current line one column to the left. Continue to "backspace" until you have obliterated the entire word. Then simply type in the word "third". When the line is correct, invoke the "char_insert" command to remove the editor from character-insert mode.

Many people prefer to work with the editor in character-insert mode most of the time. Once you have finished the tutorial, feel free to do so if you wish. The tutorial, however, is written assuming that you are not normally in character-insert mode.

2.10 The Configuration File

Several of the commands supported by the editor (such as "word_tab" and "char_insert") affect its behavior until you explicitly revoke the command. They are said to affect one of the modes of operation of the editor. Modes govern many aspects of entering text into a file. Each mode has a default, that is, a standard form of behavior that the editor follows unless otherwise instructed (see Section 3.7). The editor always sends a message to the status line when one of its default modes of operation is no longer in effect.

The "configuration" of the editor is a description of, among other things, which modes are in effect. Unless otherwise instructed, the editor begins with all default modes in effect. For instance, as you may have noticed when you reinvoked the editor, it does not normally begin an editing session in word-tab mode. Because you will be needing word-tab mode throughout the tutorial, you should once again invoke the "word_tab" command.

As you become more familiar with the editor, you will probably find that you prefer to operate it in a particular configuration. Depending on how much that configuration differs from the default, it can become quite tedious to set the correct configuration each time you invoke the editor. Fortunately, the editor supports a command called "config", which writes a description of the configuration of the editor to a file named ".useconfigure" in your working directory. The syntax for the command is

TAB ! control-U

You should invoke this command now so that the next time you exit from and reinvoke the editor you will not have to invoke the "word-tab" command. The editor has no way of confirming to you that it created the configuration file. In the absence of an error message, you must assume that the command succeeded. (For a more complete discussion of the configuration file, see the section on the "config" command in Chapter 3.)

2.11 Inserting a String

At times you will want to insert a string in the text of a file rather than a single character. The syntax for the "paste_word" command, which allows you to do so, is

TAB [<string>] control-P

Let's take a close look at this syntax statement. It tells you that the simplest form of the command is

TAB control-P

However, if you want to, you can include the optional argument, which is enclosed in square brackets. Because the argument is also enclosed in angle brackets, you know that the word in the syntax statement is a description of the kind of argument you must use (a string of characters).

When you specify the optional argument <string>, the editor stores that string in a special place called the word buffer. This particular string remains in the buffer until you explicitly replace it or exit from the editor. If you do not specify an argument, the "paste_word" command uses the string that is already in the word buffer. If the word buffer is empty, the message

No string in buffer

appears in the status line.

We will illustrate the use of the "paste_word" command by having you insert the string "UniFLEX " in two places in the sample file. You will change the words "Operating System" (on line 16) to read "UniFLEX Operating System" and the words "Screen Editor" (on line 34) to read "UniFLEX Screen Editor".

Use the "line" command and either the "column" command or one of the "tab" commands to position the cursor at column 36 of line 16, that is, at the 'O' of "Operating System". Type the tab key; the prompt "Arg: " appears in the status line. The argument should be the string you want to insert, so type "UniFLEX " (without the quotation marks). Be sure to include the space character so that the word "UniFLEX" does not run into the following word when you insert it into the file. To actually insert the string, type control-P. The editor now inserts the string you specified into the line at the position of the cursor. It also shifts the cursor and all text from the cursor to the end of the current line far enough to the right to accommodate the insertion.

You now have a string stored in the word buffer. All you have to do to insert the same string again is to position the cursor at the point of insertion (column 23 of line 34) and type

TAB control-P

Try it. Note that although the "paste_word" command inserts a string, it does not leave you in character-insert mode as does the "char_insert" command.

2.12 Deleting Characters

Now you know how to insert characters and strings in the text of a file. It is equally important to learn how to delete them. The sample file contains three words which you will delete. The words "for" (in line 23) and "very" (in line 25) are unnecessary; the word "chapter" (at the end of line 32) is superfluous because it is repeated at the beginning of the next line. We will use the three cases to illustrate three different methods of deleting text.

2.12.1 The "consume_char" Command

Use the "line" command to position the cursor at the beginning of the word "very" (column 1 of line 25). You will use the "consume_char" command to delete this word. The syntax for this command is

control-J

Try it. Each time you type "consume_char", the editor deletes the character in the current column and moves all text to the right of the cursor one column to the left. The cursor remains stationary. Now use the "consume_char" command to delete the entire word and the space character following it. The line should now read

well in your work.

2.12.2 The "consume_line" Command

A similar command, the "consume_line" command, allows you to delete, in one step, the character in the current column and all characters in the current line to the right of the current column. The cursor does not move. The syntax for the "consume_line" command is

TAB control-J

Position the cursor at column 49 of line 32:

The reference guide (which is in the third chapter

Type the "consume_line" command and watch the text at and to the right of the cursor disappear.

2.12.3 The "consume_word" Command

Yet another command, "consume_word", allows you to delete the word in which the cursor is positioned. It also shifts all text to the right of the cursor on the current line far enough to the left to compensate for the deletion, and deletes any trailing blanks (space characters between the last printed character in the line and the carriage return). The cursor may be positioned at any character in the word.

The syntax for this command is

TAB w control-J

Position the cursor anywhere in the word "for" on line 23. Now type the "consume_word" command. Notice that not only does the command delete the word "for" but also it deletes the space character after the word so that no gap exists between the words "hard" and "to". The algorithm for this command is discussed in detail in the documentation of the "consume_word" command in Chapter 3; for now it suffices to note that the command is designed to leave your text without any excessive gaps between words or between a word and a punctuation mark.

2.12.3.1 Definition of a word

Unless you tell the editor otherwise, it assumes that a word consists entirely of alphanumeric characters (the letters of the alphabet plus the digits '0' through '9'). For example, position the cursor at the hyphen in the word "screen-editing" in line 22. Now when you invoke the "consume_word" command, the editor sends the following message to the status line:

Cursor not in word

The problem is that the editor does not recognize a hyphen as part of the word.

2.12.3.2 Changing the definition of a word

The editor supports a command, the "word_def" command, that allows you to add to the set of characters that the editor recognizes as part of a word. The syntax for the command is

TAB wd<char_list> control-A

where <char_list> is a list of the characters you wish to include in the definition of a word.

Let's add the hyphen to this list. You can do so by typing

TAB wd- control-A

Now invoke the "consume_word" command again. This time the word "screen-editing" vanishes from the screen.

2.13 Inserting a Line of Text

You may have noticed that the letter in the sample file is not dated. The proper place for the date is below the zip code in the return address. There is a blank line between the return address and the business heading, but you can preserve that blank line by inserting another one. The "insert_line" command inserts a blank line between the current line and the line preceding it. The syntax for this command is

control-W

Position the cursor to column 52 of line 8 (just beneath the first digit of TSC's zip code) and type "insert_line" (control-W). The file now contains two blank lines between the return address and the business heading. The cursor is in position for you to enter the date, so type today's date in whatever format you prefer.

Because you inserted a line, the file now contains 42 rather than 41 lines. Verify this fact by invoking the "bottom" command. Remember that the status line shows you the number of the current line.

2.14 Deleting Lines

The editor supports two commands for deleting lines. The first, "delete_bl", deletes a line only if it is blank. In addition, it deletes all blank lines between the current line and the next line that contains any text. The syntax for the command is

control-B

To see how it works, position the cursor at the beginning of line 39. Type "delete_bl" (control-B). The three blank lines between the closing of the letter and the typed signature disappear.

Use the "insert_line" command (control-W) to reinsert the three blank lines. Now position the cursor on the third blank line (line 41). Once again, type "delete_bl". This time, because no other blank lines exist between the cursor and the signature, the command deletes only line 41.

Leave the letter as it is (with two blank lines) and position the cursor anywhere on line 29. You are going to use the other form of the "delete" command to remove this line, which is not blank. First of all, try using the "delete_bl" command and note that nothing happens. Now try the "delete_line" command, whose syntax is as follows:

TAB control-B

The line in question vanishes. Notice that the horizontal position of the cursor neither affects nor is affected by either form of the "delete" command.

2.15 Searching for a String

When you are editing a file, it is often convenient to be able to search forwards or backwards through the file for the next occurrence of a given string. Suppose, for example, that you wanted to see every

occurrence of the string "use" in the sample file. You can do so with the two search commands: "search_fwd" searches forwards; "search_bkw", backwards.

These two commands share a buffer called the search buffer. You can put a new entry into the search buffer each time you invoke one of the commands, or you can use whatever string is already in the search buffer. If you invoke the "search_fwd" or "search_bkw" command when the buffer is empty, the editor sends the following message to the status line:

Nothing to search for

2.15.1 Searching Forwards

Use the "top" command to position the cursor at the first column of the first line of the sample file. We will begin by searching forwards for the string "use" (including the quotation marks). The syntax for this command is

[TAB <string>] control-Z

Consider this syntax statement for a moment. It tells you that the essential part of the command is

control-Z

You may also include an optional part consisting of the tab key followed by a string. If you are having trouble understanding the syntax statements refer to Section 2.2.2 or to Appendix A.

If you specify a string, the "search_fwd" command places it in the search buffer. Otherwise, it searches for the string that is currently in the search buffer. The string in the search buffer is called the search string.

Now type the tab key. When the prompt for an argument appears, type "use" (with the quotation marks). Then type control-Z. The cursor moves to the first character of the first occurrence of the search string, which is in line 17.

Once you have entered a string in the search buffer, it stays there until you change it. Thus, to find the next occurrence of "use", you need only type control-Z. Because the line containing the next occurrence of the search string is already displayed on the screen, the editor simply moves the cursor to the first character of the search string. However, when you invoke the command a third time, the line containing the next occurrence of the search string is not displayed on the screen. In such a case the editor positions the cursor at the first character of the search string and places the line containing it approximately one third of the way down the screen (in our case, in the eighth line of the edit window).

Continue typing "search_fwd" until you see the message

    Search key not found

in the status line. This message tells you that the cursor is already at the last occurrence of the specified string in the edit buffer.

2.15.2 Searching Backwards

You can use the "search_bkw" command to search backwards through the file for the string in the search buffer. The syntax for the "search_bkw" command is similar to that for the "search_fwd" command:

[TAB <string>] control-Q

Because the "search_bkw" command shares the search buffer with the "search_fwd" command, you have only to type control-Q to find the previous occurrence of the string already in the search buffer. If you want to change the search string, you must use the long form of the command.

2.16 Replacing a String

The search commands are often used in conjunction with the "replace" command, which is similar in syntax:

[TAB <string>] control-X

If you specify a string, the "replace" command places it in a buffer called the word buffer, which it shares with the "paste_word" command (see Section 2.11). Otherwise, it uses the string that is currently in the word buffer. If you invoke the "replace" command when the word buffer is empty, the editor sends the following message to the status line:

No string in buffer

Now let's replace every occurrence of the string "So-and-So" with your first name. We will move from the beginning of the file forwards, so first use the "top" command to position the cursor at the beginning of the file. Type

TAB So-and-So control-Z

to find the first occurrence of the desired string (column 6, line 14). Once the cursor is in position, type the tab key. In response to the prompt for an argument, type

<your_first_name>

followed immediately by control-X. Be sure you type your own name, not the string "your_first_name". Be careful not to type a space character after your name or it will become part of the string in the word buffer, resulting in an extra space character everywhere the editor makes the change.

After you type control-X, the editor tests to see whether or not the cursor is positioned at the first character of the search string. Because it is, it replaces that occurrence of the string with the string in the word buffer.

To replace the next occurrence of "So-and-So" with your first name, you need only type control-X again. When you do so, the editor tests to see whether or not the cursor is positioned at the first character of the search string. Because it is not, it searches for the next occurrence of the search string ("So-and-So") and replaces it with the string in the word buffer (your first name). The editor positions the cursor at the column immediately following the last character of the string from the word buffer. If you prefer to look at the next occurrence of the search string before changing it, you can type control-Z (or control-Q) before control-X. In either case if the editor cannot find another occurrence of the search string, it tells you so.

The replacement string can only replace the string stored in the search buffer. If it cannot find the search string, it returns the message

Search key not found

Practice this procedure by replacing every occurrence of the string "use" in the sample file with the capitalized version, USE, which does not need to be in quotation marks. If you have any trouble, review this section.

Because the "replace" command and the "paste_word" command share the word buffer, the use of either command leaves a string available to the other.

2.17 Joining Two Lines of Text

The editor supports a command, "join", which allows you to append the following line to the current line. In order to use this command, you must first position the cursor at the last column of the line (or beyond). If you try to use the command without doing so, the editor sends the following message to your status line:

Clear to end of line first

You may clear to the end of the line either by moving the cursor to the end of the line ("app") or by using the "consume_line" command

TAB control-J

to remove the rest of the line.

We will illustrate the use of this command by joining the third paragraph of the letter to the preceding paragraph. To do so, first position the cursor at line 31 (the blank line between the second and third paragraphs) and use the "delete_bl" command (control-B) to delete it. Then use the "up" command, followed by the "app" command, to position the cursor at the end of the second paragraph:

the editor._

You are now ready to join the lines. The syntax for the "join" command is

TAB control-K

Use the command to join the two paragraphs. When you are done, line 30 should look like this:

the editor._ The reference guide (which is in the third

The cursor does not move when the text is joined. The extra spaces are due to the original indentation of the line. Use the "consume_char" command (control-J) to remove three of them, leaving only two space characters between the two sentences.

2.18 Splitting a Line of Text

In some cases it is necessary to split a line of text into two or more lines. You may need to do so to isolate a section of text so that you can copy or move it to another part of the file.

In the next section you will need to move a segment of text from the beginning of the sample file to the end. Therefore, in this section you will use the "split" command to isolate that text so that you can move it later.

The text we want to isolate consists of the fourth and fifth sentences of the letter: "We hope that by the time you finish editing this letter you will feel fairly comfortable with many of the features of USE. It really is not hard to learn to use!"

The syntax for the "split" command is

TAB control-W

This command splits a line of text into two lines by inserting a blank line between the current line and the line following it and moving all text at and to the right of the cursor to the newly inserted line. The cursor does not move.

For example, use the "line" and "column" commands to position the cursor at the beginning of the fourth sentence of the letter (column 39, line 21). Line 13 should be at the top of the edit window, and the current line should read as follows:

what-you-see-is-what-you-get editor. We hope that by

Once the cursor is in position, type "split" (TAB control-W). The editor inserts a blank line below the current line and moves all the text at and to the right of the cursor to that newly inserted line so that the section of the file now looks like this:

what-you-see-is-what-you-get editor. _
We hope that by

Now you need to isolate the text at the other end. Use the "down" command, followed by the "ltab" command (control-A) to position the cursor at the first character of the sixth sentence

to use! We also hope, <your_name>, that it will serve you

Once again type "split". The file should now look like Figure 2-8.

```
Dear So-and-So:
|
|     Congratulations on your decision to purchase our
|new screen editor, "use", for your Operating System.
|The acronym "use" stands for the UniFLEX Screen
|Editor.  You can use the editor in conjunction
|with our text-formatting program, "pr", or as a
|what-you-see-is-what-you-get editor.
|We hope that by
|the time you finish editing this letter you will feel
|fairly comfortable with many of the
|features of USE.  It really is not hard to learn
|to use!  _
|We also hope, <your_name>, that it will serve you
|well in your work.
|
|     The object of this tutorial is to familiarize an
|inexperienced user with the more commonly used features of
|the editor.  The reference guide (which is in the third
|chapter of the manual) contains more detailed
|information about the UniFLEX Screen Editor and the commands
-------------------------------------------------------------------------
  Col 10                          Word Tab          Line 26
```

Figure 2-8. Isolating five lines of text.

The text that you want to move has been isolated on lines 22-26.

2.19 Rewinding a File

Before we continue with our editing project, you will "rewind" your file in order to preserve the changes you have made so far. Frequently rewinding a file while you are working on it is the best way to avoid the loss of a lot of work in the event of a system crash. The "rewind" command allows you to save the current version of your file and to begin a new editing session with that version. Essentially, it is the same as exiting from the editing session and reinvoking the editor. The two crucial differences are that the editor remains configured (see Section 3.7) as it is when you invoke the "rewind" command and that most of the buffers the editor uses (see Section 3.8) remain intact.

The syntax for the "rewind" command is

TAB [y] < control-U

As with the "save" command, the 'y' may be used to tell the editor to save an existing backup file before it has a chance to ask you. The "less than" symbol, '<', followed immediately by the control-U reinvokes the editor with the same file name as you used at the beginning of the session. Thus, if you have made changes to a file named <file_name>, the editor deletes any existing backup file (assuming you give permission), renames the original version of the file <file_name.bak>, and names the newer version <file_name>. It then prepares the file <file_name> for editing and starts another editing session.

It is hard to overemphasize the importance of executing this command at intervals while you work, especially in circumstances where the power supply is unreliable. So, to be safe, type

TAB y < control-U

After you do so, you may catch a glimpse of the message

Saving file ...

before the screen goes blank. Then, rather than returning control to the operating system, the editor starts a new editing session.

2.20 Moving Text

One of the most powerful things the editor allows you to do is to move text around in a file. It mimics the cut-and-paste procedure used when editing by hand by writing text to a special buffer called the line buffer, deleting it from the original place in the text, and reinserting it on demand. You must tell the editor what text to place in the line buffer and where to reinsert it. The editor automatically deletes the original occurrence of the text being moved. You will learn how to move the text one line at a time at first. Then you will start over and move it all at once.

2.20.1 One Line at a Time

Now use the "line" command to position the cursor at line 22 (line 15 should be at the top of the edit window. The syntax for the simplest form of the "cut_line" command (which moves only one line of text) is

control-K

Type control-K. As you do so, line 22 disappears from the screen and the remaining text moves up to compensate for the deletion. The text that was in line 22 is now in the line buffer. You cannot look at the contents of the line buffer without entering it into the file. (However, if you forget what is in there, you can always write it into the file and delete it.)

You are going to move the text of the five lines you isolated in Section 2.18 to form a third paragraph at the end of the letter. First, use the "line" command to position the cursor at line 35, the line immediately following the end of the second paragraph. The cursor may be anywhere on that line; its horizontal position has no effect on the command.

The syntax for the "paste_line" command, which recovers text from the line buffer, is

control-O

Type this command. As you do so, the editor inserts the contents of the line buffer between the current line and the line preceding it. It then moves the cursor to the newly inserted line of text without altering its horizontal position.

Using the "paste_line" command does not remove the text from the line buffer. If you type "paste_line" again, another copy of the line appears in the file. (If you do so, use the "delete_line" command:

TAB control-B

to remove it.)

Now move the cursor back to line 22, which now contains the second line of the text you want to move. Again, type "cut_line" (control-K). Now this line of text is in the line buffer. It has overwritten the previous line. You cannot add text to the line buffer without deleting

any text that was previously in it.

Move the cursor to line 35; type "paste_line" (control-O), and the second line of text appears. Continue this process until you have moved all five lines of text.

We have shown you this method of moving lines because it uses the simplest form of the "cut_line" command. It is, however, admittedly a tedious and inefficient way of moving several lines of text.

2.20.2 Aborting from an Editing Session

We could demonstrate how to move all five lines at once by moving them back to their original place. However, because we want them at the end of the file, we will abort from the editing session and move them again.

The "abort" command allows you to exit from an editing session without making any of the changes you have specified during the session. This command is probably most commonly used immediately after a user accidentally deletes a large part of a file for which no backup file exists. It is a convenient choice at this point because the only change you have made in the file since you invoked the editor with the "rewind" command is the moving of the five lines. If you abort from the editing session, the file will remain as it was at the beginning of this editing session.

The syntax for the "abort" command is

TAB q control-U

When you type this command, control returns to the operating system. Take a break if you want to. When you are ready to begin again, type

use sample

2.21 Moving More than One Line

The "cut_line" command, as well as several other commands, can be extended to encompass more than one line of a file. The syntax for this form of the "cut_line" command is

TAB <range> control-K

where <range> specifies which lines to remove from the file and to place in the line buffer. In this case you will use the cursor to define the range of the command. Such an argument is called a cursor-defined argument.

You will now use this technique to move simultaneously the lines which you moved one at a time in the last section. Use the "line" command to position the cursor at line 22 and type the tab key. In response to the prompt for an argument, use the "down" command to move the cursor to line 26. As you do so you should note several things. First of all, the message

Cursor defined

appears in the status line, replacing the information that was there. This message indicates that the scope of the command is being defined by the position of the cursor. As soon as you move the cursor from its original position, the editor places an "at sign", '@', in its place. This character marks the original position of the cursor while you move it about the file. The editor replaces the "at sign" with the appropriate character when you complete the command.

The range over which the command is to have affect is bounded by the line containing the '@' character and the line containing the cursor. Thus, with the '@' character at line 22 and the cursor at line 26, you are telling the cursor to move lines 22 through 26, inclusive. Your screen should look like Figure 2-9.

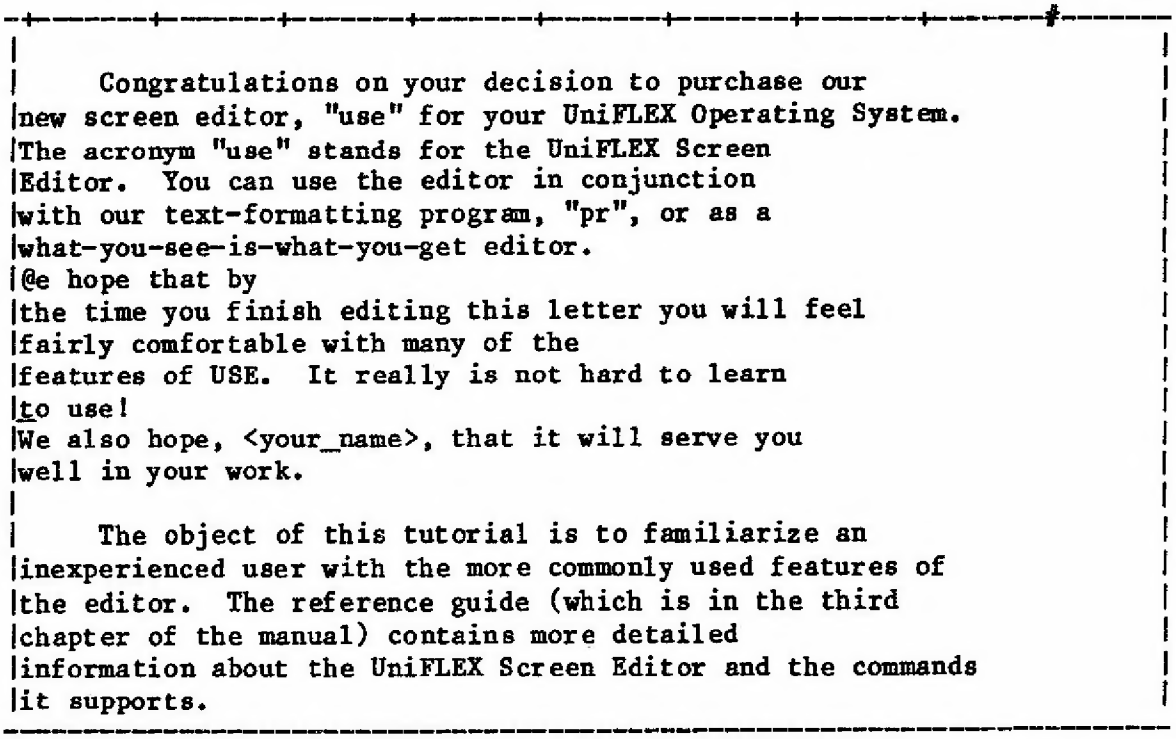

Figure 2-9. Ready to move five lines of text.

All that remains for you to put all five lines into the line buffer is to type control-K. As you do so, the five lines in question disappear from the screen.

The procedure for retrieving multiple lines from the line buffer is the same as that for retrieving a single line. Position the cursor at line 31 and type "paste_line" (control-O). The editor inserts the five lines between the current line and the line preceding it. It then positions the cursor at the first line of the inserted text (without altering its horizontal position).

2.22 Miscellaneous Adjustments

You need to make a few additional changes in the text to adjust it to accommodate the change in position of these five lines of text. First of all, it no longer makes sense to say "We also hope" in the first paragraph. Use the "consume_word" command

TAB w control-J

to remove the string "also ". The command will remove the space character so that deletion of the word does not leave you with two spaces in a row.

In addition, position the cursor at line 31 and use the "insert_line" command to insert a blank line between lines 30 and 31 to separate the two paragraphs. Finally, use the "char_insert" command (control-P) to put the editor in character-insert mode. Insert five space characters at the beginning of line 32 to obtain the proper indentation. Then reinvoke the "char_insert" command to take the editor out of character-insert mode.

2.23 Copying Text

The "copy_line" command is analogous to the "cut_line" command except for the fact that it is nondestructive--that is, it does not remove the text being copied from its original place in the file. The syntax for this command is

[TAB <range>] control-L

The "copy_line" and "cut_line" commands share the line buffer. In both cases text is retrieved from the line buffer with the "paste_line" command (control-O).

For the purposes of illustration we will copy the entire return address and place it under the signature. Just as you could move the lines one at a time, you can copy them one at a time. However, we will proceed directly with simultaneously copying more than one line.

Use the "line" command to position the cursor at the first column of line 5. (The horizontal position of the cursor has no effect on the "copy_line" command.) Type the tab key, and when the prompt for an argument appears, use the "down" command to move the cursor to line 8, the line containing the date. To copy the lines to the line buffer type control-L. Now move the cursor to the line below the last line of the file ("bottom", "down") and release the text from the buffer by typing "paste_line". Next, invoke the "sup" command so that you can see all the results of your effort.

2.24 The "shift left" Command

This unnecessary bit of copying points out that the return address is not aligned with the closing and the signature. If you look closely, you see that the closing and signature are three columns farther to the right than the return address. You could fix this discrepancy by using the "consume_char" command to remove three blank characters from both lines 38 and 41, but it is quicker to use the "shift_left" command.

The "shift_left" command is another example of a command which uses a cursor-defined argument to specify the range of its operation. It is a member of the "delete" family of commands, which are all based on the control-B character. The "shift_left" command deletes characters to the left of the text in question. The syntax for the command is

TAB <block> control-B

where <block> is a block of text described by a cursor-defined argument.

Use the line command to position the cursor at line 38 and the "column" command to move it to column 52. Type the tab key. In response to the prompt for an argument you will use the cursor to describe a rectangle of blank characters for the editor to delete (they need not be blanks for the command to function). Type "down" until the cursor is in line 41, the line containing the signature, "TSC". Type "right" (control-F) until the cursor is in column 55, at the 'T' of "TSC". Your file should look like Figure 2-10. If it does not, you can use the cursor-positioning keys to move the cursor so that at least the part of the file being manipulated does.

```
-+-------+-------+-------+-------+-------+-------+-------+-------+------
|                                                                       |
|     We hope that by                                                   |
|the time you finish editing this letter you will feel                  |
|fairly comfortable with many of the features                           |
|of USE.  It really is not hard to learn                                |
|to use!                                                                |
|                                                                       |
|                                                     @incerely yours,  |
|                                                                       |
|                                                                       |
|                                                        TSC            |
|                                                     111 Providence St.|
|                                                     Chapel Hill, NC   |
|                                                     27514             |
|                                                     <date>            |
|                                                                       |
|                                                                       |
|                                                                       |
|                                                                       |
|                                                                       |
|                                                                       |
------------------------------------------------------------------------
 Arg: Cursor defined
```

Figure 2-10. Preparation for the "shift_left" command.

Now type control-B. The text in lines 38-41 shifts three columns to the left so that it is now aligned with the return address.

Notice that the vertical boundaries of the area to be affected extend from the line the cursor was originally on to the line it is on when you type control-B, inclusive. The horizontal boundaries, however, extend from the column the cursor was originally in to the column just before its final position.

2.25 Deleting More than One Line

The lines that you copied to the end of the file do not need to be duplicated in the letter. We moved them solely for the purpose of illustrating the "copy_line" command. Now they will provide the opportunity for you to learn how to delete more than one line of text at a time.

Use the "line" and "home_line" (TAB control-N) commands to put line 30 at the top of the edit window. Use the "down" command to move the cursor to line 42, the first line of the duplicated return address. You will delete lines 42 through 45, inclusive. The command which allows you to do so is a variation of the "delete_line" command that takes a cursor-defined argument. Its syntax is as follows:

TAB [<range>] control-B

Type the tab key, and in response to the prompt, use the "down" command to position the cursor to line 45, the line containing the copy of the date. Once again, the command will affect the lines from the line containing the '@' character to the line containing the cursor, inclusive. When you have defined this range, type control-B. The five lines disappear from the file.

2.26 Aligning the Text

As you have probably noticed by now, as a result of all the changes you have made in the file, the right-hand margin of the letter is rather erratic. The editor supports a command, "align", which allows you to force the text to fit neatly within the bell column. The command operates on the text between the current line and the end of the current paragraph. A paragraph is defined as the series of lines up to but not including the next line containing a space character in the first column, which the editor assumes is intentional indentation.

Position the cursor anywhere on line 16, the first line of the first paragraph, and invoke the "align" command:

TAB j control-G

The word "Busy" appears briefly on the status line as it does whenever the editor is involved in one of its more time-consuming tasks. When the command is complete, the first paragraph will have a neat right-hand margin.

Now move the cursor to the first line of the second paragraph and repeat the command. Finally, use it to align the text of the final paragraph.

2.27 The "shift_right" Command

Now that you have aligned the return address with the closing of the letter and tidied up the right-hand margin, you may have noticed that the text of the body of the letter does not extend as far to the right as do the closing and the return address. You could use the "shift_left" command to move the closing and return address several columns to the left to bring everything into alignment. However, another solution, which uses a command that has not yet been introduced, is to move the entire body of the letter to the right, thus adding a left margin (which might be desirable if you decided to print the letter).

The "shift_right" command is analogous to "shift_left". However, rather than deleting some text (or blank characters) to shift other text to the left, "shift_right" inserts blank characters to move text to the right. The syntax for the command is

TAB <block> control-W

where <block> once again is a section of text described by a cursor-defined argument. We will illustrate its use by moving the body and header of the letter three columns to the right.

Use the "line" command to position the cursor at column 1, line 10, the first line of the header. Then use the "home_line" command (TAB control-N) to position this line at the top of the edit window. Type the tab key. When the prompt appears, use the "right" command to move the cursor to column 4. Use the "down" command to move the cursor to line 30, which should be the last line visible in the edit window. The range of any command using a cursor-defined argument is limited to the lines in the edit window. You have, therefore, defined the extent of the text which you want the editor to shift. Type control-W to actually shift it. Repeat the procedure to shift the remainder of the body of the letter four columns to the right. If you need to use the "shift_right" command for only one line of text, you invoke it similarly, but do not need to use the "down" command as part of the cursor-defined argument.

2.28 On Your Own

You may find other changes you would like to make to the letter to suit yourself. At this point you should feel free to experiment with the editor. The only way to really become familiar with the various commands is to practice them many times.

Appendix B contains a brief description of every command supported by the editor. As you can see from reading the list, this tutorial has introduced you to some of the power available to you through USE, but it by no means covers all the features of the editor. When you feel you have mastered these skills or when you want to learn what else USE can do, refer to the reference section of this manual (Chapter 3).

Chapter 3

Reference Guide to the UniFLEX Screen Editor

3.1 Introduction

This reference guide is a comprehensive description of the features of the UniFLEX Screen Editor (USE). It more fully describes the features discussed in the tutorial and introduces others. Following the general discussion of the editor are detailed descriptions (in alphabetic order) of all the commands it supports.

3.2 Invoking the Screen Editor

The syntax for invoking the screen editor is as follows:

```
use <file_name> [+b]
use <old_file_name> <new_file_name>
```

where <file_name> may be the name of an existing file or of a new file, <old_file_name> must be the name of an existing file, and <new_file_name> must be the name of a new file. Unless otherwise specified, the editor assumes that all files are in the working directory.

When a user invokes the editor with an existing file as its argument, it first creates a temporary file named "edit.<num>" where <num> is the identification number of the task (task ID). This file may be necessary during the editing session if the file being edited is a large one. After creating the temporary file the editor copies as much as possible of the file to be edited into a buffer in memory called the edit buffer. On 6809 machines the operating system limits the size of the edit buffer to 30 kilobytes. On 68xxx machines the limit on the size of the edit buffer is determined by the size that is specified in the header of the binary file, "/bin/use". If necessary, the system manager can change this limit without much affect on the performance of the operating system. In any case, if the editor cannot fit the entire file into the edit buffer, it sends the following message:

File not completely loaded

to the status line (see Section 3.4).

Any changes specified during the editing session are made on the copy of the file in the edit buffer. Thus, during an editing session two copies of the file exist: one on the disk and one in memory. The editor leaves the one on the disk intact and makes changes in the copy in memory.

If at any point during the editing session the edit buffer is filled to capacity, the editor sends the following message to the user:

Memory buffer full

When this happens, the user cannot enter more text into the file without at least partially emptying the buffer with some form of the "flush" command. The "flush" command writes some or all of the information in the memory buffer to the temporary file that the editor created when it was first invoked and flushes it from the memory buffer.

At the end of an editing session the editor renames the original file by appending the characters ".bak" to its name. If this addition would result in a name that is longer than the operating system allows (fourteen characters for 6809 UniFLEX; 55, for 68xxx), the editor shortens the original name before adding the suffix. It then writes the contents of the memory buffer to the temporary file and gives that file the name of the original file. There are now two files: <file_name> and <file_name.bak>.

If a file named <file_name.bak> already exists, the editor asks for permission to delete that file. In response to this question the user must type either 'y' for "yes" or 'n' for "no". In response to a 'y' the editor deletes the file whose name is <file_name.bak>, gives <file_name> the name <file_name.bak>, and names the copy of <file_name>, which is now in the temporary file, <file_name>. If the user types 'n', the editor deletes the original <file_name>, renames the temporary file <file_name>, and leaves the file <file_name.bak> untouched.

The editor accepts no characters other than a 'y' or an 'n' at this point. No carriage return is necessary after the response.

In any case, when a user ends an editing session in the usual way, the copy of the file, to which the changes have been made, is given the name of the original file. The contents of the backup file depend on the response to the editor's question. If the user gives permission to delete the existing backup file, the original file becomes the backup file. If the user denies permission to delete the existing backup file, the editor deletes the original file and leaves the backup file intact.

The second form of the "use" command accepts as arguments the names of two files. The first file must already exist; the second must not. The files may be in different directories, even on different devices.

This form of the "use" command behaves essentially the same as the first form. The difference is that, rather than creating a temporary file, the editor creates a file named <new_file_name>, which it uses in the same way as it normally uses the temporary file. However, at the end of the editing session, the editor renames neither the original file nor the new file.

3.2.1 Options

The "use" command supports one option, the 'b' option, which tells the editor to automatically delete the original copy of the file at the end of the editing session without creating a backup file. It does not alter any existing backup file.

3.2.2 Examples

The following examples illustrate the various ways of invoking the editor:

1. use sample
2. use sample +b
3. use sample new_sample

The first example opens the file "sample" for editing. If the file already exists, the editor loads as much of the file as possible into memory and displays one screenful of the text. If the file does not exist, the editor creates it and displays a blank screen.

The second example behaves the same as the first except that at the end of the editing session the editor does not query the user about deleting any existing backup file. Rather, it simply deletes the original copy of the file.

The third example creates a file named "new_sample" and uses it as the temporary file. At the end of the editing session the user has two files: "sample", which has not been changed; and "new_sample", which is the edited version of "sample". If a file named "new_sample" exists when the command is invoked, the editor returns an error message.

3.2.3 Error Messages

Under certain conditions a user's attempt to invoke the editor may fail. In such a case the editor returns one of the following error messages and aborts, returning control to the operating system.

Cannot open "<file_name>": Permission denied
  The permissions set for the file do not allow the user to edit it.

Error getting status of "<file_name>": Not a directory
  The path specification for the file to be edited contains a nonexistent directory.

File "<file_name>" already exists.
  If the user invokes the editor with the names of two files, the second name must specify a nonexistent file.

"<file_name>" is a <special_file> and cannot be edited.
  The user tried to invoke the editor for use with a special file such as a directory or a device.

3.3 The Edit Window

The edit window is a rectangle which surrounds the part of the file that is visible to the user. Although the number of lines displayed in the window may vary (it is typically 21), the editor always displays a total of eighty columns (including one column for the left-hand boundary of the window and one for the right-hand boundary). If the terminal cannot accept eighty characters in one line, the results are unpredictable.

The space located within the borders of an edit window is called the data space. The data space contains the cursor, which indicates where on the screen the next character entered directly from the keyboard will appear. The shape of the cursor varies from terminal to terminal. The line containing the cursor is referred to as the current line; the column containing the cursor, as the current column.

3.3.1 Primary and Secondary Windows

The editor allows 128 characters in a line of text. If the user is working in the first seventy-eight columns of a line, the text is surrounded by a "primary" window. Once the user crosses column 78, the display changes. A "secondary" window (displaying columns 50-128) now surrounds the text. These two types of window are easily distinguished by the characters which make up their margins (see Section 3.3.3).

3.3.2 Behavior in Column 128

The right-hand margin of a secondary edit window coincides with column 128. The only character which a user can enter into the file in column 128 is a carriage return. A user who tries to enter any other character is greeted with the following message:

```
Not enough room on line
```

The "rtab" command has no affect if the user executes it when the cursor is in column 128 (which always contains a tab stop). The "right" command, however, moves the cursor from column 128 of one line to column 1 of the next.

3.3.3 Window Borders

The editor uses the following symbols to create the borders of an edit window:

| The vertical bar is the default character used for the right- and left-hand margins of the window when no other character is appropriate. It is also used in the top margin to indicate the position of the bell column when that column does not also contain a tab stop.

- The minus sign is used to form the entire bottom margin of all edit windows. It is also the default character used in the top margin when no other character is appropriate.

< The "less than" symbol is used to form the entire left-hand margin of a secondary window. Thus, the user can tell at a glance that more text exists to the left of the margin (theoretically the text could be all blanks).

> The editor uses the "greater than" symbol as the right-hand margin of a line of text in a primary window when that particular line contains text beyond column 78 (even if the text is all blank characters).

+ A plus sign in any column in the top margin of an edit window indicates that that column contains a tab stop and is not the bell column. Columns 1 and 128 always contain a tab stop.

A pound sign appears in the top margin of an edit window at the bell column if that column is also a tab stop.

3.4 The Status Line

The line immediately below the bottom margin of an edit window is called the status line. By default, this line contains the strings "Col" and "Line", each followed by a number. The numbers indicate the position of the cursor in the file. Both lines and columns are numbered starting with 1. The line number is always the number of the line in the file itself, which is not necessarily the same as the number of the line in the edit buffer (see "flush").

The editor also uses the status line to display the contents of various buffers and to indicate when any mode other than one of the default modes is in effect (see Table 3-1).

Table 3-1. Conditions Indicated in Status Line

Indicator	Condition	Reference
AImode	Automatic-indent mode	Section 3.7.5
Arg:	Awaiting argument	Section 3.6.1
Busy	Time-consuming command in progress	
Caps	Capitalize letters	Section 3.7.2
Char Ins	Character-insert mode	Section 3.7.3
Cursor defined	Range of command is defined by cursor	Section 3.6.6.2
Global	Perform command repeatedly	"global"
PPmode	Paragraph mode	Section 3.7.5
RJmode	Right-justify mode	Section 3.7.5
Word Tab	Tab to word, not to tab stop	Section 3.7.4

Finally, the editor uses the status line to send messages and report errors to the user. These messages and errors are discussed in the relevant sections of the manual.

3.5 Control Sequences

The commands supported by the editor are based on "control sequences", that is, they are defined by keys which produce nonprintable characters. Many control characters are not represented by a single key on the keyboard. Rather, the user enters such a character by depressing the key marked "control" (or "ctrl") and holding it down while typing another character. In this document such a control sequence is referred to by the string

control-<char>

Alphabetic control characters are shown with uppercase letters because

that is what is usually on the keyboard. The user may, however, enter either an upper- or lowercase character.

The more commonly used control characters--such as the escape, carriage return, and tab keys--usually do have individual keys on the keyboard. In this document whenever a control sequence is referred to by a name, such as "tab", in a syntax statement or an example, the name appears entirely in uppercase letters. This convention tells the user to type the key corresponding to the name, not the name itself.

A keyboard may also have keys labeled with other functions--such as "home" and "replace". These keys may or may not send the same control sequence as the commands of the same name documented in this manual. The user must experiment with any given terminal or look in the manual that came with it in order to determine what control character a particular key represents.

3.6 The Tab Key

The control sequence for many of the commands supported by the editor include the tab key (control-I). In some cases the tab key simply distinguishes one command from another (for instance, the control sequence for the "top" command is control-T; for the "bottom" command, TAB control-T). However, in many cases the tab key provides the user with an opportunity to modify a command by specifying one or more arguments.

3.6.1 Effect on Status Line

When a user types the tab key, several things happen at once. First of all, an "at" symbol, '@' replaces the cursor. Secondly, the editor clears the status line, replacing whatever text was there with the string "Arg: ". Thirdly, the cursor moves to the status line.

3.6.2 Entering Arguments

Depending on the command, once the prompt for an argument appears in the status line, the user may have the option of entering an argument to modify the command.

Some arguments are alphabetic; others numeric. These forms of arguments are explained, when relevant, with the individual commands. Still others consist of line numbers, which specify a range of lines over which the command is to take effect. Finally, some commands which can act on a range of lines also accept as an argument the movement of the cursor in a specialized way (see Section 3.6.2.2).

3.6.2.1 Line numbers as arguments

The format for an argument that specifies the range of lines over which the command is to take affect is as follows:

#<line_num_1>,<line_num_2>

where <line_num_1> must be less than <line_num_2>. If it is not, the editor returns the message

Invalid parameter

The editor recognizes the symbols shown in Table 3-2 as valid line numbers.

Table 3-2. Symbolic Line Numbers

Symbol	Line Represented
^	First line of edit buffer
!	Last line of edit buffer
.	Current line

3.6.2.2 Cursor-defined Arguments

A number of commands supported by the editor accept so-called cursor-defined arguments. Rather than explaining this type of argument in detail with each command, we will explain it once, here.

All commands which accept cursor-defined arguments have as their basis the following general format:

TAB <block> <control_sequence>

After the user types the tab character, the usual prompt appears in the status line. At this point, rather than typing any printable characters, the user may type any of the cursor-positioning commands except "app" to define a block of text over which the command is to have effect. All these commands (up, down, right, left, rtab, ltab, and home) except "home" affect the cursor in the usual way. In the case of cursor-defined arguments the "home" command causes the cursor to toggle between the line the cursor was on when the user initiated the command and the last line of the edit window. While doing so, it maintains the horizontal position of the cursor.

While the user is manipulating the cursor, the editor displays the message

Cursor defined

in the status line. This message is displayed until the user either returns the cursor to its original position, types the control sequence that executes the command, or aborts the command.

For some commands only the vertical position of the cursor is important; for others, only the horizontal position. For still others, movement of the cursor in either direction affects the scope of the command. In any case, the vertical boundaries of the area to be affected extend from the line the cursor was originally on to the line it is on when the user types the control sequence that executes the command, inclusive. The horizontal boundaries extend from the column the cursor was originally in to the column before its final position.

The first movement of the cursor in a cursor-defined argument must be either down or to the right. After the initial movement, the user may move the cursor up or to the left but may never go above or to the left of the original position of the cursor. A cursor-defined argument may extend for the full width of the line, but it cannot extend beyond the last line in the edit window.

3.6.2.3 Correcting errors

A user who types the tab key in error can avoid entering any argument or command by immediately typing it again. If an argument has already been entered, typing the tab key twice should put the cursor back in the data space where it was when the user initiated the command. Any errors in the typing of an argument can be corrected by backspacing and retyping. The cursor-positioning commands cannot be used to move the cursor within the status line.

3.7 Modes of Operation

Several of the commands supported by the editor affect its behavior until the user revokes the command. Such a command affects one of the editor's "modes of operation". Modes govern the following aspects of entering text into a file: the case of the characters entered into the file; whether a character typed from the keyboard deletes a character already in the same location or shifts the position of that character and all others to the right of it on the same line one column to the right; whether the tab functions move the cursor to a column containing a tab stop or to the beginning of a word; behavior of the editor at the bell column; indentation after the user types a carriage return. When any mode other than a default mode is in effect, the editor so indicates with a message in the status line (see Table 3-3).

This section describes the behavior of the editor in its default modes. Details of the each mode of operation appear with the documentation of the command that invokes that mode.

3.7.1 Column Indicator

By default, the editor displays the string "Col" in the status line, followed by a number indicating the number of the current column. The "column_ind" command allows the user to remove that indicator from the status line or, if the indicator is not there, to return it (see "column_ind").

3.7.2 Case of Characters

By default, the editor enters alphabetic characters in the same case in which the user types them. In capitals-locked mode the editor enters all alphabetic characters as uppercase characters (see "caps").

3.7.3 Entering Characters

By default, the editor replaces existing text with new text as the user types over it. In character-insert mode the user can insert characters between existing ones without destroying any of the previously existing text (see "char_insert").

3.7.4 "Tab" Commands

By default, the "rtab" and "ltab" commands position the cursor at the next or the previous column that contains a tab stop. In word-tab mode these commands position the cursor at the beginning of the next or previous word (see "word_tab").

3.7.5 Margins and Indentation

By default, the editor sends a bell to the terminal when the user types a character in the bell column. The editor does, however, allow the user to continue adding text to the line until the line is full. When the user types a carriage return, the cursor moves to the first column of the next line of text.

When the editor is in paragraph mode, automatic-indent mode, or right-justify mode, it behaves differently at the bell column and when the user types a carriage return (see "auto_ind", "pp", and "rjustify").

3.8 Buffers

The editor maintains several buffers which are shared by various commands. When the user invokes the editor, these buffers are all empty. However, it is possible to preserve the contents of all these buffers except the read-name buffer from one editing session to the next by specifying the name of a file with the "exit" command (see "exit"). Table 3-3 shows which commands access which buffers.

Table 3-3. Buffers Available to the Editor

Name	Contents	Maximum Size	Accessed by
Line	Line(s) of text.	Length of window	copy_line cut_line paste_line
Read-name	Name of a UniFLEX file.	6809: 14 chars. 68xxx: 55 chars.	read_file
Search	Any string of printable characters.	1/2 max line - 6	display read_file search_bkw search_fwd search_rep
Word	Any string of printable characters.	1/2 max line - 6	copy_word cut_word display paste_word replace
Word-def	Characters to include in definition of a word.	39	display word_def
Write-name	Name of a UniFLEX file.	6809: 14 chars. 68xxx: 55 chars.	copy_to_file move_to_file

3.9 Abnormal Termination of the Editor

The editor ignores both keyboard interrupts and quit interrupts. It catches hangup interrupts. When the editor catches a hangup interrupt, it renames the temporary file "hedit.<num>", where <num> is the task ID, and copies the contents of the edit buffer to that file before it terminates.

Any other interrupt terminates the editor immediately, and the contents of the edit buffer are lost.

abort

End the editing session without saving any of the changes made to the file.

SYNTAX

TAB q [<[<file_name>]] control-U

DESCRIPTION

When the editor is invoked with a single argument, it copies the file being edited into memory and creates a temporary file in the same directory as the file being edited. The temporary file is used during the editing session if the user fills the edit buffer. Any changes the user makes during the editing session are made on the copy in memory.

The "abort" command returns control to the operating system without writing any of the changes made during the editing session to the disk. It also deletes the temporary file that was created when the user first invoked the editor. Thus, the original file and the backup file (if one exists) remain unchanged.

Arguments

<file_name> The name of the file with which to begin the next editing session. The default is the name of the file currently being edited. If the user begins an editing session this way, the contents of the word, line, and search buffers remain intact from one session to the next.

EXAMPLES

1. TAB q <control-U
2. TAB q <new_file control-U

The first example exits from the editing session without writing any of the changes made during the session to the disk and deletes the temporary file. It then begins an edit session on the same file as the previous editing session. All changes made during the previous editing session are lost.

The second example exits from the editing session without writing any of the changes made during the session to the disk and deletes the temporary file. It then begins an edit session on the file named "new_file" in the working directory.

(continued)

abort-2

SEE ALSO

exit

align

Force lines to fit within the bell column.

SYNTAX

TAB j control-G

DESCRIPTION

The "align" command forces the text between the current line and the end of the current paragraph, inclusive, to fit within the bell column. A paragraph is defined as the series of lines up to but not including the next line containing a space character in the first column (assumed to be intentional indentation).

In all formatting modes except right-justify mode, alignment is achieved by making each line as long as possible without putting any characters in or beyond the bell column. If the line extends beyond the bell column, "align" splits it at the boundary between two words. If the line is not long enough, it tries to bring one or more words from the next line up to the line it is working on.

When the editor is in right-justify mode, the command behaves similar

align

Force lines to fit within the bell column.

SYNTAX

TAB j control-G

DESCRIPTION

The "align" command forces the text between the current line and the end of the current paragraph, inclusive, to fit within the bell column. A paragraph is defined as the series of lines up to but not including the next line containing a space character in the first column (assumed to be intentional indentation).

In all formatting modes except right-justify mode, alignment is achieved by making each line as long as possible without putting any characters in or beyond the bell column. If the line extends beyond the bell column, "align" splits it at the boundary between two words. If the line is not long enough, it tries to bring one or more words from the next line up to the line it is working on.

When the editor is in right-justify mode, the command behaves similarly, but it also right-justifies each line after making it as long as possible.

The "align" command always condenses multiple space characters to a single space character unless they appear at the beginning of a line (assumed to be intentional indentation).

NOTES

- The user may invoke the "align" command in conjunction with the "global" command to align all the text in the edit buffer or to align all the text between the current line and the end of the edit buffer.

SEE ALSO

global
rjustify

app

Position the cursor at the column following the column containing the last character in the line or at the first column of the line.

SYNTAX

control-]

DESCRIPTION

If the cursor is not in the column following the column containing the last character on the line, the "app" command puts it there. If the cursor is already in that position, "app" positions it at the first column of the line.

It is not always obvious which column contains the last character of a line. When the user enters any printable character (including the space character) in a line, the editor enters a space character in each column between that character and the column which previously contained the last character. If the user erases the last character, the character preceding it--which may be a space character inserted by the editor, not the user--becomes the last character in the line. However, when it saves the file, the editor truncates any trailing space characters (space characters between the last printed character in the line and the carriage return).

auto_ind

Put the editor in automatic-indent mode.

SYNTAX

TAB a RETURN

DESCRIPTION

The "auto_ind" command puts the editor in automatic-indent mode. In this mode of operation the editor treats the right-hand margin of the text the same way that it does in the default mode--it sends a bell to the terminal when the user types a character in the bell column, but it allows the user to continue typing on that line. However, the behavior of the editor when the user types a carriage return depends on the location of the cursor with respect to the last column in the line that contains a character.

It is not always obvious which column contains the last character of a line. When the user enters any printable character (including the space character) in a line, the editor enters a space character in each column between that character and the column which previously contained the last character. If the user erases the last character, the character preceding it--which may be a space character inserted by the editor, not the user--becomes the last character in the line. However, when it saves the file, the editor truncates any trailing space characters (space characters between the last printed character in the line and the carriage return).

If the cursor is not at or beyond the last column in the line that contains a character when the user types a carriage return, the editor positions it at the first column of the next line of text. If, however, the cursor is at or beyond the last column of the line, the editor inserts a blank line just below the current line and moves the cursor to the column in that line that is directly underneath the first nonblank character in the previous line.

If the user executes the "config" command, the editor sets a flag in the configuration file indicating whether or not it is in automatic-indent mode.

This particular mode of operation is most useful for entering source code as it allows the user to easily maintain the indentation scheme of the code.

The user can exit from automatic-indent mode by returning to the default mode with the following command:

(continued)

TAB RETURN

or by entering one of the other available formatting modes.

MESSAGES

AImode
The editor displays this message on the status line when it is in automatic-indent mode.

SEE ALSO

config
pp
rjustify

backspace

Position the cursor at the column preceding the current column and delete any character in the new position by replacing it with a space character.

SYNTAX

control-H

DESCRIPTION

Normally, the "backspace" command positions the cursor at the column preceding the current column and deletes any character in the new position by replacing it with a space character. However, when the editor is in character-insert mode, the "backspace" command behaves differently. It still positions the cursor at the column preceding the current column, deleting any character at the new position. However, instead of inserting a space character, it shifts all the text from the original position of the cursor to the end of the line one column to the left. In either case, if the cursor is in the first column of a line, the command has no affect.

SEE ALSO

char_insert

bottom

Position the cursor at the first column of the last line of the edit buffer.

SYNTAX

TAB control-T

DESCRIPTION

The bottom command positions the cursor at the first column of the last line of the edit buffer and places that line at the bottom of the edit window.

caps

Change the mode of operation of the editor from the default mode to the mode in which each lowercase letter is translated to its uppercase counterpart as the user types it, or change the mode back to the default.

SYNTAX

control-_

DESCRIPTION

If the editor is in the default mode of operation, the "caps" command switches it to capitals-locked mode, in which each lowercase letter is translated to its uppercase counterpart as the user types it. If the editor is already in capitals-locked mode, the command switches it back to the default mode. If the user executes the "config" command, the editor sets a flag in the configuration file indicating whether or not it is in capitals-locked mode.

MESSAGES

Caps
: The editor displays this message on the status line when it is in not in the default mode.

SEE ALSO

config

char_insert

Change the mode of operation of the editor from the default mode to character-insert mode or vice versa.

SYNTAX

control-P

DESCRIPTION

The "char_insert" command switches the mode of operation of the editor from its default mode to character-insert mode or, if the editor is already in character-insert mode, switches it to the default mode. When the editor is in character-insert mode, it displays the message "Char Ins" in the status line. If the user executes the "config" command, the editor sets a flag in the configuration file indicating whether or not it is in character-insert mode.

When the user types a character in default mode, the editor places that character in the current column, deletes any character that is already there, and moves the cursor one column to the right. In character-insert mode the editor still places the character in the current column, but instead of deleting any text, it moves the character at the cursor, the cursor, and all text to the right of the cursor one column to the right. If doing so moves the last character in the line past the last column in the primary window, the character delimiting the right-hand edge of the edit window on that line changes from a vertical bar, '|', to a right-hand angle bracket, '>'. If the line already has a character other than a space or a carriage return in the column 127, the editor does not allow the user to insert any more characters into the line.

ERROR MESSAGES

Not enough room in line
: The user tried to insert a character into a line which already has a character other than a space or a carriage return in column 127. Under these circumstances the user cannot insert any more characters into the line.

SEE ALSO

config

clr_bell

Clear the bell column.

SYNTAX

TAB c control-G

DESCRIPTION

The "clr_bell" command clears the bell column. The cursor need not be in the bell column for the command to succeed. In the absence of a bell column the user must manually enter a carriage return character at the end of every line of text regardless of the formatting mode that is in effect. In addition, in the absence of a bell column the "align" command has no affect.

SEE ALSO

align
auto_ind
pp
rjustify
set_bell

clr_tab

Clear tab stops as specified.

SYNTAX

TAB <range> control-A

DESCRIPTION

The "clr_tab" command allows the user to remove either the tab stop in the current column or all the tab stops except those in the first and last columns, which are impossible to remove.

Normally the editor indicates the absence of a tab stop in a column by putting a minus sign, '-', in that column in the top margin of the edit window. However, if that column is the bell column, it uses a vertical bar, '|', instead.

Arguments

<range> Specifies the columns in which the command is to take effect. The valid arguments are 'c' and '0'. A 'c' clears the tab stop in the current column. A '0' clears all tab stops except those in the first and last columns of the edit window.

EXAMPLES

1. TAB c control-A
2. TAB 0 control-A

The first example clears the tab stop in the current column.

The second example clears all the tab stops except those in the first and last columns of the line.

NOTES

- The "column" command is helpful for positioning the cursor before clearing the tab stop from a particular column.

(continued)

SEE ALSO

column
set_tab

column_ind

Remove or return the column indicator.

SYNTAX

TAB c RETURN

DESCRIPTION

By default, when a user invokes the editor, it places the string "Col" in the status line, followed by a number indicating the number of the current column. The "column_ind" command allows the user to remove that indicator from the status line or, if the indicator is not there, to return it. If the user executes the "config" command, the editor sets a flag in the configuration file indicating whether or not the column indicator is present in the status line.

SEE ALSO

config

column

Position the cursor at the specified column in the current line.

SYNTAX

TAB <column_num> control-F

DESCRIPTION

The "column" command positions the cursor at the specified column in current line.

Arguments

<column_num>	The number of the column at which to position the cursor. The number specified must be a positive integer. If it is greater than 128 (the number of columns in a line), the editor divides the number by 128 and uses the remainder as the argument.

EXAMPLES

1. TAB 24 control-F

This example positions the cursor at column 24 of the current line.

ERROR MESSAGES

Invalid parameter.
The number specified must be a positive integer.

config

Create a configuration file in the working directory.

SYNTAX

TAB ! control-U

DESCRIPTION

The "config" command creates a file in the working directory called ".useconfigure", which describes the way in which the user had the editor configured when the file was created. The process of creating this configuration file is invisible to the user, who must assume that in the absence of an error message the command succeeded.

Each time the editor is invoked, it searches the working directory for a configuration file. The 68xxx versions also search the user's home directory if the working directory does not contain a configuration file. As a last resort both versions search the directory "/lib". If it finds a configuration file, the editor reads it and adopts the configuration described in the file.

The first byte of the configuration file contains a number relating to the version of the editor under which the user created the file. If this number is not what the editor expects, it returns an error message, which is accompanied by a bell (control-G). Unfortunately, if the file being loaded is too large to fit in memory all at once, the editor immediately overwrites the message with the message that informs the user that the file is not completely loaded. In such a case, the only warning the user receives is the bell.

The information contained in the configuration file consists of a series of flags which indicate which of the following modes are in effect: automatic indent, paragraph, right justify, character insert, capitals locked, column indicator, and word tab. It also describes which columns contain tab stops, which column is the bell column, and which characters are in the word-definition buffer.

If the editor does not find a configuration file or if finds an error in the configuration file, it adopts all the default modes of operation (see Section 3.7), sets tab stops every eight columns beginning in column 1, and makes column 65 the bell column.

(continued)

ERROR MESSAGES

Configuration file error

This message appears in the status line under any of the following circumstances: the first byte of the configuration file indicates that it was created by a version of the editor for which the format of the configuration file differs from the current format; the command detects an error reading the configuration file; the configuration file is not as long as it should be.

SEE ALSO

auto_ind
caps
char_insert
column_ind
pp
rjustify
set_bell
set_tab
word_def
word_tab

consume

Delete a character, word, or the remainder of the line, and shift to the left any text remaining to the right of the cursor.

SYNTAX

```
[TAB [w_or_<range>]] control-J
TAB <block> control-J
```

Variations

consume_char	[TAB <range>] control-J
consume_lb	TAB <block> control-J
consume_line	TAB control-J
consume_word	TAB w control-J

DESCRIPTION

Depending on the form used, the "consume" command deletes a character ("consume_char"), a word ("consume_word"), the remainder of the line ("consume_line"), or leading blanks ("consume_lb"). (A leading blank is any space characters that precedes the first printed character in a line.) As it deletes the text, "consume" shifts any text remaining to the right of the cursor far enough to the left to compensate for the deletion.

The "consume_char" command deletes the character in the current column, shifts all text to the right of the cursor one column to the left, and deletes any trailing blanks (space characters between the last printed character in the line and the carriage return).

The "consume_line" command deletes the characters in the current line at and to the right of the cursor. In both these forms of the command the cursor itself is stationary.

The "consume_word" command deletes the entire word in which the cursor is positioned, shifts all text to the right of the cursor far enough to the left to compensate for the deletion, and deletes any trailing blanks. The cursor may be positioned at any character in the word. By default, a word is defined as a string of alphanumeric characters. The user can, however, invoke the "word_def" command to add to the set of characters that the editor recognizes as part of a word. The word to be deleted may not be longer than the width of the screen. When the user invokes this command, the editor first positions the cursor at the beginning of the word (defined by default as the first alphanumeric character to the right of the preceding nonalphanumeric character or, if there is no preceding nonalphanumeric character, as the character in the first column of the line). It then consumes characters until it comes

(continued)

to the end of the word. If a space character follows the word, the editor deletes it; otherwise, it deletes the space character preceding the word, if there is one, unless that space character is also preceded by a space character. Thus, if a space character follows the word, the cursor ends up at the column which originally contained the first character of the word. Otherwise, it ends up in the column preceding that column. This behavior ensures that no extra space characters appear between the word that preceded the deleted word and any punctuation which followed it.

The "consume_lb" command deletes leading blanks from the block of text described by the movement of the cursor. The command accepts only cursor-defined arguments. When it is done, it returns the cursor to its original position. This command is particularly userful for changing the indentation scheme of a section of source code.

Arguments

<block> The range of the "consume_lb" command is delimited by either horizontal or both horizontal and vertical cursor-positioning (see Section 3.6.2.2). The command deletes leading blanks within the block of text described by the cursor. The text affected includes all columns from the original position of the cursor up to but not including the column containing the cursor when the user types control-J. It includes all lines from the original position of the cursor to the line containing the cursor when the user types control-J, inclusive.

<range> A positive integer specifying the number of characters to consume from the current line. No matter how large an argument is used, the editor does not consume any characters beyond the current line. If the number specified is not an integer, the editor truncates it before using it.

EXAMPLES

1. control-J
2. TAB 4 control-J
3. TAB control-J
4. TAB w control-J

The first example deletes the character in the current column, shifts one column to the left all the text to the right of the cursor, and deletes trailing blanks.

(continued)

The second example deletes the character in the current column and the following three columns (a total of four characters), shifts four columns to the left all the text to the right of the cursor, and deletes trailing blanks.

The fourth example deletes all characters in the current line at and to the right of the cursor.

The third example deletes the word in which the cursor is positioned, shifts to the left the text to the right of the cursor to compensate for the deletion, and deletes any trailing blanks.

ERROR MESSAGES

Cursor not in word or word too long
: Either the cursor was not positioned at a character that the editor recognizes as part of a word or the length of the word exceeds the width of the screen.

Invalid parameter
: The argument specified to the "consume_char" command must be a positive integer.

SEE ALSO

word_def

consume_char

Delete the character in the current column and shift to the left any text to the right of the cursor.

SYNTAX

[TAB <range>] control-J

DESCRIPTION

The "consume_char" command deletes the character in the current column, shifts one column to the left all text to the right of the cursor, and deletes any trailing blanks (space characters between the last printed character in the line and the carriage return). The cursor is stationary.

Arguments

<range> A positive integer specifying the number of characters to consume from the current line. No matter how large an argument is used, the editor does not consume any characters beyond the current line. If the number specified is not an integer, the editor truncates it before using it.

EXAMPLES

1. control-J
2. TAB 4 control-J

The first example deletes the character in the current column, shifts one column to the left all the text to the right of the cursor, and deletes trailing blanks.

The second example deletes the character in the current column and the following three columns (a total of four characters), shifts four columns to the left all the text to the right of the cursor, and deletes trailing blanks.

SEE ALSO

consume

consume_1b

Delete leading blanks and shift to the left any text to the right of the block defined by the cursor.

SYNTAX

TAB <block> control-J

DESCRIPTION

The "consume_1b" command deletes leading blanks from the block of text described by the movement of the cursor. (A leading blank is any space character that precedes the first printed character in a line.) The "consume_1b" command accepts only cursor-defined arguments. When it is done, the command returns the cursor to its original position. It is particularly useful for changing the indentation scheme of a section of code.

Arguments

<block> The range of the "consume_1b" command is delimited by either horizontal or both horizontal and vertical cursor-positioning (see Section 3.6.2.2). The command deletes leading blanks within the block of text described by the cursor. The text affected includes all columns from the original position of the cursor up to but not including the column containing the cursor when the user types control-J. It includes all lines from the original position of the cursor to the line containing the cursor when the user types control-J, inclusive.

SEE ALSO

consume
shift_left
shift_right

consume_line

Delete all characters in the current line at and to the right of the cursor.

SYNTAX

TAB control-J

DESCRIPTION

The "consume_line" command deletes all characters in the current line at and to the right of the cursor. The cursor is stationary.

SEE ALSO

consume

consume_word

Delete the word in which the cursor is positioned and shift to the left the text to the right of the cursor.

SYNTAX

TAB w control-J

DESCRIPTION

The "consume_word" command deletes the word in which the cursor is positioned, shifts all text to the right of the cursor far enough to the left to compensate for the deletion, and deletes any trailing blanks (space characters between the last printed character in the line and the carriage return). The cursor may be positioned at any character in the word. By default, a word is defined as a string of alphanumeric characters. The user can, however, invoke the "word_def" command to add to the set of characters that the editor recognizes as part of a word.

When the user invokes this command, the editor first positions the cursor at the beginning of the word (defined by default as the first alphanumeric character to the right of the preceding nonalphanumeric character or, if there is no preceding nonalphanumeric character, as the character in the first column of the line). It then consumes characters until it comes to the end of the word. If a space character follows the word, the editor deletes it; otherwise, it deletes the space character preceding the word, if there is one, unless that space character is also preceded by a space character. Thus, if a space character follows the word, the cursor ends up at the column which originally contained the first character of the word. Otherwise, it ends up in the column preceding that column. This behavior ensures that no extra space characters appear between the word that preceded the deleted word and any punctuation that followed it.

ERROR MESSAGES

Cursor not in word.
The cursor was not positioned at a character that the editor recognizes as part of a word.

SEE ALSO

consume
word_def

control_char

Allow the user to insert a control character into the text.

SYNTAX

control-\

DESCRIPTION

The "control_char" command allows the user to insert a control character into the text. This function is particularly useful in cases in which control characters must be used to communicate with a printer.

When the user types the "control_char" command, the following prompt appears on the status line:

Enter control character

At the same time the editor inserts the character '^' in the current column and moves one column to the right the cursor and all text at and to the right of the cursor. When the user types a character, the editor inserts it to the right of the '^' character. To eliminate the control character from the file the user must remove both characters.

For purposes of alignment and justification the editor treats a sequence of control characters as if each element were a printable character. For example, the editor allows two columns in a line for the sequence "^H" even though it represents a backspace character.

SEE ALSO

align
rjustify

copy

Copy a word or a line to the appropriate buffer.

SYNTAX

```
TAB w control-L
[TAB [<range>]] control-L
```

Variations

copy_line	[TAB [<range>]] control-L
copy_word	TAB w control-L

DESCRIPTION

The "copy" command copies a word or the specified number of lines into the appropriate buffer.

The "copy_word" command copies the word in which the cursor is positioned into the smaller of the two buffers, called the word buffer. By default, a word is defined as a string of alphanumeric characters. The user can, however, invoke the "word_def" command to add to the set of characters that the editor recognizes as part of a word. If the cursor is not in a word, the editor returns an error message. Each time the user invokes the command, it deletes the contents of the buffer and replaces it with the newly specified word. The size of a word that can be copied into the word buffer is limited to the number of columns on the screen minus 1. The word buffer is shared with the "cut_word" and "replace" commands. The user can view the contents of the word buffer with the "display" command and retrieve its contents with either the "replace" or "paste_word" command. (However, in order to use the "replace" command successfully, the user must have previously defined a search string with the "search" command.)

The "copy-line" command copies the specified number of lines into a larger buffer known as the line buffer. Each time the command is invoked, it deletes the contents of the buffer and replaces it with the newly specified text. The line buffer can hold up to the number of lines in the edit window. If the user tries to copy a larger number of lines, the editor returns an error message. The line buffer is shared with the "cut_line" command. The "paste_line" command retrieves text from the line buffer. The user cannot view the contents of the line buffer without actually placing the text in the file. It is, of course, always possible to retrieve the text to examine the contents of the buffer and then to delete it.

(continued)

Arguments

<range> Specifies the number of lines to copy into the line buffer. If the user specifies a number, it must be a positive integer. The minimum value for the argument is 1; the maximum is the number of lines in the edit window. The default is 1. If the number is greater than the number of lines between the current line and the last line of the edit buffer, the editor adds the appropriate number of blank lines to the line buffer. The user can also specify the number of lines to copy by using vertical cursor-positioning (see Section 3.6.2.2).

EXAMPLES

1. TAB w control-L
2. TAB 5 control-L

The first example copies the word in which the cursor is positioned into the word buffer.

The second example copies the current line and the following four lines into the line buffer.

ERROR MESSAGES

Cursor not in word or word too long
The cursor must be in a word when the user invokes the command. The size of a word that can be copied into the word buffer is limited to the number of columns on the screen minus 1.

Invalid parameter
If the user specifies the range of the "copy_line" command with a number, it must be an integer between 1 and the number of lines in the edit window, inclusive. The only other acceptable method of specifying the range is vertical cursor-positioning.

SEE ALSO

copy_to_file
cut_line
cut_word
display
paste_line
paste_word
replace
search
word_def

copy_line

Copy a line to the line buffer.

SYNTAX

[TAB [<range>]] control-L

DESCRIPTION

The "copy_line" command copies the specified number of lines into a buffer known as the line buffer. Each time the command is invoked, it deletes the contents of the buffer and replaces it with the newly specified text. The line buffer can hold up to the number of lines in the edit window. If the user tries to copy a larger number of lines, the editor returns an error message. The line buffer is shared with the "cut_line" command. The "paste_line" command retrieves text from the line buffer.

The user cannot view the contents of the line buffer without actually placing the text in the file. It is, of course, always possible to retrieve the text to examine the contents of the buffer and then to delete it.

Arguments

<range> Specifies the number of lines to copy into the line buffer. If the user specifies a number, it must be a positive integer. The minimum value for the argument is 1; the maximum is the number of lines in the edit window. The default is 1. If the number is greater than the number of lines between the current line and the last line of the edit buffer, the editor adds the appropriate number of blank lines to the line buffer. The user can also specify the number of lines to copy by using vertical cursor-positioning (see Section 3.6.2.2).

EXAMPLES

1. TAB 5 control-L

This example copies the current line and the following four lines into the line buffer.

(continued)

ERROR MESSAGES

Invalid parameter
If the user specifies the range with a number, it must be an integer between 1 and the number of lines in the edit window, inclusive. The only other acceptable method of specifying the range is vertical cursor-positioning.

SEE ALSO

copy
cut_line
paste_line

copy_to_file

Copy the specified lines to a file.

SYNTAX

TAB <output_dir> [<file_name>] [TAB [<range>]] control-L

DESCRIPTION

The "copy_to_file" command copies the specified lines to a file. If the user does not specify a range (which must be preceded by a TAB character), it defaults to the contents of the entire edit buffer. If the destination file does not already exist, it is created with read and write permissions for all users.

Arguments

<file_name> The name of the file to which to copy the lines. The editor stores this name in a special buffer called the write-name buffer. Each time the user specifies the name of a file to the "copy_to_file" command, it deletes the contents of the buffer and replaces it with the newly specified name. If the user does not specify a name, "copy_to_file" uses whatever name is in the buffer. If the buffer is empty, the editor returns an error message. The write-name buffer is shared with the "move_to_file" command.

<output_dir> Specifies whether or not the editor should delete the contents of the file before copying text to it. The two valid arguments are the UniFLEX symbols for redirecting output: a single right-hand angle bracket, '>', specifies that any text already in the file should be deleted; two right-hand angle brackets, ">>", specify that the editor should append the text to any text already in the file.

<range> Specifies which lines to copy. The range may be a number, in which case it indicates how many lines to copy, starting with the current line. The default is 1. If the number specified is larger than the number of lines between the current line and the last line in the edit buffer, the editor returns an error message.

The range may also specify the line numbers of

(continued)

the first and last lines to copy. The editor copies all lines between the specified lines, inclusive. In such a case the format for the range is

#<line_num_1>,<line_num_2>

The line numbers must appear sequentially--that is, the first number specified must be smaller than the second. Otherwise, the editor returns an error message.

Finally, the user can express the extent of the range by using vertical cursor-positioning (see Section 3.6.2.2).

EXAMPLES

1. TAB >transfer TAB 10 control-L
2. TAB >>transfer TAB #100,150 control-L
3. TAB >/usr/jeremy/test TAB #100,150 control-L

The first example copies the current line and the following nine lines to the file named "transfer" in the working directory. If the file does not already exist, the "copy_to_file" command creates it; if it does exist, "copy_to_file" deletes the contents before copying any text to it.

The second example appends lines 100 through 150, inclusive, of the file being edited to the file named "transfer" in the working directory. If the file does not already exist, the "copy_to_file" command creates it.

The third example copies lines 100 through 150, inclusive, of the file being edited to the file "/usr/jeremy/test". If the file does not already exist, the "copy_to_file" command creates it; if it does exist, "copy_to_file" deletes the contents before copying to it.

ERROR MESSAGES

Invalid parameter
If the user specifies the range with a number, it must be a positive integer. If the user chooses to specify two line numbers, the smaller number must precede the larger. If the user specifies the range with cursor-positioning, the command accepts only vertical movement.

(continued)

Line no longer in edit buffer
The line number specifying the beginning of the range is less than the number of the first line in the edit buffer. The user may access the line in question by invoking the "rewind" command and, if necessary, flushing text from the edit buffer to the disk until the line is in the buffer.

Line past end of buffer
The line number specifying the end of the range is greater than the number of the last line in the edit buffer. If the file is completely loaded in the edit buffer, no such line exists. If the line does exist but has not yet been loaded into memory, the user can access it by using a form of the "flush" command.

No file name specified
The user tried to invoke the "copy_to_file" command without specifying the name of a file, but the write-name buffer is empty.

SEE ALSO

flush
move_to_file
read_file

copy_word

Copy a word to the word buffer.

SYNTAX

TAB w control-L

DESCRIPTION

The "copy_word" command copies the word in which the cursor is positioned into a buffer called the word buffer. By default, a word is defined as a string of alphanumeric characters. The user can, however, invoke the "word_def" command to add to the set of characters that the editor recognizes as part of a word. If the cursor is not in a word, the editor returns an error message. Each time the user invokes the command, it deletes the contents of the buffer and replaces it with the newly specified word. The size of a word that can be copied into the word buffer is limited to the number of columns on the screen minus 1. The word buffer is shared with the "cut_word" and "replace" commands. The user can view the contents of the buffer with the "display" command or retrieve it with either the "replace" or "paste_word" command. (However, in order to use the "replace" command successfully, the user must have previously defined a search string with the "search" command.)

ERROR MESSAGES

Cursor not in word or word too long
: The cursor must be in a word when the user invokes the command. The size of a word that can be copied into the word buffer is limited to the number of columns on the screen minus 1.

SEE ALSO

copy
cut_word
display
paste_word
replace
search
word_def

cut

Copy a word or a line to the appropriate buffer and delete it from the text, or move the text from the following line to the end of the current line.

SYNTAX

```
TAB w control-K
[TAB [<range>]] control-K
```

Variations

cut_word	TAB w control-K
cut_line	[TAB <range>] control-K
join	TAB control-K

DESCRIPTION

The "cut" command copies a word or the specified number of lines to the appropriate buffer and deletes them from the text, or moves the text from the following line to the end of the current line.

The "cut_word" command copies the word in which the cursor is positioned into a buffer called the word buffer and deletes it from the text. By default, a word is defined as a string of alphanumeric characters. The user can, however, invoke the "word_def" command to add to the set of characters that the editor recognizes as part of a word. If the cursor is not in a word, the editor returns an error message. The size of a word that can be copied into the word buffer is limited to the number of columns on the screen minus 1. Each time the user invokes the command, it deletes the contents of the buffer and replaces it with the newly specified word. The word buffer is shared with the "copy_word" and "replace" commands. The user can view the contents of the word buffer with the "display" command or retrieve it with either the "replace" or the "paste_word" command. (However, in order to use the "replace" command successfully, the user must have previously defined a search string with the "search" command.)

The "cut_line" command copies the specified number of lines into a buffer called the line buffer and deletes them from the text. Each time the command is invoked, it deletes the contents of the buffer and replaces it with the newly specified text. The line buffer can hold up to the number of lines in the edit window. If the user tries to copy a larger number of lines, the editor returns an error message. The line buffer is shared with the "copy_line" command. The "paste_line" command retrieves text from the line buffer.

(continued)

The user cannot view the contents of the line buffer without actually placing the text in the file. It is, of course, always possible to retrieve the text to examine the contents of the buffer and then to delete it.

The final form of the "cut" command has nothing to do with either of the buffers. The "join" command essentially cuts out the following line of text and pastes it onto the end of the current line. If the current line containing does not have room to accommodate the addition, or if the cursor is not at the end of the line when the user executes the command the returns an error message.

It is not always obvious which column contains the last character of a line. When the user enters any printable character (including the space character) in a line, the editor enters a space character in each column between that character and the column which previously contained the last character. If the user erases the last character, the character preceding it--which may be a space character inserted by the editor, not the user--becomes the last character in the line. However, when it saves the file, the editor truncates any trailing space characters (space characters which appear in a line after the last visible character).

Arguments

<range> Specifies the number of lines to cut. If the user specifies a number, it must be a positive integer. The minimum value for the argument is 1; the maximum is the number of lines in the edit window. The default is 1. If the number is greater than the number of lines between the current line and the last line of the edit buffer, the editor adds the appropriate number of blank lines to the line buffer. The user can also specify the number of lines to cut by using vertical cursor-positioning (see Section 3.6.2.2).

EXAMPLES

1. TAB w control-K
2. TAB 5 control-K
3. TAB control-K

The first example copies the word in which the cursor is positioned into the word buffer and deletes it from the text.

The second example copies the current line and the following four lines of text into the line buffer. As it does so, it deletes them from the text.

(continued)

The third example moves the text from the following line to the end of the current line.

ERROR MESSAGES

Clear to end of line first
: In order for the "join" command to succeed the cursor must be positioned at or beyond the last character in the line.

Cursor not in word or word too long
: The cursor must be in a word when the user invokes the "cut_word" command. The size of a word that can be copied into the word buffer is limited to the number of columns on the screen minus 1.

Invalid parameter
: If the user specifies the range of the "cut_line" command with a number, it must an integer between 1 and the number of lines in the edit window, inclusive. The only other acceptable method of specifying the range is vertical cursor-positioning.

Not enough room in line
: The current line does not have enough room to accommodate the addition of the text from the following line.

SEE ALSO

copy_word
copy_line
display
paste_line
paste_word
replace
search
word_def

cut_line

Copy a line to the line buffer and delete it from the text.

SYNTAX

TAB [<range>] control-K

DESCRIPTION

The "cut_line" command copies the specified number of lines into a buffer called the line buffer and deletes them from the text. Each time the command is invoked, it deletes the contents of the buffer and replaces it with the newly specified text. The line buffer can hold up to the number of lines in the edit window. If the user tries to copy a larger number of lines, the editor returns an error message. The line buffer is shared with the "copy_line" command. The "paste_line" command retrieves text from the line buffer.

The user cannot view the contents of the buffer without actually placing the text in the file. It is, of course, always possible to retrieve the text to examine the contents of a buffer and then to delete it.

Arguments

<range> Specifies the number of lines to cut. If the user specifies a number, it must be a positive integer. The minimum value for the argument is 1; the maximum is the number of lines in the edit window. The default is 1. If the number is greater than the number of lines between the current line and the last line of the edit buffer, the editor adds the appropriate number of blank lines to the line buffer. The user can also specify the number of lines to cut by using vertical cursor-positioning (see Section 3.6.2.2).

EXAMPLES

1. control-K
2. TAB 5 control-K

The first example copies the current line into the line buffer. As it does so, it deletes it from the text.

The second example copies the current line and the following four lines of text into the line buffer. As it does so, it deletes them from the text.

(continued)

ERROR MESSAGES

Invalid parameter
If the user specifies the range with a number, it must an integer between 1 and the number of lines in the edit window, inclusive. The only other acceptable method of specifying the range is vertical cursor-positioning.

SEE ALSO

copy_line
paste_line

cut_word

Copy a word to the word buffer and delete it from the text.

SYNTAX

TAB w control-K

DESCRIPTION

The "cut_word" command copies the word in which the cursor is positioned word into a buffer called the word buffer and deletes it from the text. By default, a word is defined as a string of alphanumeric characters. The user can, however, invoke the "word_def" command to add to the set of characters that the editor recognizes as part of a word. If the cursor is not in a word, the editor returns an error message. The size of a word that can be copied into the word buffer is limited to the number of columns on the screen minus 1. Each time the user invokes the command, it deletes the contents of the buffer and replaces it with the newly specified word. The word buffer is shared with the "copy_word" and "replace" commands. The user can view the contents of the word buffer with the "display" command or retrieve its contents with either the "replace" or the "paste_word" command. (However, in order to use the "replace" command successfully, the user must have previously defined a search string with the "search" command.)

ERROR MESSAGES

Cursor not in word or word too long
: The cursor must be in a word when the user invokes the "cut_word" command. The size of a word that can be copied into the word buffer is limited to the number of columns on the screen minus 1.

SEE ALSO

copy_word
display
paste_word
replace
search
word_def

delete

Delete a line or a character and shift the following text to compensate for the deletion.

SYNTAX

[TAB [<range_or_block>]] control-B

Variations

delete_bl	control-B
delete_line	TAB [<range>] control-B
shift_left	TAB <block> control-B

DESCRIPTION

The "delete" command removes a line or a character and shifts the following text to compensate for the deletion.

The "delete_bl" command deletes only blank lines. It deletes the current line and all blank lines between that line and the next nonblank line, moving the text in the remainder of the edit buffer up to compensate for the deletion. The cursor remains stationary. If the current line is not blank, the command has no effect.

The "delete_line" command deletes both blank and nonblank lines. The user specifies the range of the command with an argument from the status line. Unless the user specifies the range with line numbers, the cursor remains stationary. If the user specifies the range with line numbers, the editor command positions the cursor at the first line after the deletion and places that line approximately one-third of the way down the edit window unless not enough lines of text remain above the deletion to fill the edit window. In such a case, the editor still positions the cursor at the first line after the deletion but places the first line of the edit buffer at the top of the edit window.

The "shift_left" command shifts text to the left by deleting characters in the block of text delimited by the movement of the cursor. This form of the command accepts only cursor-defined arguments (see Section 3.6.2.2).

Arguments

<range> Specifies the number of lines to delete, starting with the current line. If the user specifies a number, it must be a positive integer. The default is 1. If the number specified is larger than the number of the last line in the edit buffer, the

(continued)

"delete_line" command deletes all the lines from the current line to the last line in the buffer, inclusive.

The range may also specify the line numbers of the first and last lines to delete. The command deletes all lines between the specified lines, inclusive. In such a case the format for the range is

#<line_num_1>,<line_num_2>

The line numbers must appear sequentially--that is, the first number specified must be smaller than the second. Otherwise, the editor returns an error message.

Finally, the user can express the extent of the range by using vertical cursor-positioning (see Section 3.6.2.2).

<block> The range of the "shift_left" command is delimited by either horizontal or both horizontal and vertical cursor-positioning (see Section 3.6.2.2). The command deletes the block of text described by the movement of the cursor. The text affected includes all columns from the original position of the cursor up to but not including the column containing the cursor when the user types control-B. It includes all lines from the original position of the cursor to the line containing the cursor when the user types control-B, inclusive.

EXAMPLES

1. control-B
2. TAB control-B
3. TAB 3 control-B

The first example deletes the blank line the cursor is on and all blank lines between that line and the next nonblank line, moving the following text up to compensate for the deletion. If the current line is not blank, the command has no effect.

The second example deletes the current line, moving the text in the remainder of the edit buffer up one line to compensate for the deletion.

The third example deletes the current line and the following two lines, moving the text in the remainder of the edit buffer up three lines to compensate for the deletion.

(continued)

NOTES

- A line containing even a single space character is not a blank line and must, therefore, be deleted with the "delete_line", not the "delete_bl", command.

- The user can create a powerful tool by using the "shift_left" command when the editor is in word-tab mode. This combination makes it easy to excise a group of words from a line of text by using the "rtab" and "ltab" commands to perform the horizontal cursor-positioning.

ERROR MESSAGES

Invalid parameter
: If the user specifies the range with a number, it must be a positive integer. If the user chooses to specify two line numbers, the smaller number must precede the larger.

Line no longer in edit buffer
: The line number specifying the beginning of the range is less than the number of the first line in the edit buffer. The user may access the line in question by invoking the "rewind" command and, if necessary, flushing text from the edit buffer to the disk until the line is in the buffer.

Line past end of buffer
: The line number specifying the end of the range is greater than the number of the last line in the edit buffer. If the file is completely loaded into the edit buffer, no such line exists. If the file is not completely loaded, the user may access the line in question by using a form of the "flush" command that puts the entire range into the edit buffer.

SEE ALSO

consume
cut
flush
move_to_file
rewind
word_tab

delete_bl

Delete the blank current line and all blank lines between that line and the next nonblank line.

SYNTAX

control-B

DESCRIPTION

The "delete_bl" command deletes only blank lines. It deletes the current line and all blank lines between that line and the next nonblank line, moving the text in the remainder of the edit buffer up to compensate for the deletion. The cursor remains stationary. If the current line is not blank, the command has no effect.

NOTES

- A line containing even a single space character is not a blank line and must, therefore, be deleted with the "delete_line", not the "delete_bl", command.

SEE ALSO

consume
delete

delete_line

Delete one or more lines of text and shift the following text to compensate for the deletion.

SYNTAX

TAB [<range>] control-B

DESCRIPTION

The "delete_line" command deletes both blank and nonblank lines. The user specifies the range of the command with an argument from the status line. Unless the user specifies the range with line numbers, the cursor remains stationary. If the user specifies the range with line numbers, the editor positions the cursor at the first line after the deletion and places that line approximately one-third of the way down the edit window unless not enough lines of text remain above the deletion to fill the edit window. In such a case, the editor still positions the cursor at the first line after the deletion but places the first line of the edit buffer at the top of the edit window.

Arguments

<range> Specifies the number of lines to delete, starting with the current line. If the user specifies a number, it must be a positive integer. The default is 1. If the number specified is larger than the number of the last line in the edit buffer, the "delete_line" command deletes all the lines between the current line and the last line in the buffer, inclusive.

The range may also specify the line numbers of the first and last lines to delete. The command deletes all lines between the specified lines, inclusive. In such a case the format for the range is

<line_num_1>,<line_num_2>

The line numbers must appear sequentially--that is, the first number specified must be smaller than the second. Otherwise, the editor returns an error message.

(continued)

Finally, the user can express the extent of the range by using vertical cursor-positioning (see Section 3.6.2.2).

EXAMPLES

1. TAB control-B
2. TAB 3 control-B

The first example deletes the current line, moving the text in the remainder of the edit buffer up one line to compensate for the deletion.

The second example deletes the current line and the following two lines, moving the text in the remainder of the edit buffer up three lines to compensate for the deletion.

ERROR MESSAGES

Invalid parameter
If the user specifies the range with a number, it must be a positive integer. If the user chooses to specify the range with two line numbers, the smaller number must precede the larger.

Line no longer in edit buffer
The line number specifying the beginning of the range is less than the number of the first line in the edit buffer. The user may access the line in question by invoking the "rewind" command and, if necessary, flushing text from the edit buffer to the disk until the line is in the buffer.

Line past end of buffer
The line number specifying the end of the range is greater than the number of the last line in the edit buffer. If the file is completely loaded into the edit buffer, no such line exists. If the file is not completely loaded, the user may access the line in question by using a form of the "flush" command that puts the entire range into the edit buffer.

SEE ALSO

consume
delete
flush
rewind

display

Display the contents of the specified buffer.

SYNTAX

TAB d <buffer_name> RETURN

DESCRIPTION

The "display" command displays the contents of the specified buffer on the status line.

Arguments

<buffer_name> The name of the buffer to display. The valid arguments are 'd', which displays the word-definition buffer; 's', which displays the search buffer; and 'w', which displays the word buffer.

EXAMPLES

1. TAB s RETURN

This example displays the contents of the search buffer on the status line.

MESSAGES

Buffer is empty.
If the specified buffer is empty, the editor sends this message to the status line. A bell (control-G) accompanies the message in order to distinguish it from the case in which the buffer actually contains the string "Buffer is empty.".

ERROR MESSAGES

Invalid parameter
The only valid arguments are 'd', 's', and 'w'.

SEE ALSO

copy_word
cut_word
search
replace
word_def

down

Move the cursor to the next line of the edit buffer.

SYNTAX

control-C

DESCRIPTION

The "down" command moves the cursor to the next line in the edit buffer without altering its horizontal position. If that line of text is already in the edit window, the "down" command simply moves the cursor down one line. If, however, the cursor is at the bottom line of the edit window, the command scrolls the text up one line so that the next line of text appears at the bottom of the window. In this case the cursor actually remains stationary while the text scrolls past it. If the cursor is on the last line of the edit buffer, the "down" command behaves normally, scrolling a blank line into the edit window. If the memory buffer is full, the command has no effect.

exit

End the editing session.

SYNTAX

```
TAB [<save_instr>] [<[<file_name>]] control-U
TAB +[<command_list>] control-U
```

Variations

```
abort       TAB q [<[<file_name>]] control-U
rewind      TAB [<save_instr>] < control-U
save        TAB [<y_or_d>] <[<file_name>] control-U
temp_exit   TAB +[<command_list>] control-U
```

DESCRIPTION

The "exit" command ends the editing session. The various forms of the command control what happens to the changes made during the editing session, what happens to any backup file that is already in existence, and whether or not the editor creates a backup file if none exists.

When the editor is invoked with a single argument, it copies the file being edited into memory and creates a temporary file in the same directory as the file being edited. The temporary file is used during the editing session if the user fills the edit buffer and at the end of the editing session if the user does not use the "abort" command. Any changes the user makes during the editing session are made on the copy in memory. What happens at the end of the editing session depends on the particular form of the "exit" command the user selects.

The "abort" command returns control to the operating system without writing any of the changes made during the editing session to the disk. Thus, the original file and the backup file (if one exists) remain unchanged.

The "save" command copies the file as it exists in memory to the temporary file created by the editor at the beginning of the editing session. By default, if no backup file exists, the editor renames the original file by appending the characters ".bak" to its name. If this addition would result in a longer name than the operating system allows (fourteen characters for the 6809; fifty-five, for the 68xxx), the editor truncates the original name before adding the suffix. It then renames the temporary file, giving it the name of the original file.

If a file named <file_name.bak> already exists, the editor asks for permission to delete that file. In response to this question the user must type either 'y' for "yes" or 'n' for "no". In response to a 'y',

(continued)

the editor deletes the file whose name is <file_name.bak>, gives <file_name> the name <file_name.bak>, and names the temporary file <file_name>. In response to an 'n', the editor deletes the contents of the original file, renames the temporary file <file_name>, and leaves the file <file_name.bak> untouched. The editor accepts no characters other than a 'y' or an 'n' at this point. It is not necessary to type a carriage return after the response.

The "rewind" command exits from the current editing session and automatically reinvokes the editor with the same file. The contents of the word, line, and search buffers remain intact from one session to the next.

The "temp_exit" command gives the user access to the shell program without actually ending the editing session. By default, the command calls another shell program in an interactive mode. Logging out of the shell (by typing "log" or control-D) returns control to the editor.

The user may also specify a list of commands to be executed when invoking the "temp_exit" command. In such a case, the editor calls a shell program which, on completion of the commands, automatically returns control to the editor. If a task terminates abnormally, the editor displays a message to that effect on the status line when it regains control. The message only reports that the task failed; it does not give a reason.

It is possible to run a command in background by terminating it with an ampersand, '&'. However, a user who chooses to do so should redirect both standard error and standard output (which requires the "nshell" command in Utilities Package II for the 6809), so that the task does not send information to the screen, thereby disturbing the editor's display. (The editor is careful to rewrite only that part of the screen which it knows has been altered. It is not possible to tell the editor to rewrite the entire screen without scrolling the text.) Running a task is the background is somewhat risky as the user cannot determine if the task has terminated or if it terminated abnormally.

Arguments

<command_list>	A list of UniFLEX commands. In a list of more than one command, every command except the last one must be followed by a semicolon, ';'.
<file_name>	The name of the file with which to begin the next editing session. If the user begins an editing session this way, the contents of the word, line, and search buffers remain intact from one session to the next.
<save_instr>	Instructs the editor how to handle the backup file. The valid arguments are 'q', 'd', and 'y'.

(continued)

A 'q' tells the "exit" command not to save the edited version of the file. Both the original file and any existing backup file remain intact.

A 'd' tells the editor to retain the contents of a previously existing backup file and to discard the contents of the original file. The temporary file is then given the name of the original file. If no backup file exists, the 'd' argument tells the editor not to create one.

A 'y' tells the editor to replace the contents of a previously existing backup file with the contents of the original file. The temporary file is then given the name of the original file. If no backup file exists, the 'y' argument has no affect on the default behavior of the editor.

EXAMPLES

1. TAB control-U
2. TAB q <control-U
3. TAB y <new_file control-U
4. TAB d control-U
5. TAB + control-U
6. TAB + more other_file
7. TAB + pascal test.p >%>output& control-U

The first example writes the contents of the edit buffer to the temporary file on the disk. If necessary, it creates a backup file. If a backup file already exists, it asks for permission to delete it.

The second example exits from the editing session without writing any of the changes made during the session to the disk and deletes the temporary file. It then begins an edit session on the same file as the previous editing session.

The third example exits from the editing session, deleting any existing backup file, so that the original file replaces the backup file and the copy from memory replaces the original file. It then begins an editing session on the file named "new_file".

The fourth example writes the contents of the edit buffer to the disk. If there is no backup file, it does not create one. If a backup file already exists, the editor does not delete it. It deletes the contents of the original file and gives that name to the temporary file.

(continued)

The fifth example invokes a shell program interactively. In response to the UniFLEX prompt the user may enter any command. Control returns to the editor when the user logs out of the shell.

The sixth example invokes a shell program which executes the command "more other_file". At the completion of the "more" command control automatically returns to the editor.

The seventh example invokes a shell program which executes the command "pascal test.p" in the background. Standard error and standard output are both redirected to a file named "output". When the shell program sends the task to the background, control returns to the editor.

MESSAGES

Delete existing backup file?
The user has ended the editing session without giving instructions about an existing backup file. In response to this prompt the user should type a 'y' for "yes" or an 'n' for "no".

flush

Write the contents of the edit buffer to the temporary file and, if so instructed, read more of the original file into the edit buffer.

SYNTAX

TAB <b_n_or_w> control-U

DESCRIPTION

The "flush" command writes some or all of the contents of the edit buffer to the disk and, if so instructed, reads in text from the file being edited until it either refills the buffer or reaches the end of the file.

Once text has been written to the disk, it is inaccessible to both the editor and the user during that editing session. However, the user can invoke the "rewind" command to start an editing session at the beginning of the file.

Arguments

- b Flush from the top to the bottom of the edit buffer. The editor writes to the disk the contents of the edit buffer from the first line to the last line, inclusive. It then reads in more text from the file being edited.
- n Flush from the top of the edit buffer to the current line. The editor writes to the disk the contents of the edit buffer from the first line up to but not including the current line. It then reads in more text from the file being edited.
- w Flush from the top of the edit buffer to the current line. The editor writes to the disk the contents of the edit buffer from the first line up to but not including the current line. It does not read in more text from the file being edited. Thus, it leaves the user room for additions to the text.

EXAMPLES

1. TAB b control-U
2. TAB n control-U
3. TAB w control-U

The first example writes to the disk the contents of the edit buffer from the first line to the last line, inclusive. It then reads into the edit buffer as much text as possible from the file being edited.

(continued)

The second example writes to the disk the contents of the edit buffer from the first line up to but not including the current line. It then reads into the edit buffer as much text as possible from the file being edited.

The third example writes to the disk the contents of the edit buffer from the first line up to but not including the current line. It does not read any new text into the edit buffer.

NOTES

- When the "flush" command writes to the disk, it writes either to a temporary file named "edit.<num>", where <num> is the identification number (task ID) of the editing session. or to a file whose name was specified when the user invoked the editor.

- The editor always truncates trailing blank lines (lines between the last visible line in the edit buffer and the end of the edit buffer that are either completely blank or consist entirely of space characters) when it writes the entire buffer to the disk. Therefore, in order to avoid the loss of blank lines that belong in a file, the "flush" command never splits a file on a blank line when reading new text into the edit buffer.

MESSAGES

File not completely loaded
If the "flush" command cannot fit all of the remainder of the file being edited into the edit buffer, it reads in as much as possible and sends this message to the status line.

SEE ALSO

rewind

global

Use as the range for the following command either the entire edit buffer or the contents of the buffer between the current line and the last line of the buffer, inclusive.

SYNTAX

[TAB [<arg>]] control-^

DESCRIPTION

The "global" command sets a flag so that if the command which follows it is one which honors that flag, the editor executes it repeatedly. If the user precedes the "global" command with the tab character (and an argument, if appropriate), the editor uses the entire edit buffer, starting at line 1, as the range for the following command. If the user executes the "global" command without preceding it with the tab character, the editor uses the contents of the buffer between the current line and the last line of the buffer, inclusive, as the range for the following command.

When the "global" flag is in effect, the string "Global" appears in the status line.

Arguments

<arg> The nature of this argument depends on the command that the user wishes to execute globally.

EXAMPLES

1. TAB string_1 TAB string_2 control-^ control-X
2. TAB testing control-^ control-X
3. control-^ control-X
4. control-^ TAB J control-G
5. TAB control-^ TAB J control-G

The first example replaces every occurrence in the edit buffer of the search string, "string_1", with "string_2".

The second example replaces every occurrence in the edit buffer of whatever string is in the search buffer with the string "testing".

The third example replaces all occurrences between the current line and the last line of the edit buffer, inclusive, of whatever string is in the search buffer with whatever string is in the word buffer.

(continued)

The fourth example aligns the text between the current line and the end of the edit window, inclusive.

The fifth example aligns the text in the entire edit window.

SEE ALSO

align
replace
search_rep

goto

Position the cursor at the specified line or column.

SYNTAX

```
[TAB [<line_num>]] control-T
TAB <column_num> control-F
```

Variations

bottom	TAB control-T
column	TAB <column_num> control-F
line	TAB <line_num> control-T
top	control-T

DESCRIPTION

Depending on the form used, the "goto" command positions the cursor at the line or column specified by the user.

The "top" command positions the cursor at the first column of the first line of the edit buffer and places that line at the top of the edit window. Similarly, the "bottom" command positions the cursor at the first column of the last line of the edit buffer and places that line at the bottom of the edit window.

The "line" command positions the cursor at the first column of the line specified by the user. If the specified line is one of the first few lines of the edit buffer, the "line" command positions the first line of the file at the top of the edit window. Otherwise, it positions the line approximately one-third of the way down the edit window. The "line" command can be used to position the cursor at the first or last line of the edit buffer, but the "top" and "bottom" commands are simpler to use.

The "column" command positions the cursor at the specified column of the current line.

Arguments

<column_num>	The number of the column at which to position the cursor. The number specified must be a positive integer. If it is greater than 128 (the number of columns in a line), the editor divides the number by 128 and uses the remainder as the argument.
<line_num>	The number of the line at which to position the cursor. The number specified must be a

(continued)

positive integer. If the number used is greater than the number of the last line in the edit buffer, the "line" command positions the cursor at the beginning of the last line in the buffer. If the number used is less than the number of the first line in the edit buffer, the editor returns an error message.

EXAMPLES

1. control-T
2. TAB control-T
3. TAB 157 control-T
4. TAB 24 control-F

The first example positions the cursor at the first column of the first line of the edit buffer and places that line at the top of the edit window.

The second example positions the cursor at the first column of the last line in the edit buffer and places that line at the bottom of the edit window.

The third example positions the cursor at the first column of line 157 and places that line approximately one-third of the way down the edit window (assuming that line 157 is currently in the edit buffer).

The fourth example positions the cursor at column 24 of the current line.

ERROR MESSAGES

Invalid parameter
The number specified must be a positive integer.

Line no longer in edit buffer
The specified line number is less than the number of the first line in the edit buffer. The user may access the line in question by invoking the "rewind" command and, if necessary, flushing text from the edit buffer to the disk until the line is in the buffer.

SEE ALSO

flush
rewind

home

Position the cursor at the first column of the first line of the edit window or at the first column of the last line of the edit window.

SYNTAX

control-D

DESCRIPTION

If the cursor is not in the first column of the first line in the edit window, the "home" command puts it there. If the cursor is already in the first column of the first line in the edit window, the "home" command moves it to the first column of the last line of the edit window.

When the "home" command is used as part of a cursor-defined argument (see Section 3.6.2.2), the command toggles the cursor between the line the cursor was on when the user initiated the command and the last line of the edit window. While doing so, it maintains the horizontal position of the cursor.

home_line

Position the current line at the top of the edit window.

SYNTAX

TAB control-N

DESCRIPTION

The "home_line" command positions the current line at the top of the edit window. The cursor remains in the same place relative to the text.

insert

Insert a blank line or a space character at the designated spot and shift the following text to make room for the insertion, or insert a blank line below the current line and move the text at and to the right of the cursor to that line.

SYNTAX

[TAB [<range_or_block>]] control-W

Variations

insert_line	[TAB <range>] control-W
shift_right	TAB <block> control-W
split	TAB control-W

DESCRIPTION

The "insert" command moves all text at and below the cursor down or to the right and inserts a blank line or a space character at the position of the cursor.

The "insert_line" command inserts the specified number of blank lines between the current line and the line preceding it, then positions the cursor at the first of the newly inserted blank lines (without altering its horizontal position).

The "shift_right" command shifts text to the right by inserting space characters in the block of text delimited by movement of the cursor. This form of the command only accepts cursor-defined arguments.

The "split" command splits a line of text into two lines by inserting a blank line between the current line and the line following it and moving all text at and to the right of the cursor to the newly inserted line. The command can conceptually be reconciled with the other forms of the "insert" command by considering that it behaves as if it were inserting an end-of-line character at the current column, forcing the remaining text onto a new line. In actual fact, however, the command does not put an end-of-line character into the text.

Arguments

<range> Specifies the number of blank lines to insert. If the user specifies a number, it must be a positive integer. The default is 1. The user can also specify the number of lines to insert by using vertical cursor-positioning (see Section 3.6.2.2). The number of blank lines inserted is equal to the

(continued)

number of lines covered by the cursor, including the original line.

<block> The range of the "shift_right" command is delimited by either horizontal or both horizontal and vertical cursor-positioning (see Section 3.6.2.2). The number of space characters inserted on each line is equal to the number of columns covered by the cursor from its original position up to but not including the final column. The command alters all lines between the original and final positions of the cursor, inclusive.

EXAMPLES

1. control-W
2. TAB control-W
3. TAB 5 control-W

The first example inserts one blank line between the current line and the line preceding it.

The second example inserts a blank line between the current line and the line following it and moves all text at and to the right of the cursor onto the new line.

The third example inserts five blank lines between the current line and the line preceding it.

NOTES

- The user may insert individual characters into the text by putting the editor into character-insert mode with the "char_insert" command.

ERROR MESSAGES

Invalid parameter
The number specified must be a positive integer.

SEE ALSO

char_insert

insert_line

Insert the specified number of blank lines between the current line and the line preceding it.

SYNTAX

[TAB <range>] control-W

DESCRIPTION

The "insert_line" command inserts the specified number of blank lines between the current line and the line preceding it, then positions the cursor at the first of the newly inserted blank lines (without altering its horizontal position).

Arguments

<range> Specifies the number of blank lines to insert. If the user specifies a number, it must be a positive integer. The default is 1. The user can also specify the number of lines to insert by using vertical cursor-positioning (see Section 3.6.2.2). The number of blank lines inserted is equal to the number of lines covered by the cursor, including the original line.

EXAMPLES

1. control-W
2. TAB 5 control-W

The first example inserts one blank line between the current line and the line preceding it.

The second example inserts five blank lines between the current line and the line preceding it.

ERROR MESSAGES

Invalid parameter
The number specified must be a positive integer.

(continued)

SEE ALSO

insert

join

Move the text from the following line to the end of the current line.

SYNTAX

TAB control-K

DESCRIPTION

The "join" command essentially cuts out the following line of text and pastes it onto the end of the current line. If the current line does not have room to accommodate the addition or if the cursor is not at the end of the line when the user executes the command, the editor returns an error message.

It is not always obvious which column contains the last character of a line. When the user enters any printable character (including the space character) in a line, the editor enters a space character in each column between that character and the column which previously contained the last character. If the user erases the last character, the character preceding it--which may be a space character inserted by the editor, not the user--becomes the last character in the line. However, when it saves the file, the editor truncates any trailing space characters (space characters between the last printed character in the line and the carriage return).

ERROR MESSAGES

Clear to end of line first
: In order for the "join" command to succeed the cursor must be positioned at or beyond the last character in the line.

Not enough room in line
: The current line does not have enough room to accommodate the addition of the text from the following line.

SEE ALSO

align
split

left

Position the cursor at the previous column.

SYNTAX

control-S

DESCRIPTION

The "left" command positions the cursor at the previous column in the edit buffer. If the cursor is in the first column of the first line of the edit buffer, the "left" command has no effect. Otherwise, if the cursor is in the first column of a primary window, the "left" command moves it to the last column of the previous line and switches the display to a secondary window (see Section 3.3.1). If the cursor is in the first column of a secondary window, the "left" command moves it to the previous column and changes the display to a primary window.

line

Position the cursor at the first column of the specified line.

SYNTAX

TAB <line_num> control-T

DESCRIPTION

The "line" command positions the cursor at the first column of the line specified by the user. If that line is one of the first few lines of the edit buffer, the "line" command positions the first line of the file at the top of the edit window. Otherwise, it positions the specified line approximately one-third of the way down the edit window.

The "line" command can be used to position the cursor at the first or last line of the edit buffer, but the "top" and "bottom" commands are easier to use.

Arguments

<line_num> The line number of the target. The number specified must be a positive integer. If the number used is greater than the number of the last line in the edit buffer, the "line" command positions the cursor at the beginning of the last line of the edit buffer. If the number used is less than the number of the first line of the edit buffer, the editor returns an error message.

EXAMPLES

1. TAB 157 control-T

This example positions the cursor at the first column of line 157, assuming that line 157 is currently in the edit buffer.

ERROR MESSAGES

Invalid parameter
The number specified must be a positive integer.

Line no longer in edit buffer
The specified line number is less than the number of the first line in the edit buffer. The user may access the line in question by issuing the "rewind" command and, if necessary, flushing text from

(continued)

the edit buffer to the disk until the line is in the buffer.

SEE ALSO

flush
goto
rewind

ltab

Move the cursor to the left until it reaches a column that contains a tab stop.

SYNTAX

control-A

DESCRIPTION

The "ltab" command moves the cursor to the left until it reaches a column that contains a tab stop. If the cursor is in the first column of a secondary window, the "ltab" command moves it to the preceding tab stop and switches the display to a primary window (see Section 3.3.1). If the cursor is in column 1, the "ltab" command has no effect.

By default, tab stops occur every eighth column starting in column 1. The first and last columns of the edit window always contain tab stops. The user may set and clear tab stops with the "set_tab" and the "clr_tab" commands.

If the editor is in word-tab mode, the "ltab" command moves the cursor to the left until it reaches the beginning of a word. By default, a word is defined as a string of alphanumeric characters. The user can, however, invoke the "word_def" command to add to the set of characters that the editor recognizes as part of a word.

SEE ALSO

rtab
set_tab
clr_tab
word_def
word_tab

move_lines

Move a section of text from one place in the file to another.

SYNTAX

TAB <range> control-O

DESCRIPTION

The line buffer can hold no more lines than can fit in the edit window. Thus, the "copy_line", the "cut_line", and the "paste_line" commands are fairly restricted in the amount of text they can work with. The "move_lines" command allows the user to avoid this limit when moving text (the limit cannot be avoided when copying text). The command moves the specified lines of text from their original place in the file to a new location between the current line and the line preceding it. It positions the cursor at the first line of text that was moved without altering its horizontal position. No scrolling occurs.

In order for the command to succeed the current line must be outside the range of the lines being moved. The limit to the number of lines that can be moved is determined by the size of the edit buffer.

Arguments

<range> Specifies the line numbers of the first and last lines to move. The "move_lines" command moves all lines between the specified lines, inclusive. The format for the range is

#<line_num_1>,<line_num_2>

The line numbers must appear sequentially--that is, the first number specified must be smaller than the second. Otherwise, the editor returns an error message.

EXAMPLES

1. TAB #100,150 control-O

This example moves lines 100 through 150, inclusive, from their original position in the file to a position between the current line and the line preceding it.

(continued)

ERROR MESSAGES

Invalid parameter
: The line number that specifies the beginning of the range must be less than the line number that specifies the end of the range.

Line no longer in edit buffer
: The line number specifying the beginning of the range is less than the number of the first line in the edit buffer. The user may access the line in question by invoking the "rewind" command and, if necessary, flushing text from the edit buffer to the disk until the line is in the buffer.

Line past end of buffer
: The line number specifying the end of the range is greater than the number of the last line in the edit buffer. If the file is completely loaded into the edit buffer, no such line exists. If the file is not completely loaded, the user may access the range of lines in question by using a form of the "flush" command that puts the entire range into the edit buffer.

Range overlaps cursor
: The range specified includes the current line.

SEE ALSO

copy_line
cut_line
flush
paste_line
rewind

move_to_file

Move the specified lines to a file.

SYNTAX

TAB <output_dir> [<file_name>] [TAB [<range>]] control-K

DESCRIPTION

The "move_to_file" command copies the specified lines to a file and deletes them from the file being edited. If the user does not specify a range (which must be preceded by a TAB character), it defaults to the contents of the entire edit buffer. If the file does not already exist, it is created with read and write permissions for all users.

Arguments

<file_name> — The name of the file to which to move the lines. The editor stores this name in a special buffer called the write-name buffer. Each time the user specifies the name of a file to the "move_to_file" command, it deletes the contents of the buffer and replaces it with the newly specified name. If the user does not specify a name, "move_to_file" uses whatever name is in the buffer. If the buffer is empty, the editor returns an error message. The write-name buffer is shared with the "copy_to_file" command.

<output_dir> — Specifies whether or not the editor should delete the contents of the file before moving text to it. The two valid arguments are the UniFLEX symbols for redirecting output: a single right-hand angle bracket, '>', specifies that any text already in the file should be deleted; two right-hand angle brackets, ">>", specify that the editor should append the text to any text already in the file.

<range> — Specifies which lines to move. The range may be a number, in which case it indicates how many lines to move, starting with the current line. The default is 1. If the number specified is larger than the number of lines between the current line and the last line in the edit buffer, the editor returns an error message.

The range may also specify the line numbers of

(continued)

the first and last lines to move. The editor moves all lines between the specified lines, inclusive. In such a case the format for the range is

#<line_num_1>,<line_num_2>

The line numbers must appear sequentially--that is, the first number specified must be smaller than the second. Otherwise, the editor returns an error message.

Finally, the user can express the extent of the range by using vertical cursor-positioning (see Section 3.6.2.2).

EXAMPLES

1. TAB >transfer TAB 10 control-K
2. TAB >>transfer TAB #100,150 control-K
3. TAB >/usr/jeremy/test TAB #100,150 control-K

The first example copies the current line and the following nine lines to the file named "transfer" in the working directory and deletes them from the file being edited. If the file does not already exist, the "move_to_file" command creates it; if it does exist, "move_to_file" deletes the contents before moving any text to it.

The second example appends lines 100 through 150, inclusive, of the file being edited to the file named "transfer" in the working directory and deletes them from the file being edited. If the file does not already exist, the "move_to_file" command creates it.

The third example copies lines 100 through 150, inclusive, of the file being edited to the file "/usr/jeremy/test" and deletes them from the file being edited. If the file does not already exist, the "move_to_file" command creates it; if it does exist, "move_to_file" deletes the contents before moving to it.

ERROR MESSAGES

Invalid parameter
If the user specifies the range with a number, it must be a positive integer. If the user chooses to specify two line numbers, the smaller number must precede the larger.

(continued)

Line past end of buffer
The line number specifying the end of the range is greater than the number of the last line in the edit buffer. If the file is completely loaded in the edit buffer, no such line exists. If the line does exist but has not yet been loaded into memory, the user can access it by using a form of the "flush" command.

No file name specified
The user tried to invoke the "move_to_file" command without specifying the name of a file, but the write-name buffer is empty.

SEE ALSO

copy_to_file
flush
read_file

page_dwn

Scroll the file down by the screenful.

SYNTAX

TAB [<num>] control-R

DESCRIPTION

The "page_dwn" command scrolls the file down a screenful at a time without changing the position of the cursor relative to the boundaries of the edit window. It functions as if the file were a scroll beneath the edit window and the user were pulling on the bottom of the scroll. Thus, after execution of the command the text in the edit window is nearer the beginning of the file than the previous text was. The cursor is stationary.

If execution of the "page_dwn" command would cause the first line of text in the edit buffer to scroll below the first line of the edit window, the editor positions the first line at the top of the window. Subsequent execution of the command has no effect.

Arguments

<num> The number of screens to scroll through. The number specified must be a positive integer.

EXAMPLES

1. TAB control-R
2. TAB 4 control-R

The first example scrolls the text down one screenful. Thus, if line 22 is at the top of the window when the user invokes the command and the screen contains the usual twenty-one lines, line 1 appears at the top of the edit window.

The second example scrolls the text down four screens of text. Thus, if line 85 is at the top of the window when the user invokes the command and the screen contains the usual twenty-one lines, line 1 appears at the top of the edit window.

(continued)

ERROR MESSAGES

Invalid parameter
The number specified must be a positive integer.

SEE ALSO

page_up
sdwn
sup

page_up

Scroll the file up by the screenful.

SYNTAX

TAB [<num>] control-V

DESCRIPTION

The "page_up" command scrolls the file up a screenful at a time without changing the position of the cursor relative to the boundaries of the edit window. It functions as if the file were a scroll beneath the edit window and the user were pulling on the top of the scroll. Thus, after execution of the command the text in the edit window is nearer the end of the file than the previous text was. The cursor is stationary.

If execution of the "page_up" command causes the end of the file or the last line of text in the edit buffer to scroll past the window, the editor positions the text so that the first line in the window is the line immediately following the last line. Subsequent execution of the command has no effect.

Arguments

<num> The number of screens to scroll through. The number specified must be a positive integer.

EXAMPLES

1. TAB control-V
2. TAB 4 control-V

The first example scrolls the text up one screenful. Thus, if line 1 is at the top of the window when the user invokes the command and the screen contains the usual twenty-one lines, line 22 appears at the top of the edit window.

The second example scrolls the text up four screens of text. Thus, if line 1 is at the top of the window when the user invokes the command and the screen contains the usual twenty-one lines, line 85 appears at the top of the edit window.

(continued)

ERROR MESSAGES

Invalid parameter
The number specified must be a positive integer.

SEE ALSO

page_dwn
sdwn
sup

paste

Retrieve text from the appropriate buffer and insert it into the file, or move a section of text from one place to another.

SYNTAX

```
TAB [<str>] control-P
[TAB <range>] control-O
```

Variations

move_lines	TAB <range> control-O
paste_line	control-O
paste_word	TAB [<str>] control-P

DESCRIPTION

The "paste" command retrieves text from the appropriate buffer and inserts it into the file, or moves a section of text from one place to another within the file without using the line buffer.

The "paste_word" command retrieves a string from a buffer called the word buffer. It inserts the string at the current column and moves to the right the cursor and all text at and to the right of it to accommodate the insertion. If the buffer is empty or if the line does not have enough room for the insertion, the editor returns an error message. If the user specifies an argument, "paste_word" first writes that string to the word buffer. Thus, even if the line does not have enough room for the insertion, "paste_word" writes the string to the buffer. The string can be composed of any printable characters. The maximum length of the string is restricted to one half of the maximum length of a line minus 6. Each time the user specifies a string, the "paste_word" command deletes the contents of the word buffer and enters the newly specified string. The "paste_word" command shares the word buffer with the "copy_word", "cut_word", and "replace" commands.

The "paste_line" command retrieves text from a buffer called the line buffer. It inserts the contents of the buffer between the current line and the line preceding it, then positions the cursor at the first line of the inserted text (without altering its horizontal position). If the buffer is empty, the editor returns an error message. The user may enter text into the buffer with either the "copy_line" or "cut_line" command.

The line buffer can hold no more lines than can fit in the edit window. Thus, the "copy_line" and "cut_line" commands are fairly restricted in the amount of text they can work with. A variation of the "paste" command, "move_lines", allows the user to avoid this limit when moving

(continued)

text (the limit cannot be avoided when copying text). The command moves the specified lines of text from their original place in the file to a new location between the current line and the line preceding it. It positions the cursor at the first line of text that was moved without altering the its horizontal position. No scrolling occurs. In order for the command to succeed the cursor must be outside the range of the lines being moved. The limit to the number of lines that can be moved is determined by the size of the edit buffer.

Arguments

<str> The string to store in the word buffer before pasting. If the user does not specify a string, "paste_word" uses the one currently in the word buffer.

<range> Specifies the line numbers of the first and last lines to move. The "move_lines" command moves all lines between the specified lines, inclusive. The format for the range is

#<line_num_1>,<line_num_2>

The line numbers must appear sequentially--that is, the first number specified must be smaller than the second. Otherwise, the editor returns an error message.

EXAMPLES

1. TAB testing control-P
2. TAB control-P
3. control-O
4. TAB #100,150 control-O

The first example stores the string "testing" in the word buffer and pastes that word into the file beginning at the current column, moving to the right the text at and to the right of the cursor to accommodate the insertion.

The second example retrieves the string that is in the word buffer and pastes it into the file beginning at the current column, moving to the right the text at and to the right of the cursor to accommodate the insertion.

The third example retrieves the contents of the line buffer and pastes it into the file between the current line and the line preceding it. It positions the cursor at the first line of the inserted text (without altering its horizontal position).

(continued)

The fourth example removes lines 100 through 150, inclusive, from their original position in the file and inserts them between the current line and the line preceding it. It positions the cursor at the first line of the inserted text (without altering its horizontal position).

ERROR MESSAGES

Invalid parameter
: The line number that specifies the beginning of the range of the "move_lines" command must be less than the line number that specifies the end of the range.

Line no longer in edit buffer
: The line number specifying the beginning of the range of the "move_lines" command is less than the number of the first line in the edit buffer. The user may access the line in question by invoking the "rewind" command and, if necessary, flushing text from the edit buffer to the disk until the line is in the buffer.

Line past end of buffer
: The line number specifying the end of the range is greater than the number of the last line in the edit buffer. If the file is completely loaded in the edit buffer, no such line exists. If the line does exist but has not yet been loaded into memory, the user can access it by using a form of the "flush" command.

No string in buffer
: The word buffer is empty. The user may enter text into the buffer with either the "copy_word" or "cut_word" command.

Not enough room in line
: The current line does not have enough room to accommodate the insertion of the contents of the word buffer by the "paste_word" command. If the user has specified a string to put into the buffer, that part of the command succeeds.

Nothing in the line buffer
: The line buffer is empty. The user may enter text into the buffer with either the "copy_line" or "cut_line" command.

Range overlaps cursor
: The range specified by the arguments to the "move_lines" command includes the current line.

SEE ALSO

copy
cut
flush
replace
rewind

paste_line

Retrieve text from the line buffer and insert it into the file.

SYNTAX

control-O

DESCRIPTION

The "paste_line" command retrieves text from a buffer called the line buffer. It inserts the contents of the buffer between the current line and the line preceding it, then positions the cursor at the first line of the inserted text without altering its horizontal position. If the buffer is empty, the editor returns an error message. The user may enter text into the buffer with either the "copy_line" or "cut_line" command.

ERROR MESSAGES

Nothing in the line buffer
: The line buffer is empty. The user may enter text into the buffer with either the "copy_line" or "cut_line" command.

SEE ALSO

copy_line
cut_line
paste

paste_word

Retrieve text from the word buffer and insert it into the current line.

SYNTAX

TAB [<str>] control-P

DESCRIPTION

The "paste_word" command retrieves a string from a buffer called the word buffer. It inserts the string at the current column and moves to the right the cursor and all text at and to the right of it to accommodate the insertion. If the buffer is empty or if the line does not have enough room for the insertion, the editor returns an error message. If the user specifies an argument, "paste_word" first writes that string to the word buffer. Thus, even if the line does not have enough room for the insertion, "paste_word" writes the string to the buffer. The string can be composed of any printable characters. The maximum length of the string is restricted to the half of the maximum length of a line minus 6. Each time the user specifies a string, the "paste_word" command deletes the contents of the word buffer and enters the newly specified string. The "paste_word" command shares the word buffer with the "copy_word", "cut_word", and "replace" commands.

Arguments

<str> The string to store in the word buffer before pasting. If the user does not specify a string, "paste_word" uses the one currently in the word buffer.

EXAMPLES

1. TAB testing control-P
2. TAB control-P

The first example stores the string "testing" in the word buffer and pastes that word into the file at the current column, moving to the right the cursor and all the text at and to the right of the cursor to accommodate the insertion.

The second example retrieves the string that is in the word buffer and pastes it into the file at the current column, moving to the right the cursor and all the text at and to the right of the cursor to accommodate the insertion.

(continued)

ERROR MESSAGES

No string in buffer
The word buffer is empty. The user may enter text into the buffer with either the "copy_word" or "cut_word" command.

Not enough room in line
The current line does not have enough room to accommodate the insertion of the contents of the word buffer. If the user has specified a string to put into the buffer, that part of the command succeeds.

SEE ALSO

copy_word
cut_word
replace

PP

Put the editor in paragraph mode.

SYNTAX

TAB p RETURN

DESCRIPTION

The "pp" command puts the editor in paragraph mode. In this mode of operation the editor automatically wraps the text from one line to the next, thereby maintaining a ragged right-hand margin. When the user types a character in the bell column, the editor sends a bell to the terminal and inserts a blank line between the current line and the line following it. It then searches the current line for the preceding space character and moves all characters to the right of it to the newly inserted line, placing the first character in the column that is directly underneath the first nonblank character in the preceding line.

This automatic processing of the end of the line means that a user does not need to type a carriage return when entering text. Of course, the user may still manually type a carriage return. In such a case the behavior of the editor depends on the location of the cursor with respect to the last column in the line that contains a character.

It is not always obvious which column contains the last character of a line. When the user enters any printable character (including the space character) in a line, the editor enters a space character in each column between that character and the column which previously contained the last character. If the user erases the last character, the character preceding it--which may be a space character inserted by the editor, not the user--becomes the last character in the line. However, when it saves the file, the editor truncates any trailing space characters (space characters between the last printed character in the line and the carriage return).

If the cursor is not at or beyond the last column in the line that contains a character when the user types a carriage return, the editor places it in the first column of the next line of text. If, however, the cursor is at or beyond the last column of the line, the editor inserts a blank line between the current line and the line following it, then moves the cursor to the column in that newly inserted line that is directly underneath the first nonblank character in the line preceding it.

If the user executes the "config" command, the editor sets a flag in the configuration file indicating whether or not it is in paragraph mode.

(continued)

The user can exit from paragraph mode by returning to the default mode with the following command:

TAB RETURN

or by entering one of the other available formatting modes.

MESSAGES

PPmode
The editor displays this message on the status line when it is in paragraph mode.

SEE ALSO

auto_ind
config
rjustify

read_file

Read lines from the specified file and insert them in the file being edited.

SYNTAX

TAB < [<file_name>] [TAB [<range>]] control-O

DESCRIPTION

The "read_file" command reads lines from a specified file and inserts them in the file being edited between the current line and the line preceding it. If the user does not specify a range (which must be preceded by a TAB character), it defaults to the entire file.

The editor adjusts the position of the text at and below the cursor to accommodate the insertion and positions the cursor at the first line of the inserted text without altering its horizontal position. No scrolling occurs. When the work is finished, "read_file" sends a message to the status line reporting how many lines of text it read.

Arguments

<file_name> The name of the file from which to read the lines. The editor stores this name in a special buffer called the read-name buffer. Each time the user specifies the name of a file to the "read_file" command, it deletes the contents of the buffer and replaces it with the newly specified name. If the user does not specify a name, "read_file" uses whatever name is in the buffer. If the buffer is empty, the editor returns an error message.

<range> Specifies which lines to read from the file being edited. The default is the entire file. The range may be a number, in which case it indicates the number of lines to read and copy. The default is 1. Unless the user specifies the name of the file, the editor starts reading at the line following the last line that it read from the file. Specifying the name of the file rewinds it so that the editor starts to read from the first line of the file. If the number specified is larger than the number of lines remaining in the file, the editor reads in the remainder of the file and sends a message to the status line reporting that it reached the end of the file.

(continued)

The range may also specify the line numbers of the first and last lines to read. The editor reads all lines between the specified lines, inclusive, and copies them to the file being edited. In such a case the format for the range is

#<line_num_1>,<line_num_2>

The line numbers must appear sequentially--that is, the first number specified must be smaller than the second. Otherwise, the editor returns an error message.

Finally, the user can designate a string as part of the range. The editor places the specified string in the search buffer (which is shared with the "search_fwd" and "search_bkw" commands), searches through the file until it finds the string, and begins reading at the line which contains it. The format for this version of a range is

[<num>]^[<str>]

where <num> is a positive integer specifying the number of lines to read into the file being edited, starting with the line containing the search string. The default value is 1. If the number is larger than the number of lines between the line containing the designated string and the end of the file, the editor reads in the remainder of the file and sends a message to the status line reporting that it reached the end of the file. If the user does not specify a search string, it defaults to whatever string is in the search buffer when the user invokes the command. The user may use the "display" command to view the contents of the search buffer. If the search buffer is empty, "read_file" starts reading at the line following the last line that it read from the file. If it does not find the search string, it reports that it read zero lines.

(continued)

EXAMPLES

1. TAB <transfer TAB #100,150 control-L
2. TAB < TAB 10 control-O
3. TAB </usr/jeremy/test TAB 15'EXAMPLES

The first example reads lines 100 through 150, inclusive, from the file named "transfer" in the working directory and copies those lines into the file being edited.

The second example reads the next ten lines from the file whose name is in the read-name buffer and copies them into the file being edited.

The third example reads through the file "/usr/jeremy/test" until it finds the string "EXAMPLES". It then copies that line and the following fourteen lines into the file being edited.

MESSAGES

Read <num> lines.
This message indicates the number of lines that the command read and copied to the file being edited.

ERROR MESSAGES

File not completely loaded
The edit buffer did not have enough room for the insertion of as much of the file as the user requested. The "read_file" command read and copied as much of the file as would fit in the available space.

Invalid parameter
If the user specifies the range with a number, it must be a positive integer. If the user chooses to specify two line numbers, the smaller number must precede the larger.

No file name specified.
The user tried to invoke the "read_file" command without specifying the name of a file, but the read-name buffer is empty.

Reached end of file.
The number of lines specified by the user was greater than the number of lines remaining in the file being read. The "read_file" command read and copied the remainder of the file to the file being edited.

(continued)

SEE ALSO

copy_to_file
display
move_to_file
search

replace

Replace the next occurrence of the string in the search buffer with the string in the word buffer.

SYNTAX

[TAB <str>] control-X

DESCRIPTION

The "replace" command first tests to see whether or not the cursor is positioned at the first character of the string in the search buffer (the search string). If it is, the command replaces that occurrence of the search string with the string in the word buffer. Otherwise, it searches forwards from the current column towards the end of the edit buffer for the next occurrence of the search string. If it finds the search string, the "replace" command replaces it with the string in the word buffer. If the next occurrence of the search string is in the edit window when the user invokes the command, the text remains stationary. Otherwise, the editor places the line containing the search string approximately one-third of the way down the edit window and positions the cursor at the column immediately following the last character of the replacement string.

If the line containing the search string is too long to accommodate the replacement or if either the word buffer or the search buffer is empty, the editor returns an error message.

If the user specifies an argument, the "replace" command puts that string in the word buffer before beginning to search for the search string. Each time the user specifies a string, "replace" deletes the contents of the word buffer and enters the newly specified string. The user may use the "display" command to view the contents of the word buffer, which is shared with the "cut_word", "copy_word", and "paste_word" commands.

Arguments

<str> — Specifies the string to place in the word buffer. It may be composed of any printable characters. The maximum length of the string is restricted to one-half of the maximum length of a line minus 6.

(continued)

EXAMPLES

1. TAB test_string control-X
2. TAB control-X

The first example puts the string "test_string" into the word buffer and tests to see whether or not the cursor is on the first character of the search string. If it is, it replaces the search string with "test_string". Otherwise, it searches forwards through the edit buffer for the next occurrence of the search string, which it replaces with "test_string".

The second example replaces the next occurrence (including an occurrence at the cursor) of the search string with whatever string is in the word buffer.

NOTES

- The "replace" command and the "search_fwd" command do not begin their searches at the same point in the edit buffer: "replace" begins at the current column; "search_fwd", at the column immediately following the current column. Thus, if the cursor is positioned at the first character of the search string, the "replace" command replaces that occurrence while the "search_fwd" command looks for the next one.

- The user may invoke the "replace" command in conjunction with the "global" command to change all occurrences of the search string in the edit buffer to the replacement string or to change all occurrences of the search string between the current column and to the end of the edit buffer, inclusive, to the replacement string.

MESSAGES

Search key not found
The "replace" command searched forwards from the current column to the end of the edit buffer and did not find the search string.

ERROR MESSAGES

No string in buffer
The user invoked the "replace" command without having placed a string in the word buffer at any time during the editing session.

Not enough room in line
The replacement string is too long to fit in the current line.

(continued)

Nothing to search for
The user invoked the "replace" command without having placed a string in the search buffer at any time during the editing session.

SEE ALSO

copy_word
cut_word
paste_word
search_fwd
search_rep

return

Position the cursor at the first column of the next line.

SYNTAX

control-M

DESCRIPTION

The "return" command positions the cursor at the first column of the next line of the edit buffer. If that line is already in the edit window, the "return" command simply moves the cursor to the beginning of the line. If, however, the cursor is at the bottom of the edit window, "return" scrolls the text up, positioning it so that the line the cursor moves to is approximately one third of the way down the edit window. This automatic scrolling is equivalent to the user's executing two "sup" commands followed by a "return". If the cursor is at the bottom of the edit buffer, the "return" command behaves normally, scrolling blank lines into the edit window, until the memory buffer is filled.

When the editor is in paragraph mode, right-justify mode, or automatic-indent mode, the "return" command functions slightly differently. Its behavior depends on the location of the cursor with respect to the column containing the last character in the line. It is not always obvious which column that is. When the user enters any printable character (including the space character) in a line, the editor enters a space character in each column between that character and the column which previously contained the last character. If the user erases the last character, the character preceding it--which may be a space character inserted by the editor, not the user--becomes the last character in the line. However, when it saves the file, the editor truncates any trailing space characters (space characters between the last printed character in the line and the carriage return).

If the cursor is not beyond the last character in the line when the user invokes the command, "return" behaves normally, positioning the cursor at the first column of the next line. If, however, the cursor is beyond the column containing the last character in the line when the user invokes the command, "return" inserts a blank line between the current line and the line following it, then positions the cursor at the column in that line that is directly underneath the first nonblank character in the preceding line.

(continued)

NOTES

. Most, if not all, terminals have a single key that executes the control-M sequence. It is usually labeled "return", "carriage return", or "enter".

SEE ALSO

auto_ind
pp
rjustify
sup

rewind

End the editing session and reinvoke the editor with the same file.

SYNTAX

TAB [<save_instr>] < control-U

DESCRIPTION

The "rewind" command exits from the current editing session and automatically reinvokes the editor with the same file. Essentially, it is the same as exiting from the edit session and invoking the editor again. The two crucial differences are that the editor remains configured as it was when the user invoked the "rewind" command and the contents of all buffers except the read-name buffer remain intact from one session to the next.

Arguments

<save_instr> Instructs the editor how to handle both the copy of the file in memory and the backup file. The valid arguments are 'q', 'd', and 'y'.

A 'q' tells the "rewind" command not to save the edited version of the file. Both the original file and any existing backup file remain intact.

A 'd' tells the editor to retain the contents of a previously existing backup file and to discard the contents of the original file. The temporary file is then given the name of the original file. If no backup file exists, the 'd' argument tells the editor not to create one.

A 'y' tells the editor to replace the contents of a previously existing backup file with the contents of the original file. The temporary file is then given the name of the original file. If no backup file exists, the 'y' argument has no affect on the behavior of the editor.

EXAMPLES

1. TAB <control-U
2. TAB q <control-U

(continued)

The first example writes the contents of the edit buffer to the temporary file on the disk and begins another editing session on the same file. If necessary, it creates a backup file. If a backup file already exists, it asks for permission to delete it.

The second example exits from the editing session without writing any of the changes made during the session to the disk and deletes the temporary file. It then begins an edit session on the same file as the previous editing session.

MESSAGES

Delete existing backup file?
The user has ended the editing session without giving instructions about an existing backup file. In response to this prompt the user should type a 'y' for "yes" or an 'n' for "no".

SEE ALSO

config
exit

right

Position the cursor at the next column.

SYNTAX

control-F

DESCRIPTION

The "right" command moves the cursor to the next column in the edit buffer. If the cursor is in the last column of a primary window, the "right" command moves it to the next column and switches the display to a secondary window (see Section 3.3.1). If the cursor is in column 128, the "right" command moves it to the first column of the next line and changes the display to a primary window. If the cursor is in the last column of the last line of text in the edit buffer, the "right" command behaves normally, scrolling a blank line into the bottom of the edit window, until the memory buffer is filled. At that point execution of the command has no effect.

rjustify

Put the editor in right-justify mode.

SYNTAX

TAB r RETURN

DESCRIPTION

The "rjustify" command puts the editor in right-justify mode. In this mode of operation the editor automatically right-justifies the text. When the user types a character in the bell column, the editor sends a bell to the terminal and inserts a blank line just below the current line. It then searches the current line for the preceding space character and moves all characters to the right of it to the newly inserted line, placing the first character in the column that is directly underneath the first nonblank character in the preceding line. In addition, it right-justifies the text of the completed line by inserting enough space characters between words to bring the last character of the line into the column immediately preceding the bell column. The process of justification never alters the number of leading blanks on a line, which are assumed to be intentional indentation.

This automatic processing of the end of the line means that a user does not need to type a carriage return when entering text. Of course, the user may still manually type a carriage return. In such a case the behavior of the editor depends on the location of the cursor with respect to the last column in the line that contains a character.

It is not always obvious which column contains the last character of a line. When the user enters any printable character (including the space character) in a line, the editor enters a space character in each column between that character and the column which previously contained the last character. If the user erases the last character, the character preceding it--which may be a space character inserted by the editor, not the user--becomes the last character in the line. However, when it saves the file, the editor truncates any trailing space characters (space characters between the last printed character in the line and the carriage return).

If the cursor is not at or beyond the last column in the line that contains a character when the user types a carriage return, the editor places it in the first column of the next line of text. If, however, the cursor is at or beyond the last column of the line, the editor inserts a blank line just below the current line and moves the cursor to the column in that newly inserted line that is directly underneath the first nonblank character in the line preceding it.

(continued)

If the user executes the "config" command, the editor sets a flag in the configuration file indicating whether or not it is in right-justify mode.

The user can exit from right-justify mode by returning to the default mode with the following command:

TAB RETURN

or by entering one of the other available formatting modes.

MESSAGES

RJmode
The editor displays this message on the status line when it is in right-justify mode.

SEE ALSO

auto_ind
config
pp

rtab

Move the cursor to the right until it reaches a column that contains a tab stop.

SYNTAX

control-G

DESCRIPTION

The "rtab" command moves the cursor to the right until it reaches a column that contains a tab stop. If the cursor is in the last column of the primary window, the "rtab" command moves it to the next tab stop and switches the display to a secondary window (see Section 3.3.1). If the cursor is in column 128, the "rtab" command has no effect.

By default, tab stops occur every eighth column starting in column 1. The first and last columns of the edit window always contain tab stops. The user may set and clear tab stops with the "set_tab" and the "clr_tab" commands.

If the editor is in word-tab mode, the "rtab" command moves the cursor to the right until it reaches the beginning of a word. By default, a word is defined as a string of alphanumeric characters. The user can, however, invoke the "word_def" command to add to the set of characters that the editor recognizes as part of a word.

SEE ALSO

clr_tab
ltab
set_tab
word_def
word_tab

save

End the editing session and save the edited version of the file.

SYNTAX

TAB [<save_instr>] [<[<file_name>]] control-U

DESCRIPTION

The "save" command ends the editing session and saves the edited version of the file.

When the editor is invoked with a single argument, it copies the file being edited into memory and creates a temporary file in the same directory as the file being edited. The temporary file is used during the editing session if the user fills the edit buffer and at the end of the editing session if the user invokes the "save" command. Any changes the user makes during the editing session are made on the copy in memory.

When a user ends the editing session with the "save" command, it copies the file as it exists in memory to the temporary file created by the editor at the beginning of the editing session. By default, if no backup file exists, the editor renames the original file by appending the characters ".bak" to its name. If this addition would result in a longer name than the operating system allows (fourteen characters for the 6809; fifty-five, for the 68xxx), the editor truncates the original name before adding the suffix. It then renames the temporary file, giving it the name of the original file.

If a file named <file_name.bak> already exists, the editor asks for permission to delete that file. In response to this question the user must type either 'y' for "yes" or 'n' for "no". In response to a 'y', the editor deletes the file whose name is <file_name.bak>, gives <file_name> the name <file_name.bak>, and names the temporary file <file_name>. In response to an 'n', the editor deletes the contents of the original file, renames the temporary file <file_name>, and leaves the file <file_name.bak> untouched. The editor accepts no characters other than a 'y' or an 'n' at this point. It is not necessary to type a carriage return after the response.

Arguments

<file_name> The name of the file with which to begin the next editing session. If the user begins an editing session this way, the editor maintains whatever configuration it was in when the user invoked the "save" command (see Section 3.7).

(continued)

In addition, the contents of the word, line, search, and word-definition buffers remain intact from one session to the next.

<save_instr> Instructs the editor how to handle the backup file. The valid arguments are 'd' and 'y'.

A 'y' tells the editor to replace the contents of a previously existing backup file with the contents of the original file. The temporary file is then given the name of the original file. If no backup file exists, the 'y' argument has no affect on the default behavior of the editor.

A 'd' tells the editor to retain the contents of a previously existing backup file and to discard the contents of the original file. The temporary file is then given the name of the original file. If no backup file exists, the 'd' argument tells the editor not to create one.

EXAMPLES

1. TAB control-U
2. TAB y <new_file control-U
3. TAB d control-U

The first example ends the editing session and writes the contents of the edit buffer to the temporary file on the disk. If necessary, it creates a backup file. If a backup file already exists, it asks for permission to delete it.

The second example exits from the editing session, deleting any existing backup file, so that the original file replaces the backup file and the copy from memory replaces the original file. It then begins an editing session on the file named "new_file".

The third example ends the editing session and writes the contents of the edit buffer to the temporary file on the disk. If there is no backup file, it does not create one. If a backup file already exists, the editor does not delete it. It deletes the contents of the original file and gives that name to the temporary file.

MESSAGES

Delete existing backup file?
The user has ended the editing session without giving instructions about an existing backup file. In response to this prompt the user should type a 'y' for "yes" or an 'n' for "no".

SEE ALSO

exit

sdwn

Scroll the text down the specified number of lines.

SYNTAX

[TAB <num>] control-Y

DESCRIPTION

The "sdwn" command scrolls the text down the specified number of lines. It functions as if the file were a scroll beneath the edit window and the user were pulling on the bottom of the scroll. Thus, after execution of the command the text in the edit window is nearer the beginning of the file than the previous text was.

If the current line remains on the screen after the execution of the "sdwn" command, the cursor remains on that line. If, on the other hand, execution of the "sdwn" command forces the current line off the screen, the editor places the cursor on the line at the bottom of the edit window. The horizontal position of the cursor does not change.

If execution of the "sdwn" command would cause the first line of text in the edit buffer to scroll below the first line of the edit window, the editor positions the first line at the top of the window. Subsequent execution of the command has no effect.

Arguments

<num> The number of lines to scroll. The number must be a positive integer.

EXAMPLES

1. control-Y
2. TAB 10 control-Y

The first example scrolls the screen down one-third of a screenful. If the screen contains twenty-one lines and line 22 is at the top of the edit window, execution of the command puts line 15 at the top of the window and line 35 at the bottom. If the cursor was on any line between line 22 and line 35, it remains on that line. Otherwise, the editor positions it at line 35.

The second example scrolls the screen up ten lines. If the screen contains twenty-one lines and line 22 is at the top of the edit window, execution of the command puts line 12 at the top of the window and line 32 at the bottom. If the cursor was on any line between line 22 and

(continued)

line 32, it remains on that line. Otherwise, the editor positions it at line 32.

ERROR MESSAGES

Invalid parameter.
The number specified must be a positive integer.

SEE ALSO

page_dwn
page_up
sup

search

Search forwards or backwards through the edit buffer for the next or previous occurrence of the specified string.

SYNTAX

```
[TAB <search_str>] control-Q
[TAB <search_str>] control-Z
```

Variations

```
search_bkw   [TAB <search_str>] control-Q
search_fwd   [TAB <search_str>] control-Z
```

DESCRIPTION

The "search" command searches forwards or backwards from the character immediately following or immediately preceding the cursor towards the end or beginning of the edit buffer for the string that is currently in the search buffer (the search string). If the search buffer is empty, the editor returns an error message.

If the user specifies an argument, the "search" command puts that string in the search buffer before beginning the search. Each time the user specifies a string, "search" deletes the contents of the search buffer and enters the newly specified string. The search buffer is shared with the "read_file" command.

If the "search" command finds the search string, it positions the cursor at the first character of that string. If that occurrence of the string is in the edit window when the user invokes the command, the text remains stationary. If it is not in the edit window and is in the first few lines of the edit buffer, the editor places the first line of the edit buffer at the top of the edit window. Otherwise, the editor places the line containing the string approximately one-third of the way down the edit window.

Arguments

<search_str> Specifies the string to place in the search buffer. It may be composed of any printable characters. The maximum length of the string is restricted to one-half of the maximum length of a line minus 6. The default is the string currently in the search buffer.

(continued)

EXAMPLES

1. TAB test_string control-Z
2. control-Z
3. control-Q
4. TAB new_string control-Q

The first example places the string "test_string" in the search buffer and searches forwards from the character immediately following the cursor towards the end of the edit buffer for the next occurrence of that string.

The second example searches forwards from the character immediately following the cursor towards the end of the edit buffer for the next occurrence of whatever string is in the search buffer.

The third example searches backwards from the character immediately preceding the cursor towards the beginning of the edit buffer for the previous occurrence of whatever string is in the search buffer.

The fourth example places the string "new_string" in the search buffer and searches backwards from the character immediately preceding the cursor towards the beginning of the edit buffer for the previous occurrence of that string.

MESSAGES

Search key not found
The "search" command searched from the character immediately following or immediately preceding the cursor to the end or beginning of the edit buffer (depending on which variation of the command the user invoked) and did not find the search string.

ERROR MESSAGES

Nothing to search for
The user invoked the "search" command without having specified a search string at any time during the editing session.

SEE ALSO

read_file
replace
search_rep

search_bkw

Search backwards through the edit buffer for the previous occurrence of the specified string.

SYNTAX

[TAB <search_str>] control-Q

DESCRIPTION

The "search_bkw" command searches backwards from the character immediately preceding the cursor towards the beginning of the edit buffer for the previous occurrence of the string that is currently in the search buffer (the search string). If the search buffer is empty, the editor returns an error message.

If the user specifies an argument, the "search_bkw" command puts that string in the search buffer before beginning the search. Each time the user specifies a string, "search_bkw" deletes the contents of the search buffer and enters the newly specified string. The search buffer is shared with the "search_fwd" and the "read_file" commands.

If the "search_bkw" command finds the search string, it positions the cursor at the first character of that string. If that occurrence of the string is in the edit window when the user invokes the command, the text remains stationary. If it is not in the edit window and is in the first few lines of the edit buffer, the editor places the first line of the edit buffer at the top of the edit window. Otherwise, the editor places the line containing the string approximately one-third of the way down the edit window.

Arguments

<search_str> Specifies the string to place in the search buffer. It may be composed of any printable characters. The maximum length of the string is restricted to one-half of the maximum length of a line minus 6. The default is the string currently in the search buffer.

EXAMPLES

1. control-Q
2. TAB new_string control-Q

(continued)

The first example searches backwards from the character immediately preceding the cursor towards the beginning of the edit buffer for the previous occurrence of whatever string is in the search buffer.

The second example places the string "new_string" in the search buffer and searches backwards from the character immediately preceding the cursor towards the beginning of the edit buffer for the previous occurrence of that string.

MESSAGES

Search key not found
: The "search_bkw" command searched from the character immediately preceding the cursor to to the beginning of the edit buffer and did not find the search string.

ERROR MESSAGES

Nothing to search for
: The user invoked the "search_bkw" command without having specified a search string at any time during the editing session.

SEE ALSO

read_file
replace
search_fwd

search_fwd

Search forwards through the edit buffer for the next occurrence of the specified string.

SYNTAX

[TAB <search_str>] control-Z

DESCRIPTION

The "search_fwd" command searches forwards from the character immediately following the cursor towards the end of the edit buffer for the next occurrence of the string that is currently in the search buffer (the search string). If the search buffer is empty, the editor returns an error message.

If the user specifies an argument, the "search_fwd" command puts that string in the search buffer before beginning the search. Each time the user specifies a string, "search_fwd" deletes the contents of the search buffer and enters the newly specified string. The search buffer is shared with the "search_bkw" and the "read_file" commands.

If the "search_fwd" command finds the search string, it positions the cursor at the first character of that string. If that occurrence of the string is in the edit window when the user invokes the command, the text remains stationary. Otherwise, the editor places the line containing the string approximately one-third of the way down the edit window.

Arguments

<search_str> Specifies the string to place in the search buffer. It may be composed of any printable characters. The maximum length of the string is restricted to one-half of the maximum length of a line minus 6. The default is the string currently in the search buffer.

EXAMPLES

1. TAB test_string control-Z
2. control-Z

The first example places the string "test_string" in the search buffer and searches forwards from the character immediately following the cursor towards the end of the edit buffer for the next occurrence of the search string.

(continued)

The second example searches forwards from the character immediately following the cursor towards the end of the edit buffer for the next occurrence of whatever string is in the search buffer.

MESSAGES

Search key not found
: The "search_fwd" command searched from the character immediately following the cursor to the end of the edit buffer and did not find the search string.

ERROR MESSAGES

Nothing to search for
: The user invoked the "search_fwd" command without having specified a string string at any time during the editing session.

SEE ALSO

read_file
replace
search_bkw
search_rep

search_rep

Put one string in the search buffer, another in the word buffer, and replace the next occurrence of the search string with the string in the word buffer.

SYNTAX

TAB <search_str> TAB <replace_str> control-X

DESCRIPTION

The "search_rep" command puts one string in the search buffer and another in the word buffer. If the cursor is on the first character of the search string when the user invokes the command, it replaces that occurrence of the search string with the string in the word buffer. Otherwise, "search_rep" searches forwards through the edit buffer for the next occurrence of the search string and tries to replace it with the string in the word buffer. If the line containing the search string is too long to accommodate the replacement, the editor returns an error message.

If the next occurrence of the search string is in the edit window when the user invokes the command, the text remains stationary. Otherwise, the editor places the line that contained the search string approximately one-third of the way down the edit window. It positions the cursor at the column immediately following the last character of the replacement string.

Arguments

<replace_str> Specifies the string to place in the word buffer. It may be composed of any printable characters. The maximum length of the string is restricted to one-half of the maximum length of a line minus 6.

<search_str> Specifies the string to place in the search buffer. It may be composed of any printable characters. The maximum length of the string is restricted to one-half of the maximum length of a line minus 6.

EXAMPLES

1. TAB 100 TAB one hundred control-X

(continued)

This example puts the string "100" in the search buffer and the string "one hundred" in the word buffer. If the cursor is on the first character of the string "100" when the user invokes the command, "search_rep" replaces that occurrence of the search string with "one hundred". Otherwise, it searches forwards through the edit buffer for the next occurrence of the search string (100) and replaces it with the string in the word buffer (one hundred).

NOTES

- The user may invoke the "search_rep" command in conjunction with the "global" command to change all occurrences of the search string in the edit buffer to the replacement string or to change all occurrences of the search string from the current column to the end of the edit buffer to the replacement string.

- The "search_rep" command and the "search_fwd" command do not begin their searches at the same point in the edit buffer: "search_rep" begins at the current column; "search_fwd", at the column immediately following the current column. Thus, if the cursor is positioned at the first character of the search string, the "search_rep" command replaces that occurrence while the "search_fwd" command looks for the next one.

MESSAGES

Search key not found
  The "search_rep" command searched from the current column to the end of the edit buffer and did not find the search string.

ERROR MESSAGES

Not enough room in line
  The replacement string is too long to fit in the current line.

SEE ALSO

global
replace
search_fwd

set_bell

Set the bell column to the current column.

SYNTAX

TAB s control-G

DESCRIPTION

By default, when a user invokes the editor, it sets the bell column in column 65. The "set_bell" command sets the bell column at the current column. If another column has already been designated as the bell column, the editor automatically clears the bell from that column before setting the new bell column as it can honor only one bell column at a time. Normally the editor places a vertical bar, '|', in the top margin of the edit window at the bell column. If the bell column also contains a tab stop, it uses a pound sign, '#', instead.

When a user types a character in the bell column, the editor sends a bell (control-G) to the screen. If the editor is in its default mode or in automatic-indent mode, the user can type past the bell column; it simply serves as an audible reminder that the cursor has reached a certain position. If the editor is in paragraph mode, it wraps the text to the next line when the user types a character in the bell column. In right-justify mode it both wraps and right-justifies the text. A useful combination, which allows the user to enter one word per line, is to put the editor in one of these two modes and to set the bell column in the first column.

The editor preserves the location of the bell column in the configuration file when the user executes the "config" command.

NOTES

- The "column" command is helpful for positioning the cursor at the desired column before executing the "set_bell" command.

SEE ALSO

auto_ind
clr_bell
column
config
pp
rjustify

set_tab

Set tab stops as specified.

SYNTAX

TAB <interval_size> control-A
TAB s control-A

DESCRIPTION

By default, the editor sets tab stops every eighth column starting in column 1 (i.e., in columns 1, 9, 17, 25, etc.). The "set_tab" command allows the user to change the location of the tab stops. The invariant form of the command sets a tab stop in the current column without affecting any previously defined tab stops. If used with a numerical argument, the command first clears all previously designated tab stops, then sets tab stops at regular intervals, beginning in column 1. The first and last columns of the edit window always contain tab stops.

Normally the editor indicates the presence of a tab stop in a column by putting a plus sign, '+', in that column in the top margin of the edit window. If that column is also the bell column, it uses a pound sign, '#', instead.

The editor preserves the location of all tab stops in the configuration file when the user executes the "config" command.

Arguments

<interval_size> Specifies the interval at which to set the tab stops. The number used must be between 0 and 127. An argument of 0 removes all tab stops except the ones in the first and last columns. If the number specified is not an integer, the editor truncates it before using it.

EXAMPLES

1. TAB 4 control-A
2. TAB s control-A

The first example clears all previously designated tab stops, then sets a tab stop in every fourth column, starting in column 1 (i.e., in columns 5, 9, 13, 17, etc.).

(continued)

The second example sets a tab stop in the current column.

NOTES

- The "column" command is helpful for positioning the cursor at the desired column before executing the "set_tab" command.

SEE ALSO

clr_tab
column
config
ltab
rtab

shift_left

Shift text to the left by deleting characters in the block of text delimited by the movement of the cursor.

SYNTAX

TAB <block> control-B

DESCRIPTION

The "shift_left" command shifts text to the left by deleting characters in the block of text delimited by the movement of the cursor. This command accepts only cursor-defined arguments.

Arguments

<block> The range of the "shift_left" command is delimited by either horizontal or both horizontal and vertical cursor-positioning (see Section 3.6.2.2). The command deletes the block of text described by the movement of the cursor. The text affected includes all columns from the original position of the cursor up to but not including the column containing the cursor when the user types control-B. It includes all lines from the original position of the cursor to the line containing the cursor when the user types control-B, inclusive.

NOTES

- The user can create a powerful tool by using the "shift_left" command when the editor is in word-tab mode. This combination makes it easy to excise a group of words from a line of text by using the "rtab" and "ltab" commands to perform the horizontal cursor-positioning.

SEE ALSO

consume
delete
word_tab

shift_right

Shift text to the right by inserting space characters in the block of text delimited by movement of the cursor.

SYNTAX

TAB <block> control-W

DESCRIPTION

The "shift_right" command shifts text to the right by inserting space characters in the block of text delimited by movement of the cursor. This command accepts only cursor-defined arguments (see Section 3.6.2.2).

Arguments

<block> The range of the "shift_right" command is delimited by either horizontal or both horizontal and vertical cursor-positioning (see Section 3.6.2.2). The number of space characters inserted on each line is equal to the number of columns covered by the cursor from its original position up to but not including the final column. The command alters all lines between the original and final positions of the cursor, inclusive.

SEE ALSO

insert

split

Split a line of text into two lines.

SYNTAX

TAB control-W

DESCRIPTION

The "split" command splits a line of text into two lines by inserting a blank line between the current line and the line following it and moving all text at and to the right of the cursor to the newly inserted line.

SEE ALSO

insert

sup

Scroll the text up the specified number of lines.

SYNTAX

[TAB [<num>]] control-N

Variations

sup	[TAB <num>] control-N
home_line	TAB control-N

DESCRIPTION

The "sup" command scrolls the text up the specified number of lines. It functions as if the file were a scroll beneath the edit window and the user were pulling on the top of the scroll. Thus, after execution of the command the text in the edit window is nearer the end of the file than the previous text was.

If the current line remains on the screen after the execution of the "sup" command, the cursor remains on that line of text. If, on the other hand, execution of the "sup" command forces the current line off the screen, the editor positions the cursor at the line at the top of the edit window. The horizontal position of the cursor does not change.

If execution of the "sup" command causes the end of the file or the last line of text in the edit buffer to scroll past the window, the editor positions the text so that the first line in the window is the line immediately following that line. Subsequent execution of the command has no effect.

The "home_line" positions the current line at the top of the edit window.

Arguments

<num> The number of lines to scroll. The number must be a positive integer.

EXAMPLES

1. control-N
2. TAB 10 control-N
3. TAB control-N

(continued)

The first example scrolls the screen up one-third of a screenful. If the screen contains twenty-one lines and line 1 is at the top of the edit window, execution of the command puts line 8 at the top of the edit window.

The second example scrolls the screen up ten lines. If the screen contains twenty-one lines and line 1 is at the top of the edit window, execution of the command puts line 11 at the top of the edit window. If the cursor was on any line between line 11 and line 21, inclusive, it remains on that line. Otherwise, the editor positions it at line 11.

The third example positions the current line at the top of the edit window.

ERROR MESSAGES

Invalid parameter.
The number specified must be a positive integer.

SEE ALSO

page_dwn
page_up
sdwn

temp_exit

Give the user access to the shell program without ending the editing session.

SYNTAX

TAB +[<command_list>] control-U

DESCRIPTION

The "temp_exit" command gives the user access to the shell program without actually ending the editing session. By default, the command calls another shell program in an interactive mode. Logging out of the shell (by typing "log" or control-D) returns control to the editor.

The user may also specify a list of commands to be executed when invoking the "temp_exit" command. In such a case, the editor calls a shell program which, on completion of the commands, automatically returns control to the editor. If a task terminates abnormally, the editor displays a message to that effect on the status line when it regains control. The message only reports that the task failed; it does not give a reason.

It is possible to run a command in background by terminating it with an ampersand, '&'. However, a user who chooses to do so should redirect both standard error and standard output (which requires the "nshell" command in Utilities Package II for the 6809), so that the task does not send information to the screen, thereby disturbing the editor's display. (The editor is careful to rewrite only that part of the screen which it knows has been altered. It is not possible to tell the editor to rewrite the entire screen without scrolling the text.) Running a task is the background is somewhat risky as the user cannot determine if the task has terminated or if it terminated abnormally.

Arguments

<command_list> A list of UniFLEX commands. In a list of more than one command, every command except the last one must be followed by a semicolon, ';'.

EXAMPLES

1. TAB + control-U
2. TAB + more other_file
3. TAB + pascal test.p >%>output& control-U

(continued)

The first example invokes a shell program interactively. In response to the UniFLEX prompt the user may enter any command. Control returns to the editor when the user logs out of the shell.

The second example invokes a shell program which executes the command "more other_file". At the completion of the "more" command, control automatically returns to the editor.

The third example invokes a shell program which executes the command "pascal test.p" in the background. Standard error and standard output are both redirected to a file named "output". When the shell program sends the task to the background, control returns to the editor.

SEE ALSO

exit

top

Position the cursor at the first line of the edit buffer.

SYNTAX

control-T

DESCRIPTION

The "top" command positions the cursor at the first column of the first line of the edit buffer and places that line at the top of the edit window.

up

Move the cursor to the previous line of the edit buffer.

SYNTAX

control-E

DESCRIPTION

The "up" command moves the cursor to the previous line in the edit buffer without altering its horizontal position. If that line of text is already in the edit window, the editor simply moves the cursor up one line. If, however, the cursor is at the top line of the edit window, the editor scrolls the text down one line so that the previous line of text appears at the top of the window. In this case the cursor actually remains stationary while the text scrolls past it. If the cursor is on the first line of the edit buffer, the "up" command has no effect.

word_def

Include the specified characters in the set of characters considered to be part of a word.

SYNTAX

TAB wd <char_list> control-A

DESCRIPTION

By default, the editor considers that a word consists entirely of alphanumeric characters. All other printable characters are considered to be punctuation. The "word_def" command allows the user to add to the basic set of characters that the editor recognizes as part of a word.

The editor maintains a buffer called the word-definition buffer that contains all the punctuation characters that the user wishes to include in the set of characters recognized as part of a word. Each time the user invokes the "word_def" command, it rewrites the entire buffer. Thus, the user cannot add characters to or delete them from the buffer without specifying the contents of the buffer as a whole. The editor preserves the contents of the word-definition buffer in the configuration file when the user executes the "config" command.

The user may use the "display" command to view the contents of the word-definition buffer.

Arguments

<char_list> The list of characters to put in the word-definition buffer so that the editor will recognize them as part of a word. The user should be careful not to accidentally include the space character in this list.

EXAMPLES

1. TAB ws-_/

This example adds the hyphen, underscore, and slash characters to the list of characters that the editor recognizes as part of a word.

(continued)

SEE ALSO

config
copy_word
cut_word
display
word_tab

word_tab

Change the mode of operation of the editor from the default mode to the mode in which the tab functions recognize the beginning of a word rather than a tab stop, or change the mode back to the default.

SYNTAX

TAB w control-A

DESCRIPTION

The "word_tab" command changes the mode of operation of the editor from the default mode to the mode in which the "ltab" and "rtab" commands recognize the beginning of a word rather than a tab stop. If the editor is already in that mode, "word_tab" puts it in the default mode. If the user executes the "config" command, the editor sets a flag in the configuration file indicating whether or not it is in word-tab mode.

By default, the editor considers that a word is composed entirely of alphanumeric characters. It considers all other printable characters to be punctuation. The user can invoke the "word_def" command to add characters to the set that the editor recognizes as part of a word.

MESSAGES

Word Tab
: The editor displays this message on the status line when it is in word-tab mode.

SEE ALSO

config
ltab
rtab
word_def

Appendix A

Syntax Conventions

The following conventions are used in syntax statements throughout this manual.

1. The commands supported by the editor are based on "control characters", that is, they are defined by keys which produce nonprintable characters. Many control characters are not represented by a single key on the keyboard. Rather, the user enters such a character by depressing the key marked "control" (or "ctrl") and holding it down while typing another character. In this document such a control sequence is referred to by the string

   control-<char>

   where <char> is the name of the key to press while holding down the control key. Alphabetic control characters are shown with uppercase letters because that is what is usually on the keyboard. The user may, however, type either the upper- or lowercase letter.

2. A string entirely composed of uppercase letters represents the name of a control key. The user should type the key that corresponds to that name rather than the name itself. For instance, the string "TAB" means to type whatever keys are necessary to generate the tab function. Depending on the terminal, the user may be able to type a single key marked "TAB" or may have to type the sequence control-I.

3. Many of the syntax statements in this manual contain space characters in order to make them easier to read. The user, however, should not type any space characters between the parts of a command.

4. Characters other than spaces and strings representing control characters that are not enclosed in angle brackets, '<' and '>', or square brackets, '[' and ']', are "keywords" and should be typed as shown.

5. Angle brackets, '<' and '>', enclose descriptions which the user must replace with a specific argument. Expressions enclosed only in angle brackets are essential parts of the command line. For example, in the command

   TAB <line_num> control-T

   the number of a line must be specified in the place indicated by <line_num>.

6. Square brackets, '[' and ']', indicate optional items. These items may be omitted if their effect is not desired.

7. The underscore character, '_', is used to link separate words that describe one term, such as "line" and "num".

8. If the word "list" appears as part of a term in a syntax statement, that term consists of one or more of the elements described by the rest of the term. For example, the term

   <command_list>

   represents a list of UniFLEX commands.

Appendix B

Command Summaries for the UniFLEX Screen Editor

abort	End the editing session without saving any of the changes made to the file.
align	Force lines to fit within the bell column.
app	Position the cursor at the column following the column containing the last character in the line or at the first column of the line.
auto_ind	Put the editor in automatic-indent mode.
backspace	Position the cursor at the column preceding the current column and delete any character in the new position by replacing it with a space character.
bottom	Position the cursor at the first column of the last line of the edit buffer.
caps	Change the mode of operation of the editor from the default mode to the mode in which each lowercase letter is translated to its uppercase counterpart as the user types it, or change the mode back to the default.
char_insert	Change the mode of operation of the editor from the default mode to character-insert mode or vice versa.
clr_bell	Clear the bell column.
clr_tab	Clear tab stops as specified.
column	Position the cursor at the specified column in the current line.
column_ind	Remove or return the column indicator.
config	Create a configuration file in the working directory.
consume_char	Delete the character in the current column and shift to the left any text to the right of the cursor.
consume_lb	Delete leading blanks and shift to the left any text to the right of the block defined by the cursor.
consume_line	Delete all characters in the current line at and to the right of the cursor.
consume_word	Delete the word in which the cursor is positioned and shift to the left the text to the right of the cursor.
control_char	Allow the user to insert a control character into the text.
copy_line	Copy a line to the line buffer.
copy_to_file	Copy the specified lines to a file.
copy_word	Copy a word to the word buffer.
cut_line	Copy a line to the line buffer and delete it from the text.
cut_word	Copy a word to the word buffer and delete it from the text.
delete_bl	Delete the blank current line and all blank lines between that line and the next nonblank line.
delete_line	Delete one or more lines of text and shift the following text to compensate for the deletion.
display	Display the contents of the specified buffer.

down	Move the cursor to the next line of the edit buffer.
flush	Write the contents of the edit buffer to the temporary file and, if so instructed, read more of the original file into the edit buffer.
global	Use as the range for the following command either the entire edit buffer or the contents of the buffer between the current line and the last line of the buffer, inclusive.
home	Position the cursor at the first column of the first line of the edit window or at the first column of the last line of the edit window.
home_line	Position the current line at the top of the edit window.
insert_line	Insert the specified number of blank lines between the current line and the line preceding it.
join	Move the text from the following line to the end of the current line.
left	Position the cursor at the previous column.
line	Position the cursor at the first column of the specified line.
ltab	Move the cursor to the left until it reaches a column that contains a tab stop.
move_lines	Move a section of text from one place in the file to another.
move_to_file	Move the specified lines to a file.
page_dwn	Scroll the file down by the screenful.
page_up	Scroll the file up by the screenful.
paste_line	Retrieve text from the line buffer and insert it into the file.
paste_word	Retrieve text from the word buffer and insert it into the current line.
pp	Put the editor in paragraph mode.
read_file	Read lines from the specified file and insert them in the file being edited.
replace	Replace the next occurrence of the string in the search buffer with the string in the word buffer.
return	Position the cursor at the first column of the next line.
rewind	End the editing session and reinvoke the editor with the same file.
right	Position the cursor at the next column.
rjustify	Put the editor in right-justify mode.
rtab	Move the cursor to the right until it reaches a column that contains a tab stop.
save	End the editing session and save the edited version of the file.
sdwn	Scroll the text down the specified number of lines.
search_bkw	Search backwards through the edit buffer for the previous occurrence of the specified string.
search_fwd	Search forwards through the edit buffer for the next occurrence of the specified string.

search_rep	Put one string in the search buffer, another in the word buffer, and replace the next occurrence of the search string with the string in the word buffer.
set_bell	Set the bell column to the current column.
set_tab	Set tab stops as specified.
shift_left	Shift text to the left by deleting characters in the block of text delimited by the movement of the cursor.
shift_right	Shift text to the right by inserting space characters in the block of text delimited by movement of the cursor.
split	Split a line of text into two lines.
sup	Scroll the text up the specified number of lines.
temp_exit	Give the user access to the shell program without ending the editing session.
top	Position the cursor at the first line of the edit buffer.
up	Move the cursor to the previous line of the edit buffer.
word_def	Include the specified characters in the set of characters considered to be part of a word.
word_tab	Change the mode of operation of the editor from the default mode to the mode in which the tab functions recognize the beginning of a word rather than a tab stop, or change the mode back to the default.

Appendix C

Syntax Summaries for the UniFLEX Screen Editor

abort	TAB q [<[<file_name>]] control-U
align	TAB j control-G
app	control-]
auto_ind	TAB a RETURN
backspace	control-H
bottom	TAB control-T
caps	control-_
char_insert	control-P
clr_bell	TAB c control-G
clr_tab	TAB <range> control-A
column	TAB <column_num> control-F
column_ind	TAB c RETURN
config	TAB ! control-U
consume_char	[TAB <range>] control-J
consume_lb	TAB <block> control-J
consume_line	TAB control-J
consume_word	TAB w control-J
control_char	control-\
copy_line	[TAB [<range>]] control-L
copy_to_file	TAB <output_dir> [<file_name>] [TAB <range>] control-L
copy_word	TAB w control-L
cut_line	TAB [<range>] control-K
cut_word	TAB w control-K

delete_bl	control-B
delete_line	TAB [<range>] control-B
display	TAB d <buffer_name> RETURN
down	control-C
flush	TAB <b_n_or_w> control-U
global	[TAB [<arg>]] control-^
home	control-D
home_line	TAB control-N
insert_line	[TAB <range>] control-W
join	TAB control-K
left	control-S
line	TAB <line_num> control-T
ltab	control-A
move_lines	TAB <range> control-O
move_to_file	TAB <output_dir> [<file_name>] [TAB <range>] control-K
page_dwn	TAB [<num>] control-R
page_up	TAB [<num>] control-V
paste_line	control-O
paste_word	TAB [<str>] control-P
pp	TAB p RETURN
read_file	TAB < [<file_name>] [TAB <range>] control-O
replace	[TAB <str>] control-X
return	control-M
rewind	TAB [<save_instr>] < control-U
right	control-F
rjustify	TAB r RETURN

rtab	control-G
save	TAB [<save_instr>] [<[<file_name>]] control-U
sdwn	[TAB <num>] control-Y
search_bkw	[TAB <search_str>] control-Q
search_fwd	[TAB <search_str>] control-Z
search_rep	TAB <search_str> TAB <replace_str> control-X
set_bell	TAB s control-G
set_tab	TAB <interval_size> control-A TAB s control-A
shift_left	TAB <block> control-B
shift_right	TAB <block> control-W
split	TAB control-W
sup	[TAB [<num>]] control-N
temp_exit	TAB +[<command_list>] control-U
top	control-T
up	control-E
word_def	TAB ws <char_list> control-A
word_tab	TAB w control-A

Appendix D

Summary of Commands by Control Character

control-A	clr_tab
	ltab
	set_tab
	word_def
	word_tab
control-B	delete_bl
	delete_line
	shift_left
control-C	down
control-D	home
control-E	up
control-F	column
	right
control-G	align
	clr_bell
	rtab
	set_bell
control-H	backspace
control-J	consume_char
	consume_lb
	consume_line
	consume_word
control-K	cut_line
	cut_word
	join
	move_to_file
control-L	copy_line
	copy_to_file
	copy_word
control-M	auto_ind
	column_ind
	display
	pp
	return
	rjustify

control-N	home_line
	sup
control-O	move_lines
	paste_line
	read_file
control-P	char_insert
	paste_word
control-Q	search_bkw
control-R	page_dwn
control-S	left
control-T	bottom
	line
	top
control-U	abort
	config
	flush
	rewind
	save
	temp_exit
control-V	page_up
control-W	insert_line
	shift_right
	split
control-X	replace
	search_rep
control-Y	sdwn
control-Z	search_fwd
control-_	caps
control-\	control_char
control-^	global
control-]	app