UniFLEX® Text Editor

MANUAL REVISION HISTORY

Revision	Date	Change
A	08/80	Original Release
B	04/81	Added documentation of the "File is a directory" error message. Removed documentation of the "Insufficient Stack Space" error message.
C	05/84	Updated for MC68000 version.
D	01/86	Fixed errors in the tutorial.

Contents

Contents

Preface

This manual describing the UniFLEX® Text Editor is divided into five chapters which include information on how to call the editor; a description of the interface between the editor and UniFLEX; a tutorial for those who are not familiar with the capabilities of the editor; a description of each of the editor commands, with examples; and finally, an annotated list of the messages that the editor may issue.

The editor is both content-oriented and line-oriented. Lines in the file being edited may be referenced either by specifying a line number or by specifying some part of the content of the line. Because the editor is not screen-oriented, it is not necessary to use a terminal with the capability to address the cursor directly.

Users who are unfamiliar with the previous versions of this editor (the stand-alone cassette version and the FLEX™ version) are encouraged to read the tutorial so as to get a feeling for the operation of the editor. The tutorial does not describe all of the features of the editor, but it does provide enough information to permit a user to perform useful edits. Once familiar with the basics, users should refer to the editor command descriptions for information on the rest of the commands.

CALLING THE EDITOR

The UniFLEX Text Editor is called with the UniFLEX command "edit". Arguments may be given to the editor which tell it the name of the file being edited, the name of the file that is to contain the revised file, and some editor options. All arguments are optional. How the editor functions when various arguments are supplied is described below through examples.

Calling the Editor with No Arguments

Example: edit

When the editor is called with no arguments, it issues a message that a new file is being created, and then prompts for the information that is to be put into the file. When the editing session is terminated (by the "stop" command, for example), the editor will prompt for the name of the file to which to write the information. The user responds to this prompt by typing in the file name, including a UniFLEX path name if necessary. If a UniFLEX end-of-file signal is typed in response to the prompt for a file name, all information is discarded and the editing session is terminated. (See UniFLEX INTERFACE section for more information on the end-of-file signal.)

Calling the Editor with a File Name

Example: edit test

If only one file name is given as an argument to the editor, it is assumed that this is the name of the file that is being edited. This name may include a UniFLEX path name if one is necessary to adequately describe the location of the file.

If the file does not exist, it is assumed that a new file having the specified name is being created. A message stating that fact is issued, and the editor then prompts for the information to be stored in the file. When the editing session is terminated, the information is written to the file.

If the file already exists, the information in it is read into an edit buffer and a prompt for an editor command is issued. When the editing session is terminated, the file will contain the revised information. The file as it was before editing is preserved in a backup file (unless the 'b' option was specified, as described later). The name of the backup file is normally the name of the original file with the characters ".bak" appended to the end of it. If the original name is too long to accommodate the additional four characters, the name is truncated and the

".bak" appended to the shortened name.

Calling the Editor with Two File Names

Example: edit test newtest

When the editor is called with two file names, the first file name is assumed to be the name of the file containing the information to be edited, and the second name is that of the file that is to receive the revised information. Both file names may contain UniFLEX path names if necessary to adequately describe their locations. If a path name is specified for the first file name, it is not propagated to the second file name. In the example, the file "test" is assumed to contain the information which is to be edited, and the file "newtest" is going to contain the edited information.

If the first file does not exist, the editor writes a message indicating that the edit file does not exist, and then terminates the edit session.

If the second file already exists, a prompt is issued asking for permission to delete the existing file. (This prompt may be avoided with the 'y' option, described below.) If a UniFLEX end-of-file signal is typed in response to this prompt, it is assumed that the file is not to be deleted, and the editing session is immediately terminated with no changes having been made.

Options

Options are specified to the editor by specifying an argument whose first character is a plus sign (+). The plus sign is immediately followed by one or more lowercase letters indicating the option or options selected. The valid letters are 'b', 'n', and 'y'. The option letters must be specified with lowercase letters. The options may be before, after, or intermixed with file name arguments.

The 'b' option - Do not create a backup file.

> Specifying a 'b' as an option tells the editor to not create a backup file containing the original, unedited, information.

The ´n´ option - Do not initially read the file being edited.

The ´n´ option is meaningful only if an existing file is being edited. Normally, the editor reads the file into memory so that the information may be manipulated with editor directives. By specifying ´n´ as an option, the information is not initially read into memory. The user may then use editor directives to enter new information, either from the terminal or by reading other files, that will appear in front of the information in the file being edited. The "new" command must be used to start the reading of the edit file. This option is most useful if a large amount of information is to be entered in front of the data being read from the file being edited. To insert only a small amount of information at the front of a file, the "insert" command may be used.

The ´y´ option - Delete any existing copy of the new file or the backup file.

Specifying ´y´ as an option will cause the editor to delete any existing copy of the backup file (if only one file name is specified) or the new file (if two file names are specified), without asking permission from the user. In effect, the editor assumes that a ´y´ response would be given in response to the prompts: "Delete existing backup file?", and "Delete existing copy of new file?" so the prompts are not issued, and the appropriate action is taken automatically.

If the editor cannot recognize a letter as a valid option, a message is issued and the editor continues to look for valid arguments.

Examples of calls including options.

```
edit test +b
edit test newtest +y
edit +nb test
```

UniFLEX INTERFACE

The UniFLEX Text Editor runs only under the UniFLEX Operating System. The editor therefore follows the conventions of the operating system with regard to special characters and file names. UniFLEX file names are treated in the documentation of the UniFLEX Operating System proper. The special characters and their effect on the editor are treated in this chapter.

Control Characters in a File

Normally, the editor allows any character to be in a file, including control characters. There are some characters, however, which have special meaning to the operating system and thus cannot be typed in from the keyboard. The special UniFLEX characters with which the editor is concerned are:

backspace character (User-definable using "ttyset" program)
escape character
line-delete character (User-definable using "ttyset" program)
horizontal-tab character (control-I)
control-D: keyboard signal for end-of-file
control-: keyboard interrupt
control-\: "quit" signal

Backspace character

The backspace character is used when entering commands and data to erase the last character typed. This backspace character may be changed by the user by calling the "ttyset" utility program.

Escape character

The ASCII escape character is used by the UniFLEX Operating System as a method of temporarily stopping and resuming the printing of information at the terminal. A more detailed description of the function of the escape character is described in the documentation of the UniFLEX Operating System proper. Here, it suffices to say that it is not possible to enter the escape character into a file using the editor.

Line-delete character

The line-delete character is used when entering commands and data to delete the line currently being typed. This line-delete character may be changed by the user by calling the "ttyset" utility program.

Horizontal-tab character

This tab character refers to the ASCII horizontal-tab character (HT), a hexadecimal 09. This is not the same as the tab character which can be defined within the editor. The editor itself is not concerned with the HT character, but UniFLEX may perform special handling when this character is typed or displayed. The action of UniFLEX on detecting this character is described elsewhere in the documentation of UniFLEX proper. It suffices to note that this HT character is treated as a single character by the editor, regardless of how it is displayed on the terminal.

Control-D: keyboard signal for end-of-file

When control-D is typed from the keyboard, the editor interprets this as an "end-of-file". The action taken by the editor on detecting this character depends on what the editor was expecting as input. If a control-D is typed in the middle of a command, it has the same effect as a line-delete character. If it is typed as the first character in response to a request for a command (that is, in response to the # prompt), it is treated as a "stop" command. A control-D typed while inserting lines has the same effect as typing the line-delete character followed by the line number character and a carriage return. That is, it cancels the current input line and the editor requests an editor command. The effect of typing this end-of-file signal in response to specific prompts depends on the prompt that was issued. Each such case is treated in the chapter "Editor Commands" when discussing the commands that issue the prompts.

Control-C: keyboard interrupt

The editor traps the control-C keyboard interrupt and uses it as a signal to stop executing an "append", "cchange", "change", "find", or "print" command. It has no effect on other commands. If the editor is executing multiple commands typed on a single line, typing a control-C will cause the editor to stop processing those commands and request a command from the keyboard.

Control-\: "quit" signal

The UniFLEX "quit" signal causes the editor to terminate immediately, without making any attempt to save the edited information. If an existing file was being edited when the "quit" signal was typed, the original file is left intact without any of the changes that had been made during the edit session.

The Editor's Use of Disk Files

The UniFLEX Text Editor is a disk-oriented editor. This means that the information being edited is read from and written to disk files. Other than the user's terminal, the only way to provide information to the editor is through disk files.

When the editor is called to edit an existing file, the information in that file is read into a large buffer in memory called the edit buffer. It is in this buffer that all of the changes to the information take place. When the user is satisfied with the changes made, the updated information is written to a disk file in response to specific commands.

If a file is larger than will fit in the edit buffer, the file must be processed in segments. With few exceptions, the editing commands operate only on that data which is in the edit buffer. Commands are provided which permit the user to flush the edit buffer of updated information and read in the next segment of data for editing.

How the editor manipulates disk files depends on whether it is creating a new file or editing an existing file. In some cases, a temporary file is created to hold the updated information. If used, this temporary file has a name composed of "edit." followed by 5 digits and a single letter. The 5 digits represent the task number of the editor. For example, "edit.00324a" is a valid temporary file name used by the editor. Unless the editor is terminated by a "quit" signal or a fatal system error, the temporary file is destroyed at the end of the edit session.

Creating a new file

When the editor is called with a single file name, and that file does not already exist, the editor will create the file at the start of the edit session and write directly into it as the edit session progresses.

When the editor is called with no file names specified, a temporary file in the user's current directory is created into which the information is written as the edit session progresses. At the end of the edit session, this temporary file is renamed to that name specified by the user in response to the "File name?" prompt.

Editing an existing file

When the editor is called with a single file name, and that file already exists, a temporary file is created into which the information is written as the edit session progresses. The temporary file is created in the same directory in which the file being edited resides. At the end of the edit session, the

original file is renamed to the backup file name and the temporary file is given the name of the original file. If no backup file is requested (by specifying the 'b' option), the original file is destroyed and the temporary file is given the name of the original file.

When the editor is called with two file names specified, the second file is created and the updated information is written directly into it. The original file is not changed.

Command Input from a File

It is possible to use the UniFLEX I/O Redirection capability of the shell program to have the editor read its commands from a file instead of from the keyboard. The editor will process the commands as though they were entered from the terminal's keyboard. If the end of the command file is reached before a "stop" or "abort" command is read, the action is the same as though a control-D were typed from the keyboard. This action has been described previously in the discussion of the control-D character.

Fatal Errors

The UniFLEX Text Editor attempts to make an intelligent decision when confronted with an error response to a UniFLEX system call. However, if an error is received which is unexpected and indicates that the editor cannot continue to function, it will issue a message and terminate immediately. The various messages, both fatal and non-fatal, are listed in the chapter "Editor Messages".

TUTORIAL

The purpose of this section is to briefly introduce the reader to the use of the UniFLEX Text Editor. We will, therefore, illustrate its use with a number of examples.

We will enter the editor with the UniFLEX command "edit test". Let us assume that the file "test" does not already exist, so we are creating a new one. The editor tells us "New file is being created" and prompts for information by displaying the first line number (1.00) followed by an equal sign. At this time we will create our file by simply typing all lines until finished, terminating each line with a carriage return.

```
New file being created
    1.00=This is an example of the
    2.00=UniFLEX Text Editing System. A number of
    3.00=examples will be presented to allow easy and
    4.00=rapid learning of its capabilities.
    5.00=The following are some nonsense lines:
    6.00=ABCDEFGHIJKL
    7.00=AAAAAAAA
    8.00=TESTING 1234
    9.00=This editor is easy to use!
   10.00=BBBBBBBB
   11.00=
   12.00=This is the end of this file,
   13.00=at least for now.
   14.00=#
   13.00=at least for now.
#
```

Notice it was necessary to type a pound sign (#) in column one to leave the data input mode. At this time, the system printed the last line and returned with the system prompt (a pound sign). The editor is now ready to accept commands.

Note also that both uppercase and lowercase letters were used in the text. The editor distinguishes between the two cases as will become more evident later on.

Each line of text in the edit file is given or has a line number which is used by the editor to uniquely identify the line. Each line number is of the form "m.nn" where 'm' is an integer and 'n' represents any of the digits 0 through 9. To specify a line number, one has to specify only enough of the line number to identify it uniquely. For example, 73, 73., 73.0, and 73.00 may be used to refer to line 73.00; 259.6 refers to line 259.60. The largest line number used with the editor is 65535.99. Let's denote a line number by the symbol "<line>". Later on, we will expand this definition of <line> to include other means of specifying a line; but for now, let us assume that it means a line number. We will be using this symbol throughout this manual.

An editor command tells the editor what action is to be performed and usually which line or block of lines are to be affected (if any). For each editing capability supported by the editor, there is a command which is used to indicate the desired action. For example, the editor can delete lines of text from a file, insert lines of text into the file, print lines contained in the file, and so on. Corresponding to each capability there is a command; hence, there is a Delete command, an Insert command, a Print command, and so on. If we define the symbol <command> to mean any editor command, the basic form of an edit command is

```
<line><command>
```

For example, the command to display (Print) line 12.00 is

```
#12P
   12.00=This is the end of this file,
#
```

where "12" is the <line> specification and 'P' is the <command> in this command. As can be seen in the example, this causes line number 12 to be printed on the terminal. Note that in this example we have used an uppercase 'P' as the command. Both upper- and lowercase letters are acceptable as commands; thus the command 'p' is the same as the command 'P'. This equivalence of upper- and lowercase does not hold true for the data in the lines of text. Here, a lowercase letter is treated as distinct from its uppercase form.

Now, let's learn how to use the insert command. In normal usage of the word "insert" we say something like: "Insert this card after this other card". To use the Insert command, we specify the line after which we want to insert new lines followed by an I:

```
<line>I
```

After typing the command followed by a carriage return, the editor will select an appropriate line number and prompt for input by displaying the line number followed by an equal sign. After each line of text is entered and the carriage return is typed, the editor will prompt for the next line. To exit from the "Insert mode" one simply types a pound sign followed by an edit command in response to a new line prompt. An "end-of-file" signal may also be used to leave the "Insert mode".

Some examples of the use of Insert are:

```
#8I
    8.10=This is an inserted line.
    8.20=So is this.
    8.30=#11 I
   11.10=Another inserted line.
   11.20=#6 P
    6.00=ABCDEFGHIJKL
#
```

It should be noted that the editor may renumber some lines following the inserted text. This occurs when enough lines are inserted such that the inserted line numbers overlap line numbers in the original text. When this occurs, an informative message is issued.

Next, let's learn how to use the Delete command. With this command we can delete one line or a block of lines with one command. To delete only one line, we specify the <line> to be deleted followed by a D:

```
<line>D
```

When the carriage return is typed, the line disappears.

To delete more than one line we need to indicate not only the first line to delete but also the last line to be deleted. Let's call the last line the "target" line and denote its specification as "<target>". Although the editor supports fancier ways to specify the <target>, we will just consider the two simplest: (1) <target> may be the number of lines to be deleted (counting both the first and last line of the block), or (2) <target> may be a pound sign followed immediately by the line number of the last line of the block to be deleted. Some example <target>s are: 3 (deletes three lines), 26 (delete 26 lines), and #26 (delete lines through line 26.00).

The syntax to Delete a block of lines is

```
<line>D <target>
```

where <line> indicates the first line to delete and <target> indicates the scope of the delete.

To illustrate the use of the Delete command, let's assume we have a file containing 53 lines with integer line numbers (i.e., 1, 2, 3, ...,53). With the commands

```
#15D
#24D #31
#52D 2
Bottom of file reached
#
```

we now have a file with lines 1 through 14, 16 through 23, and 32

through 51. The first command deleted line 15. The second command deleted lines 24 through 31. The third command deleted two lines starting with line 52. Since it deleted the last line of the file, the editor displayed the informative message "Bottom of file reached".

Before we discuss any more commands, we need to expand the definitions of <line> and <target>.

As editing operations are performed, the editor keeps track of the current line, which is usually the line most recently affected by a successful edit command. Upon entering the editor, the current line is the first line of the file. If, for example, we have just inserted three lines between lines 12.00 and 13.00, the current line will be 12.30. One should note that after a line or a block of lines have been Deleted, the line immediately following the last one deleted is made the current line (if the last line of the file was deleted, the new last line of the file will be the current line).

In our discussions above, we have implied that one has to explicitly indicate a <line> for each command by specifying the line number of the line of interest. However, if <line> is not specified in a command, the current line is used. For example, enter the following command:

```
#D 2
#
```

The editor will delete two lines starting with the current line. Since we were at line 6.00, the "D 2" operation deleted lines 6.00 and 7.00. As you will learn to appreciate, the current line default for <line> is extremely handy.

After performing all of the above operations, our file now looks like this:

```
 1.00=This is an example of the
 2.00=UniFLEX Text Editing System. A number of
 3.00=examples will be presented to allow easy and
 4.00=rapid learning of its capabilities.
 5.00=The following are some nonsense lines:
 8.00=TESTING 1234
 8.10=This is an inserted line.
 8.20=So is this.
 9.00=This editor is easy to use!
10.00=BBBBBBBB
11.00=
11.10=Another inserted line.
12.00=This is the end of this file,
13.00=at least for now.
```

We have seen that <line> may be specified by a line number or by default to the current line. There are also several other ways

to specify <line>. One may also specify <line> with a "+n" or "-n" (where 'n' is an integer) meaning the next nth line in the file or the nth previous line in the file, respectively. If the integer n is omitted, '+' indicates the next line, and '-' indicates the previous line. Two other useful <line> designators are '^' and '!'. The caret or circumflex '^' is used to designate the top or first line in the file. The exclamation point '!' is used to move to the last line or bottom of file. These various <line> specifiers are shown in the example below using the PRINT command.

```
#^P
    1.00=This is an example of the
#+3 P
    4.00=rapid learning of its capabilities.
#! P
   13.00=at least for now.
#-2P
   11.10=Another inserted line.
#
```

There may be times during the editing a file when we know part of the contents of the line of interest but don't know its line number or its displacement from the current line. In such a case we can use the content-oriented feature of the editor to find it. The syntax to specify <line> in this way is

```
/<string>/
```

where '/' is a character to delimit (enclose) the <string> which is a sequence of characters known to be in the line. When <line> is specified as "/<string>/", the editor will search from the current line through the file to find the next line containing the specified <string>. Some examples will help to clarify this: (1) /PRINT/ denotes the next line containing the character string "PRINT", and (2) /GO TO 35/ refers to the next line containing "GO TO 35". If the <string> is found in any subsequent line of the file, that line is made the current line and the requested edit operation is performed on it. If the <string> does not occur anywhere subsequent in the file, the editor issues the message "No such line" and does not change the current line pointer. Note that the delimiter does not need to be a slash; it may be some other character such as a quote (') or a comma. For example, 'A/B' refers to the next line containing "A/B".

It is also possible to prefix the string designator with '-' (minus sign) to indicate a previous line containing that string. A few examples with our sample file will show the use of "/<string>/" as a <line> designator.

```
#-/rapid/P
    4.00=rapid learning of its capabilities.
#;123; P
    8.00=TESTING 1234
#+'end' P
   12.00=This is the end of this file,
#
```

To summarize, we have seen that <line> may be specified in a number of ways, namely: (1) by default to the current line, (2) by typing a line number, (3) by "+n" denoting the nth subsequent line, (4) by "-n" referring to the nth previous line, (5) by /<string>/ denoting the next line in the file containing the indicated string of characters, (6) "-/<string>/" to denote the nearest previous line containing the specified character string, (7) '^' to denote the first line of the file, and (8) '!' to denote the last line of the file.

Now let's turn our attention to expanding the definition of <target>. As you may recall, a <target> is used in some commands to indicate the number of lines to be affected by the edit operation. We have already seen that a <target> may be specified by (1) an integer 'n' indicating the number of lines to be affected, such as P3, meaning print 3 lines, and (2) a line number preceded by a pound sign (#) indicating the line number of the last line to be affected, such as P #6, meaning print all lines to and including line #6. The <target> is simply a designator telling how many lines the edit command should affect. In addition to the two mentioned forms of <target>, we also have, (3) if no <target> is specified in a command whose syntax accepts one, a <target> of 1 is assumed, thereby affecting only one line. As with <line>, one may specify <target> by (4) "/<string>/" which indicates the next line in the file containing the specified character string, (5) '^' to denote the top line in the file, and (6) '!' to denote the bottom line in the file. A minus sign may be used to indicate that processing is to proceed backward through the file as in the following two cases: (7) "-n" and (8) "-/<string>/".

With this understanding of <line> and <target>, we can now discuss some more commands. The Print command is used to display a line or a group of lines. Its syntax is

```
<line>P <target>
```

where "<line>" and "<target>" may be specified in any of the ways discussed above. To print just a single line, one need specify only <line> followed by a carriage return; therefore, the following two commands perform the same thing:

```
<line>P
```

and

```
<line>
```

Going back to our test file, we can illustrate the various forms of <target> as used with the Print command.

```
#2P
    2.00=UniFLEX Text Editing System. A number of
#-1
    1.00=This is an example of the
#P /easy/
    1.00=This is an example of the
    2.00=UniFLEX Text Editing System. A number of
    3.00=examples will be presented to allow easy and
#! P -3
   13.00=at least for now.
   12.00=This is the end of this file,
   11.10=Another inserted line.
#-/BBB/ P -/123/
   10.00=BBBBBBBB
    9.00=This editor is easy to use!
    8.20=So is this.
    8.10=This is an inserted line.
    8.00=TESTING 1234
#12P!
   12.00=This is the end of this file,
   13.00=at least for now.
#
```

The first command displayed line 2.00 and made that line the current line. The second command requested that the line immediately preceding the current line be displayed. The third command displayed the block of lines from the current line down through the line containing the character string "easy". The fourth command printed 3 lines starting at the bottom of the file and ending at line 11.10, which becomes the current line. The fifth command requested the previous line containing the character string "BBB" be found, and then starting with that line, display all lines going backwards through the file until a line containing the character string "123" has been displayed. This shows the extreme usefulness and power of the content-oriented characteristic of the editor. The last command requested that all lines from line 12.00 to the end or bottom of file be displayed.

The next command to discuss is "Next" which is used primarily to move the line pointer. Although it may be used otherwise, usually it is used only with the default <line>. Its syntax is

```
N<target>
```

This command finds the line indicated by <target>, displays it, and makes it the current line. A few examples will illustrate its use.

```
#^P
    1.00=This is an example of the
#N
    2.00=UniFLEX Text Editing System. A number of
#N 6
    8.20=So is this.
#N -2
    8.00=TESTING 1234
#
```

The following command performs single-line replacements or inserts. Its syntax is

```
<line>=<text>
```

where "<line>" specifies the number of the line to be replaced or inserted and may, of course, default to the current line. "<text>" is the text that is to comprise the line. To illustrate this command, let's continue our example series.

```
#=Replace current line here
#5.25=This line created with "equals".
#
```

The first command changed the contents of line 8.00, the current line. The second example inserted a line with the line number 5.25.

The next command to be discussed is the "Change" command. It is used to change occurrences of one character string into another. Its syntax is

```
<line>C/<string1>/<string2>/ <target> <occurrence>
```

where '/' is a delimiter character to separate the two character strings; "<string1>" is the character string to be replaced; "<string2>" is the string of characters to replace them; "<target>" specifies the range of the changes; and "<occurrences>" specifies which occurrence(s) of <string1> should be replaced in the line(s). If <occurrence> is 1 or is not specified, then only the first occurrence of <string1> in any line of the block will be changed - the second or subsequent occurrence of the string in such a line will not be affected. If 2 is specified for <occurrence>, then only the second occurrence of <string1> in any line of the block will be changed. To change all occurrences of the indicated string in the block, use an asterisk (*) for <occurrence>. Let's illustrate the Change command by continuing our example.

```
#4C/rapid/fast/
    4.00=fast learning of its capabilities
#8.1C/This is //
    8.10=an inserted line.
#-5C;a;$; ;some; *
    3.00=ex$mples will be presented to $llow e$sy $nd
    4.00=f$st le$rning of its c$p$bilities.
    5.00=The following $re some nonsense lines:
#12C/e/?/ -2 3
   12.00=This is the end of this fil?,
   11.10=Another insert?d line.
#
```

The first example replaced the string "rapid" with the string "fast" in line 4.00. The second example deleted the string "This is" and a blank from line 8.1. The third example starts at the fifth previous line (line 3.00) and changes every occurrence of ´a´ to ´$´ down through all lines until the line containing the character string "some" (line 5.00) is reached. The last example changes the third occurrence of ´e´ to ´?´ in line 12.00 and then in line 11.10. Note that the target "-2" should also include line 11.00. However, since line 11.00 does not contain 3 occurrences of ´e´, no change is made.

The last command to be discussed is used to exit from the editor. This can be done several ways: STOP, S, or LOG. This will return you to the operating system.

Now let´s go back to our test file and illustrate some of the features and commands we have discussed.

```
#^P!
    1.00=This is an example of the
    2.00=UniFLEX Text Editing System. A number of
    3.00=ex$mples will be presented to $llow e$sy $nd
    4.00=f$st le$rning of its c$p$bilities.
    5.00=The following $re some nonsense lines:
    5.25=This line created with "equals".
    8.00=Replace current line here
    8.10=an inserted line.
    8.20=So is this.
    9.00=This editor is easy to use!
   10.00=BBBBBBBB
   11.00=
   11.10=Another insert?d line.
   12.00=This is the end of this fil?,
   13.00=at least for now.
#2C/ing System/or/
    2.00=UniFLEX Text Editor. A number of
#/BBB/
   10.00=BBBBBBBB
```

```
#-;This is; C 'e'xx' ! *
    1.00=This is an xxxamplxx of thxx
    2.00=UniFLEX Txxxt Editor. A numbxxr of
    3.00=xxx$mplxxs will bxx prxxsxxntxxd to $llow xx$sy $nd
    4.00=f$st lxx$rning of its c$p$bilitixxs.
    5.00=Thxx following $rxx somxx nonsxxnsxx linxxs:
    5.25=This linxx crxxatxxd with "xxquals".
    8.00=Rxxplacxx currxxnt linxx hxxrxx
    8.10=an insxxrtxxd linxx.
    9.00=This xxditor is xxsy to usxx!
   11.10=Anothxxr insxxrt?d linxx.
   12.00=This is thxx xxnd of this fil?,
   13.00=At lxxast for now.
#N -4
   10.00=BBBBBBBB
#-1 I
    9.10=TEST-TEST-TEST
    9.20=1234567890
    9.30=#D!
Bottom of file reached
#^D!
Bottom of file reached
#^P!

    1.00=This is an xxxamplxx of thxx
    2.00=UniFLEX Txxxt Editor. A numbxxr of
    3.00=xxx$mplxxs will bxx prxxsxxntxxd to $llow xx$sy $nd
    4.00=f$st lxx$rning of its c$p$bilitixxs.
    5.00=Thxx following $rxx somxx nonsxxnsxx linxxs:
    5.25=This linxx crxxatxxd with "xxquals".
    8.00=Rxxplacxx currxxnt linxx hxxrxx
    8.10=an insxxrtxxd linxx.
    8.20=So is this.
    9.00=This xxditor is xxsy to usxx!
    9.10=TEST-TEST-TEST
    9.20=1234567890

#S
```

This tutorial has been only a brief introduction to the UniFLEX Text Editor. The next section of this manual contains a detailed description of each command with examples. It is important to read and study the entire manual in order to fully understand and utilize all of the power and features of this editor.

EDITOR COMMANDS

The following section describes all of the editor commands. If you are not familiar with the editor, you would be well advised to first read the Tutorial section of this manual. It will give you an overall feel for what the editor can do, thus making the detailed descriptions more understandable. Before getting into the complete descriptions of the editor commands, a few general points will be covered.

Using Strings

Several of the editor commands use character strings as arguments. These arguments are either matched against strings in the text, or replace a string in the text. A string argument begins after a delimiter character and continues as a sequence of any characters until the delimiter is again encountered. The delimiters are not considered part of the string to be used in the matching or replacement operations. Although the delimiters in the following descriptions are frequently represented as slashes, '/', most any non-blank, non-alphanumeric character may be used as the delimiter such as: * / ( ) $, . [ ] : ' etc. Note that the following characters may not be used to enclose strings unless they are preceded by either a plus (+) or minus (-) sign: '^' (denotes first line of file), '!' (denotes last line of file), '-' (denotes target is above current line), and the character denoted by "lino" (normally a pound sign) which is used to indicate line numbers. The equals sign '=' may not be used as a string delimiter. The delimiter character is redefined in each new request by its appearance before a string. If two strings exist in one command (as in the "change" command), the same delimiter character must be used for each string.

All of the editor commands use the <line> information preceding the command to position the pointer prior to any command action. The <line> parameter may of course be null, meaning leave the pointer at its current position. All of the following are valid <line> designators:

1. Any number	references a specific line number
2. +n	denotes the nth subsequent line
3. -n	denotes the nth previous line
4. /<string>/	refers to the next line in the file containing the indicated string of characters

5. -/<string>/	refers to a previous line containing the indicated string
6. ^	denotes the first line of the file
7. !	denotes the last line of the file
8. null	stay at the current line

Line numbers less than 1.00 must be specified with a leading zero. For example, even though the editor may display a line number as ".10", it should be specified as "0.10" when used in commands. The maximum line number is 65535.99. Inserting after this maximum line number will cause the line numbers to "wrap around" back to zero.

Many of the editor commands require <target> information. This tells the editor to operate on the current line and all other lines in the file up to the line referenced by the <target>. In cases where a <target> is required, leaving it null will make the <target> default to one, meaning only the current line will be affected by the command. All of the following are valid <target> designators:

1. an integer n	indicates that n lines should be affected by the edit operation
2. #n	denotes the line number of the last line to be affected. The '#' is actually the "lino" character and may be changed by the user with the "set" command.
3. /<string>/	denotes the next line in the file containing the specified character string
4. -/<string>/	references the previous line containing the indicated string
5. ^	denotes all lines up to the top of the file
6. !	denotes all lines to the bottom or last line of the file
7. +or- n	indicates that n lines should be affected and in which direction from the current line
8. (null)	defaults to 1 and only the current line is affected

As we have seen, the form <target> is used to specify a range of lines to which the command will apply. The command will be applied to each line, starting with the line specified by <line> and continuing until the target is reached.

If a string <target> is specified, the command will apply to successive lines of text until a line containing the string is reached. Processing proceeds downward in the edit buffer unless the target is preceded by a '-' (minus sign), indicating that processing is to proceed upward (toward the first line) in the edit buffer. Targets may also be preceded by a plus sign (indicating downward movement). If a line number target is specified, processing begins at <line> and proceeds toward the target line number. Some examples of <target>s are:

```
2
+10
-3
/STRING/
+/STRING TARGET/
-/BACKWARD DISPLACEMENT TO A STRING/
+*ANY DELIMITER WILL WORK FOR STRING*
++EVEN PLUS SIGNS CAN WORK+
#23.00
```

Specifying a Column Number

Any "/<string>/" descriptor may be postfixed with a column number immediately after the second delimiter to indicate that the preceding string must begin in the column specified. If the column specified is not in the range of the zone in effect, the request will be ignored. (See the "zone" command.) Some examples are:

```
/IDENT/11
/PROGRAM/77
*LABEL*2
$COMMENT$30
```

Using the "Don't Care" Character

See 'SET' on page 6

A "Don't Care" character may be set to allow indiscriminate matches of parts of a string. When this character is placed in a string, any character in the file will automatically match. The Don't Care character will have its special meaning only in a string being used to search the file. In other words, the Don't Care character will not act as such in a replacement string such as the second string of a "change" command. The Don't Care character may be effectively disabled by setting it to a null. Assuming we have previously set the Don't Care character to a '?', here are some examples:

/A???/	This would match any 4 letter string beginning with A
@03/??/78@	This would match all days in the 3rd month of 1978
/???/9	This would match any 3 letter string starting in column 9

The "Command Repeat" Character

A special "Command Repeat" character has been set up in the editor to allow you to exactly repeat the last command in the input buffer. The repeat character is set to a control-R. Some examples of commands which may be useful to repeat are:

PRINT 15	To print a screen of lines at a time
NEXT	Allows you to single step through the file with one key
^CO!!	To quickly fill the workspace
FIND/SOME STRING/	If the first string found is not the one desired

Using the "End of Line" Character

The editor supports an or "End Of Line" or "eol" character to allow multiple commands in a single line. There are some commands that cannot be followed by another command on the same line. This fact is documented in the descriptions of those commands. The "eol" character may be changed by using the "set" command. An example of "eol" use (with "eol" set to '$') is:

```
^D2$P10$T
```

This sequence will delete the first 2 lines of the file, then print the next 10 lines, and finally return the pointer to the top of the file.

Using Tab Characters

The user may specify a tab character and up to 20 tab stops. The tab character may then be inserted into a line where it will be replaced by the appropriate number of fill characters when the end of the line is received. The fill character defaults to a space, but may be changed to another character with the "set" command. If tab stops or the tab character have not been previously set, but some character has been used throughout the file as a tab, it can still be expanded by setting it to be the tab character, setting up your tab stops and then using the "expand" command on the file. Note that if the tab character has been set, subsequent uses of the "insert" or "replace" commands will cause automatic tab expansion. However if a tab character is added to the file by the use of a "change", "append", or "overlay" command, that character will remain intact in the file until the "expand" command is invoked on the line containing that tab character. After tabs are expanded, the tab character no longer exists in the data. All occurrences will have been replaced by the appropriate number of fill characters. Setting the tab character to be the same as the fill character effectively disables the tab feature.

Note the the tab character described above is distinct from the ASCII horizontal-tab character (HT or control-I). The effect of the HT character is described in the chapter "UniFLEX INTERFACE". It is possible to set the editor tab character to the HT character. If this is done, UniFLEX may take special action when the HT character is typed, but the character will be replaced by fill characters when it is put into the edit buffer.

Length of Lines of Text

Lines entered from the keyboard are limited to 255 characters. The lines in the text file may be of any length. Lines longer than 255 characters may be created with the "merge" and "append" commands.

Editor Commands

There are five groups of editor commands: environment commands, system commands, current line movers, edit commands, and disk commands. A complete description of all commands in each group is given below. In the following descriptions, quantities enclosed in square brackets ([...]) are optional and may be omitted. A backslash (\) is used to separate options. Many commands have abbreviations. Both the full name of the command and its abbreviation are given. A command and its abbreviation may be used interchangeably. All commands below are in lower case; however, in use, a command may be in either upper case or lower case.

Environment commands

dkl <command string>

MEANING:

"dkl" is used to define one of two "command constants". The <command string> is a single command or several commands separated by the "eol" character (see "set" command). All of the command line, including the carriage return is assumed to be the argument to the "dkl" command. This command string may be executed at any time by the "kl" command, described below. The "dkl" command is most useful for remembering and re-executing a frequently used sequence of commands.

EXAMPLES:

dkl f -/.nl/1$i/.sp	Define a command sequence of "f -/.nl/1" followed by "i/.sp". This assumes that "eol" is '$'. This sequence may be executed by typing "kl".

dk2 <command string>

MEANING:

"dk2" is used to define one of two "command constants". The <command string> is a single command or several commands separated by the "eol" character (see "set" command). All of the command line, including the carriage return is assumed to be the argument to the "dk2" command. This command string may be executed at any time by the "k2" command, described below. The "dk2" command is most useful for remembering and re-executing a frequently used sequence of commands.

EXAMPLES:

dk2 c /sample// 1 2	Define the command constant: "c /sample// 1 2". This command may be executed by typing "k2".

esave [<UniFLEX path specification>]

MEANING:

The "esave" command saves the current editor "environment" on an "editor configuration" disk file named ".editconfigure" in the user's directory. The editor environment consists of the "header" column count; the "numbers" and "verify" flags; current tab stops; the "tab", "dcc", "fill", "eol", and "lino" characters; the commands saved as command constants "k1" and "k2"; and the search zones in effect. When the editor is called, the environment is automatically set from the configuration file in the user's directory, if one exists. The editor environment may also be reset from the configuration file at any time during the edit session by the "eset" command, described below.

The environment information may be saved in a directory other than the user's current directory by specifying a UniFLEX path as an argument to the "esave" command. This path must include only directory names and must be terminated by the separator character '/'.

EXAMPLES:

esave	Save the current editor environment on the file ".editconfigure" in the user's directory.
esave /usr/dde/	Save the current editor environment in file "/usr/dde/.editconfigure".

eset [<UniFLEX path specification>]

MEANING:

The "eset" command is used to reset the editor environment from an editor "configuration" file created by the "esave" command (see above). The configuration file is named ".editconfigure" and is normally expected to be found in the user's current directory. A UniFLEX path may be specified as an argument to the "eset" command to force the searching of a different directory. This path must include only directory names and must be terminated by the separator character '/'.

EXAMPLES:

eset	Reset the editor environment from the file ".editconfigure" in the user's directory.
eset /usr/dde/	Reset the editor environment from file "/usr/dde/.editconfigure".

header [<count>]
h [<count>]

MEANING:
A header line of <count> columns will be displayed. The heading consists of a line showing the column numbers by tens, followed by a line of the form "123456789012..." to indicate the column number. Columns for which tab stops are set will contain a hyphen instead of the normal digit. If a column count is given, it becomes the default such that if just 'h' is subsequently typed, that number of columns will be printed.

EXAMPLES:

header 72	Display column number headings for 72 columns
h 30	Display column numbers for 30 columns

k1

MEANING:
Execute the command constant that was defined by "dk1". If no command constant was defined, the current line is printed. This command may not be followed by another command on the same line.

EXAMPLES:

k1	Execute the command constant.

k2

MEANING:
Execute the command constant that was defined by "dk2". If no command constant was defined, the current line is printed. This command may not be followed by another command on the same line.

EXAMPLES:

k2	Execute the command constant.

lkl

MEANING:
Display the command constant that was defined by "dkl". If no command constant was defined, a blank line is printed.

EXAMPLES:

lkl	Display the command constant.

lk2

MEANING:
Display the command constant that was defined by "dk2". If no command constant was defined, a blank line is printed.

EXAMPLES:

lk2	Display the command constant.

numbers [off\on]
nu [off\on]

MEANING:
The line number flag is turned off or on. If the flag is off, then line numbers will never be printed. If neither "off" nor "on" is specified, then the flag will be toggled from its current state.

EXAMPLES:

numbers off	Turn line number printing off
nu on	Turn it back on
nu	Toggle from on to off or from off to on

renumber
ren

MEANING:
The "renumber" command will renumber all of the lines in the current edit buffer. Lines in the renumbered buffer will start with the line number of the first line in the buffer and will have an increment of one. The current line does not change, although its number will probably have been changed.

EXAMPLES:

renumber	Renumber the lines in the current edit buffer
ren	

set <name> = '<char>'

MEANING:
"set" is used to define certain special characters or symbols. The <name>s which may be set are:

tab - the tab character
fill - the tab fill character
dcc - the "don't care" character for string searches
eol - the end of line character which may be used to separate several commands on a single line
lino - the line number flag character which is used to indicate that a target is a specific line number

The default values are: dcc, tab, and eol are null
fill is the space character
lino is '#'

The default values may be initialized from a configuration file in the user's directory. See the "esave" command.

EXAMPLES:

set tab='/'	Set the tab character to a slash
set tab=''	Disable tabbing by setting the tab character to a null
set fill=' '	Set tab fill character to a blank
set eol='$'	Set the EOL character to $
set lino='@'	Set the line number flag to @

tab [<columns>]

MEANING:
Used to set the tab stops. All previous tab stops are cleared. If no columns are specified, then the only action is to clear all tab settings. Any tab characters occurring beyond the last tab stop are left in the text. The maximum number of tab stops allowed is 20. Tab stops MUST be entered in ascending order.

EXAMPLES:

tab 11,18,30	Set tab stops at columns 11, 18, and 30
tab 7 72	Set tab stops for a FORTRAN program
tab	Clear all tab stops

verify [on\off]
v [on\off]

MEANING:
The verify flag is turned on or off. The verify flag is used by the commands "change" and "find" (and several others) to display their results. If neither "on" nor "off" is specified, then the flag will be toggled from its current state.

EXAMPLES:

verify off	Turn verification off
v on	Turn it back on

zone [c1,c2]
z [c1,c2]

MEANING:
"zone" is used to restrict all sub-string searches (find, change, <target>s, etc.) to columns "c1" through "c2" inclusive. Any substrings beginning outside those columns will not be detected. If "c1" and "c2" are not specified, then the zones will be reset to their default values (columns 1 and 255). A string which starts within the specified search zone and extends out of it will still match a target.

EXAMPLES:

zone 11,29	Restrict searches to columns 11 through 29

zone	Search columns 1 through 255

System commands

abort

MEANING:

This command terminates the edit session without saving any of the changes made during that session. The original file, if one exists, is left intact. When typed, this command will prompt "Are you sure?". If a 'y' is then typed, the edit session will be terminated. Typing an 'n' or end-of-file signal will cause the editor to look for another command. Typing any other character will cause the prompt to be issued again.

EXAMPLES:

abort	Abort the editing session.

edit <editor arguments>
e <editor arguments>

MEANING:

The "edit" command causes the current editing session to be terminated (as though a "stop" or "log" command had been entered), and another editing session started. The <editor arguments> are any valid file names and editor options as described in the section on "Calling the Editor". This command may not be followed by another command on the same command line. All changes to the editing environment made by "Environment Commands" remain in effect.

EXAMPLES:

edit test +b	Terminate the current editing session and start editing file "test" with editor option 'b'.

log

MEANING:

This command causes the editing session to terminate. The updated information is written to the new file, and, if necessary, any unprocessed data from any existing file is copied to the new file. A backup file is created if appropriate (see the chapter "UniFLEX Interface for more information on the editor's handling of disk files at the end of an editing session).

EXAMPLES:

log

stop
s

MEANING:
Same as "log".

EXAMPLES:

stop

s

u <UniFLEX Command>

MEANING:
The 'u' command permits the execution of a UniFLEX command. The specified command is passed to the shell program for execution. The editor waits for the UniFLEX command to finish before prompting for another editor command. This command may not be followed by another editor command on the same line.

EXAMPLES:

u list test	List the file "test"
u copy test test1	Copy the file "test" to "test1"

wait

MEANING:
The "wait" command is used to wait for the completion of a background task generated by the 'x' command (described below). This command cannot be used to wait for completion of a background task that was not generated by the editor. The editor will not request a command until the background task is completed or a keyboard interrupt (control-C) is typed. When the background task terminates, a message is displayed specifying the task number and whether it completed normally or abnormally. In the event of abnormal termination, the response code or interrupt code that caused the termination is given.

EXAMPLES:

wait	Wait for the background task to complete

x <UniFLEX Command>

MEANING:
The 'x' command is used to start a background task running. The <UniFLEX Command> which was specified as the argument is passed to the shell program for execution. The task generated must run to completion before the editor will allow the generating of another such background task. The "wait" command must be used to receive the termination status of a task before the 'x' command may be used again. This command may not be followed by another command on the same line.

EXAMPLES:

x copy test test1	Copy "test" to "test1" as a background task. A "wait" command must be used to determine the termination status of the task before another background task can be generated.

Current line movers

bottom
b

MEANING:
Move to the last line in the file and make it the current line.

EXAMPLES:

bottom	Make the last line of the file the current line
b	

find <target> [<occurrence>]
f <target> [<occurrence>]

MEANING:
Move the current line pointer to the line specified by <target> and make it the current line. If the verify flag is on (see "verify"), the line will be printed. If <occurrence> is specified (an unsigned integer or an asterisk), the command will be repeated <occurrence> times. If <occurrence> is an integer, it must not start in the first column following the second delimiter of a string <target>, as it would then appear to be a column specifier for that string. If no column is to be

specified, insert a space after the second delimiter and before the <occurrence> as in the second example given below. An asterisk means all occurrences of the <target> will be found until the bottom or top of the edit buffer is reached. If the target is not found, the current line pointer will not be moved.

EXAMPLES:

find /string/	Find the next line containing the string "string"
f/three lines/ 3	Find the next three lines containing the string "three lines"
f/all 'til bottom/*	Find all following occurrences of the indicated string
f-/program/7 *	Find all previous lines which have the word "program" starting in column seven

next [<target> [<occurrence>]]
n [<target> [<occurrence>]]

MEANING:
The line specified by the target is made the current line. If the verify flag is on, the line will be printed. If <occurrence> is specified, it must be an unsigned integer. It indicates which next occurrence of a line containing the target is to be made the current line. If the target is not reached, the current line pointer will be positioned at the bottom of the edit buffer (or top of the edit buffer for a negative <target>). If no target is specified, the next line will be made the current line.

EXAMPLES:

next 5	Make the fifth following line the current line
n	Make the next line the current line
n-10	Make the 10th previous line current
n/string target/	Make the next line containing "string target" to be the current line
n/3rd occurrence/ 3	Make the third line containing the indicated string the current line

position <target>
pos <target>

MEANING:
Search forward through the file for an occurrence of <target>. If the target is not found in the current edit buffer, the edit buffer is flushed and the next edit buffer is read from the file being edited. This process continues until the target is located or the end of the file is detected. If the target is found, it is made the current line. If the target cannot be located, the current position is the first line in the last edit buffer. The <target> may not be a "backwards target" (preceded by a minus sign) and may not be an integer indicating relative displacement. Only a string or a line number (preceded by the "lino" character) are valid targets. Search zones are honored during the search for the target. A column number is allowed after the target, but an occurrence specification is not permitted.

EXAMPLES:

position /string/5	Position to the line containing the string "string" in column 5.
pos #1000	Position to line number 1000

top
t

MEANING:
The first line of the file becomes the current line.

EXAMPLES:

top	Make the first line of the file the current line

Edit commands

append /<string>/ [<target>]
a /<string>/ [<target>]

MEANING:
Append the specified <string> just beyond the last character of the current line (and to successive lines until the target is reached). If the string is postfixed with a column number, then append the string beginning at the specified column (rather than at the end of the line). Any characters previously in the line following the specified column will be lost.

EXAMPLES:

append /./	Append a period to the end of the current line
a *HELLO* 2	Append the word "HELLO" to the end of the current line and to the end of the next line.
a/sequence/73 *END*7	Append the word "sequence" starting in column 73 of the current line and successive lines until a line containing the characters "END" beginning in column seven is found.

break

MEANING:

The "break" command allows the splitting of a line into two lines. The current line is printed, then a line of input is accepted from the terminal (the break line). When the line is printed, all ASCII HT characters will be displayed as spaces so that the terminal cursor will not be artificially advanced. The break line will be positioned directly beneath the line printed out. Characters are then typed by the user until the cursor is beneath the character that is to be the first character of the second line. A carriage return is typed to effect the splitting of the line. Any characters may be typed to move the cursor. After the line is split, the second half of the broken line becomes the current line. If an end-of-file signal is typed in response to the "Break---" prompt, the current line will not be changed. The current line will also not be changed if the carriage return typed in the break line is beyond the end of the current line.

EXAMPLES:

```
break
  25.00 This is the current line.
Break---xxxxxxxxxxxx
```

The line will be broken at the start of the word "current".

change /<string1>/<string2>/ [<target> [<occurrence>]]
c /<string1>/<string2>/ [<target> [<occurrence>]]

MEANING:

Replace the string specified by <string1> with the string specified by <string2>. If no <target> is specified, only the current line is affected. The slashes represent any non-blank delimiter character. <occurrence> is used to specify which occurrence of <string1> is to be replaced in each line. It is either an unsigned integer or an asterisk (*) signifying that all occurrences of the substring <string1> are to be replaced with <string2>. By default, only the first occurrence will be changed. Note that if <occurrence> is specified, and if changes are to occur to the current line only, then the target should be a 1 (one).

EXAMPLES:

change /this/that/	Replace the first occurrence of "this" in the current line with "that"
c/A/B/ 1*	Change all occurrences of 'A' in the current line to 'B'
c /first/last/10	Change the first occurrence of "first" to "last" in the current line and also in the nine following lines
c /new/old/ /a target/	Change the first occurrence of "new" to "old" in each line down through the line containing the string "a target"
c ,a,, -10 *	Remove all 'a's in the current line and in the nine preceding lines
c*Hello*	Delete the character string "Hello" from the current line

To insert at front of line use c //

cchange /<string1>/<string2>/ [<target> [<occurrence>]]
cc /<string1>/<string2>/ [<target> [<occurrence>]]

MEANING:

"cchange" stands for Controlled Change. This command is exactly like the normal "change" command except that the user can interactively specify whether each line containing <string1> should actually be changed or left as is. This allows the user to step through the edit buffer and selectively change certain strings. When a

line containing <string1> is found, it is displayed at the terminal and the user receives a prompt, "Change?" If it is desired that the line be changed, type a 'y' for yes. A character other than 'y' will cause the line not to be changed. If an 's' or end-of-file signal is typed, the command will terminate. Other characters will cause a search for the next line containing <string1>.

EXAMPLES:

cchange/ALPHA/OMEGA/!*	Perform a Controlled Change on all occurrences of "ALPHA" through the rest of the file
cc;a;z;-20 3	Perform a Controlled Change on the third occurrence of 'a' in the current and previous 19 lines

copy [<destination-target> [<range-target>]]
co [<destination-target> [<range-target>]]

MEANING:

The current line and successive lines until the <range-target> is reached are copied so that they follow the line specified by <destination-target>. The default <destination-target> is 1, thereby causing a copy of the current line to be placed after the next line. The default <range-target> is 1, thereby copying only one line. After the command is executed, the current line pointer will be set to the new position of the last line copied. Some lines may be renumbered after a copy with no renumbering message issued.

EXAMPLES:

co #18	Put a copy of the current line after line 18
copy #3 4	Copy four lines beginning with the current line and place them after line 3
co /check/ +/range/	After the next line which has the string "check", place a copy of each line starting with the current line through the line containing "range"

delete [<target>]
d [<target>]

MEANING:
The current line (and successive lines until the target is reached) is deleted. After the command is executed, the current line will be the line following the last line deleted.

EXAMPLES:

delete 5	Delete five lines (the current line and the next four lines)
d	Delete the current line
d /STRING/	Delete lines from the current line through the next line that contains the string "STRING"

expand [<target>]
exp [<target>]

MEANING:
The current tab character is expanded within all lines, beginning with the current line, continuing down to and including the line specified by <target>. Since tabs are normally expanded as lines are inserted into the file, this command is primarily of use when one has forgotten to define a tab character or has inserted a tab character with an "append", "overlay", or "change" command.

EXAMPLES:

expand 100	Expand 100 lines starting with the current line
exp	Expand the current line

insert
i

MEANING:
The editor will enter the input mode, prompting with line numbers (unless line numbers have been disabled, see the "numbers" command) and insert the lines below the current line. Input continues until a line beginning with the "lino" character in column one is received, or the end-of-file signal is typed in column one. The characters following the "lino" character are treated as an editor command. The editor will try to choose an insertion increment sufficient to insert at least 10

lines, or if that is not possible, the smallest increment possible. The current line pointer is positioned at the last line inserted. Lines may be inserted at the top of the edit buffer by specifying a line number of zero. It should be noted that the editor may renumber text lines following the inserted text if the inserted line numbers overlap line numbers previously in the file. This command may not be followed by another command on the same line. Note that if the line-delete character is typed, the prompt is not reissued.

EXAMPLES:

insert	Accept line input after the current line
0i	Insert at the top of the edit buffer.

insert <text>
i <text>

MEANING:
The text (sequence of characters) which immediately follows the separator (or blank) after the command name will be inserted as a separate line below the current line of the file. The line inserted becomes the current line. It should be noted that the editor may renumber text lines following the inserted text if the inserted line number overlaps line numbers previously in the file. This command may not be followed by another command on the same line.

EXAMPLES:

```
I This below the current line of the file

insert everything after the first blank
```

merge

MEANING:
The "merge" command is used to combine the current line and the line immediately following it into a single line. The merged line becomes the current line.

EXAMPLES:

merge	Merge the current line and the next line into a single line.

move [<destination-target> [<range-target>]]
mo [<destination-target> [<range-target>]]

MEANING:

The current line and successive lines until the <range-target> is reached are moved so that they follow the line specified by <destination-target>. The default <destination-target> is 1, thereby moving the current line after the next line in the file. The default <range-target> is 1, thereby moving only one line. After the command is executed, the current line pointer will be set to the new position of the last line moved. Some lines may be renumbered after a move with no renumbering message issued.

EXAMPLES:

move 3	Move the current line down three lines
mo #1 /TARGET STRING/	Move the current line and all lines up to and including the line containing "TARGET STRING" after line 1
mo -/Program/ 5	Move five lines (including the current line) up within the file so that they follow a line containing the character string "Program"
mo #10 -5	Move the current line and the four previous lines below line number 10

overlay[<delimiter>]
o[<delimiter>]

MEANING:

The current line is printed, then a line of input is accepted from the terminal (the overlay line). When the line is printed, all ASCII HT characters will be displayed as spaces so that the terminal cursor will not be artificially advanced. The overlay line will be positioned directly beneath the line printed out. Each character of the overlay that is different from the <delimiter> character (which defaults to a blank) will replace the corresponding character in the current line. The overlaid line will be printed if verify is "on". If the end-of-file signal is typed in response to the prompt for the overlay line, the current line will not be changed.

EXAMPLES:

```
overlay
  25.00=THIP IS THE CORRENT LUNE.
Overlay    S         U       I
  25.00=THIS IS THE CURRENT LINE.
```

overlay<d><text>
o<d><text>

MEANING:
This command is similar to the previous form of the "overlay" command with these differences: (1) The current line is not printed. (2) The remainder of the command line (after the delimiter character) is taken as the overlay text.

EXAMPLES:

```
overlay---AT-------------------- NUMBER.
  25.00=THAT IS THE CURRENT LINE NUMBER.
```

print [<target>]
p [<target>]

MEANING:
Beginning with the current line, lines are printed until the line specified by <target> is reached. By default, only the current line will be printed.

EXAMPLES:

p	Print the current line
print 5	Print 5 lines starting with the current line
p -10	Print the current line and the nine previous lines
print *string*	Print all lines up to and including the next line containing "string"
p -/string/	Print all lines up through the next previous line containing "string"

replace [<target>]
r [<target>]

MEANING:
A "delete" from the current line through the <target> line is performed. The editor then enters the input mode, putting the new lines into the area vacated. It is not necessary to enter the same number of lines as were deleted. The line numbers of the lines inserted will probably not be the same as those deleted. The current line pointer will be positioned at the last line inserted. By default, only the current line will be deleted. This command may not be followed by another command on the same line.

EXAMPLES:

r	Replace the current line
replace 10	Replace 10 lines starting with the current line
r /TARGET STRING/	Replace all lines from the current line through the line containing "TARGET STRING"

=<text>

MEANING:
The '=' command replaces the current line with the text supplied. The replacement text begins with the first character following the equals sign. The current line pointer is not moved.

EXAMPLES:

=THIS IS REPLACEMENT TEXT.

(null)

MEANING:
The null command (i.e., just a carriage return) prints the current line.

flush

MEANING:
The information above the current line in the edit buffer is written to the file containing the updated data and then deleted from the edit buffer. This command is used to make room in the edit buffer for large insertions.

EXAMPLES:

flush	Flush information above the current line to updated file.
200flush	Flush information above line 200 to the updated file.

new

MEANING:
The information above the current line in the edit buffer is written to the file containing the updated data and then deleted from the edit buffer. The available space in the edit buffer is then filled with data read from the file being edited. This command is used primarily to proceed to the next segment of the file when modifications to the current edit buffer have been completed. If a new file is being created, the "new" command is the same as the "flush" command.

EXAMPLES:

new	Write the information above the current line to the updated file and read more data from the file being edited.
!new	Write the current edit buffer (except for the last line) to the updated file and read the next segment from the file being edited into the edit buffer.

read [<file name>]

MEANING:

The entire content of the file specified as an argument is read and placed after the current line. The last line of the information read becomes the current line. If no file name is specified, a prompt for one will be issued. If an end-of-file signal is typed in response to the prompt for a file name, no data is read. The file name may contain path information if any is necessary to locate the file. The entire content of the file must fit into the remaining unused space in the edit buffer. If the file being read will not fit into the edit buffer, the message "Not enough room" is issued and no data is read.

EXAMPLES:

read /dde/data	The information in the file "/dde/data" is read and placed after the current line.
100read moredata	The information in the file "moredata" is read and placed after line 100.

write [<target>]

MEANING:

The information from the current line through <target> is written to a file. A prompt is issued for the name of the file that is to receive the information. If an end-of-file signal is typed in response to the prompt for the file name, no information is written. If the file being written already exists, it is destroyed and a new file created. If no <target> is specified, only the current line is written.

EXAMPLES:

write /window/	Write the information from the current line through the line containing the string "window".
100write 200	Write lines 100 through 200, inclusive, to a scratch file.

EDITOR MESSAGES

A task is already running

The 'x' command was used when there was already a task generated by a previous 'x' command still running. The "wait" command must be used to wait for the previous task to complete before another background task may be initiated.

Attempting to merge onto last line of text

Attempting to use the "merge" command on the last line of the file fails because the "merge" command joins the specified line with the following line, and if the specified line is the last line of the file, there is no line following the specified line to join with it.

Bottom of file reached

An informative message issued when the last line of the file is deleted.

Cannot create configuration file

A configuration file could not be created in the directory specified in the "esave" command (current directory if no directory was mentioned). Usually this means that the directory specified could not be found or you don't have write permissions on that directory. Make sure the directory was specified with a trailing '/' character.

Cannot create new file

The editor was called with two file names as arguments, but the second file could not be created. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor. This message occurs only at the beginning of an editing session.

Cannot create new backup file

The editor detected an error attempting to create a backup file containing the information as it was prior to the editing session. This message is preceded by a message indicating which error was detected. The new backup file is not created and the editing session continues.

Cannot create task

An error was detected when trying to generate a task with the 'u' or 'x' command. This message is preceded by a message indicating which error was detected. The command is aborted and the editor requests a new command.

Cannot create temporary file

The editor detected an error when trying to create the temporary file that holds the updated information. This message is preceded by a message indicating which error was detected. This message occurs only at the beginning of an editing session.

Cannot delete old backup file

At the end of an editing session, the editor attempts to create a backup file containing the information as it was prior to the editing session. However, a file with the same name as the backup file would have already exists and could not be deleted. This message is preceded by a message indicating which error was detected. The new backup file is not created and the editing session continues.

Cannot open configuration file

The configuration file in the directory specified in an "eset" command could not be opened. This usually means that there was no configuration file in the specified directory, or that the specified directory could not be found, or that the permissions on the configuration file in the specified directory exclude the user from opening the file for reading. Remember that the directory name must be specified with a trailing '/' character.

Cannot open edit file

The file that is being edited exists, but could not be opened. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor. This message occurs only at the beginning of an editing session.

Cannot open new file

The editor was called with two file names as arguments, but could not open the second file to determine if it already exists. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor. This message occurs only at the beginning of an editing session.

Cannot read configuration file

The UniFLEX system reported a media error while the editor was trying to read from the editor configuration file.

Cannot read edit file

The UniFLEX system reported a media error while the editor was reading from the file whose data is being edited.

Cannot rename files

The editor detected an error trying to rename the files at the end of an editing session. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor. The user should then search for the temporary file used by the editor. This file will contain the updated information and should be copied to another file for safekeeping.

Cannot write configuration file

The UniFLEX system reported a media error while the editor was writing configuration data to the configuration file in the specified directory (current directory if the specification was omitted).

Delete existing backup file?

At the end of an editing session, the editor attempts to create a backup file containing the information as it was prior to the editing session. However, a file with the same name as the backup file would have already exists. This message is a request for permission to delete the existing file, replacing it with the new backup file. The prompt must be answered with a 'y', for "yes", or an 'n', for "no". If 'y' or the end-of-file signal is typed, the file is deleted and the new backup file is created. If 'n' is typed, the file will not be deleted and no new backup file created. If none of these are typed, the prompt is re-issued.

Delete existing copy of new file?

The editor was called with two file names as arguments. The second file already exists and must be deleted before the editing session can continue. This message is request for permission to delete the file. The prompt must be answered with a 'y', for "yes", or an 'n', for "no". If 'y' is typed, the file is deleted and the editing session continues. If 'n' or the end-of-file signal is typed, the file will not be deleted and the editing session is terminated. If none of these are typed, the prompt is re-issued.

Edit file does not exist

The file mentioned in a two-file "edit" command as the <edit> file, the file containing the data to be edited, could not be found. The editor will terminate immediately. This behavior is different from the MC6809 versions of the UniFLEX Text Editor. That version ignores the second file name of the "edit" command and treats the command as though it were a single file name command where the file name mentioned does not exist.

Empty text buffer

This indicates that the text buffer is empty (contains no text) and the requested command could not be completed.

Error attempting to open file

The file specified in a "write" command could not be opened for writing. This usually means that the specified file could not be created because the path to the file was inaccessible, or the permissions on the directory in which the file was to reside exclude the user from creating a file there, or the file exists but the user does not have write permission for the file.

Error copying edit file

At the end of an editing session, any unread data on the file that is being edited is copied to the new file being written. An error was detected during this copy process. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor.

Error creating scratch file

The file specified in a "write" command could not be created. This message is preceded by a message indicating which error was detected. The "write" command is aborted and the editor requests a new command.

Error opening scratch file

The file specified in a "read" command could not be opened. This message is preceded by a message indicating which error was detected. The "read" command is aborted and the editor requests a new command.

Error reading data file

The editor detected an error when trying to read from the file being edited or from a scratch file with the "read" command. This message is preceded by a message indicating which error was detected. The current command is aborted

and the editor requests a new command; no data read from the file is kept. If the file being read was the file being edited, it is recommended that the editing session be abandoned with the "abort" command since the file being read is no longer positioned correctly.

Error waiting for task to complete

An error was detected when waiting for a task generated by the 'u' or 'x' command to complete. This message is preceded by a message indicating which error was detected. The command is aborted and the editor requests a new command.

Error writing new file

The editor detected an error when trying to write the contents of the edit buffer to the file that holds the updated information. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor. All changes to information still in the edit buffer are lost.

File is a directory

An attempt was made to edit a directory, not a text file. This is a fatal error and causes an immediate exit from the editor. This message occurs only at the beginning of an editing session.

File name?

This is the prompt used when the editor requests a file name. Commands that may request a file name are "read" and "write". The editor will also request a file name in response to the "stop" and "log" commands if no file names were specified as arguments to the editor when it was called.

Input error

An error status was returned by the UniFLEX Operating System in response to a request for input from the standard input device. This is normally the terminal keyboard and should not generate any such error. If the standard input has been redirected to a disk file, an error may be generated when reading the disk for input characters. In either case, this is a fatal error and causes an immediate exit from the editor. All changes to information still in the edit buffer are lost.

Line too long

The maximum size for a line being input to the editor is 255 characters. Lines in the file being edited may be of any length, but those entered from the standard input device are limited to 255 characters.

Name too long

The file name entered in response to a "File name:" prompt is too long. The maximum size of a file name, including the path specification, is 254 characters.

New file being created

This is an informative message indicating that there is no existing file of information to be edited and that a new file is being created.

New file is the same as the old file

The editor was called with two file names as arguments, but both names point to the same file. Either the file names are the same, or the two files have been linked with the UniFLEX "link" system call.

No child task exists

The "wait" command was used when no background task had been generated by the editor.

No lines deleted

An informative message indicating that the "delete" command was used but the target could not be located. The user answered "no" to the prompt asking if the delete was to proceed.

No such line

A line number or target that could not be found was specified in front of a command.

Not enough room

The file being read with the "read" command could not fit in the available space in the edit buffer. None of the information which was read from the file is kept. The "flush" command may be used to try to make room for the file. If that fails, the file being read should be split into smaller files that may be read individually.

Not found
A target was specified that could not be found.

Output error
An error status was returned by the UniFLEX Operating System in response to a request to output to the standard output device. This is normally the terminal display and should not generate any such error. If the standard output has been redirected to a disk file, an error may be generated when writing the data to the disk file. In either case, this is a fatal error and causes an immediate exit from the editor. All changes to information still in the edit buffer are lost.

Positioning backwards is not allowed
The "position" command was called with a target that has a leading minus sign, indicating a backward search.

Relative positioning is not allowed
The "position" command was called with a target that is an unsigned integer, indicating a relative displacement forward in the file.

Some lines renumbered
An "insert", "replace", or "break" command caused some lines in the file to be renumbered. Note that the "move" and "copy" commands will cause renumbering without this message being issued.

Source overlaps destination
The "copy" or "move" command was issued with arguments such that the target line was within the range of data being copied or moved.

Syntax error
A syntax error was detected in a command.

Target not reached
Are you sure?
The "delete" command was used but the target could not be located. If the user wishes the delete to proceed to the end of the edit buffer, this prompt should be answered with a 'y'. Answering with an 'n' or the end-of-file signal will cause the delete to be aborted.

Task ttt: Abnormal Termination
Interrupt code: i

The background task "ttt" generated by the 'x' command was interrupted before it could complete. The interrupt code returned by the task is indicated by 'i'. This message is returned only in response to the "wait" command.

Task ttt: Abnormal Termination
Termination response: xxx

The background task "ttt" generated by the 'x' command has completed abnormally. The termination response returned by the task is indicated by "xxx". This message is returned only in response to the "wait" command.

Task ttt initiated

Task number "ttt" has been started by the use of the 'x' command.

Task ttt: Normal termination

The background task "ttt" generated by the 'x' command has completed normally. This message is returned only in response to the "wait" command.

Too many file names specified

More than two file names were specified as arguments to the editor. This is an informative message only; the extra file names and any options specified after them are ignored.

Unable to open file

The file specified in a "read" command could not be found or could not be opened for reading because of its permissions.

Unexpected error, edit session aborted

An error response was received from a UniFLEX system call that the editor is incapable of handling. The editing session is terminated immediately.

Unknown option specified

An unrecognizable option was specified when the editor was called. This is an informative message only; the unrecognizable option is ignored.

Write ends with an error

The UniFLEX system reported a media error while the editor was writing data to the file specified in a "write" command.

...zones OK?

A target could not be found and the search zones were not set to their default values. This is an informative message asking the user to check the zones as they may have been the reason that the target could not be found. This message does not require a response from the user.

?

The editor is not able to interpret the given command. Either the command could not be recognized or the format of the command was undecipherable.