XED

V E R S I O N   3 . 2

All rights to the XED software and its documentation are reserved by the authors. Copyright 2016 Larry Fish, Richard Ottosen, and Loren Blaney.

This software and its documentation are provided as-is. There are no warranties, expressed or implied.

CONTENTS

1.0 INTRODUCTION

XED is a screen-oriented text editor that runs on the Raspberry Pi under Raspbian. It's designed for writing programs, but it's also useful for writing any kind of text--like this manual for instance. XED doesn't have fancy fonts or use a mouse, but instead concentrates on being fast and efficient.

XED has features not found in other editors. It can handle files of any size and lines of any length. It handles all ASCII characters including control and IBM extended characters. It works with displays 80 characters by 25 lines and larger. It has automatic wordwrap, and can reformat paragraphs and center lines. It has a sophisticated macro capability that lets you write small programs to do complex editing tasks. It does cut, copy and paste--both line and column oriented. You can draw using box characters. You can restore text that you accidentally delete. XED does many kinds of searches: forward and backward, case sensitive and insensitive, wildcard, search and replace, and search across multiple buffers. It has place markers that let you jump back to a marked location in the text or alternate between two marked locations. Program files can be displayed with highlighted syntax.

Most commands can be executed with a single keystroke, such as Ctrl+A. Common commands are done without moving your fingers from the home-row key position. In fact, the most common commands are done with the left hand, leaving the right hand free to shuffle papers or scribble notes.

1.1 GETTING STARTED

To run XED type this at the command prompt:

```
        xed <filename> <Enter>
```

<Filename> is the name of your file. It can include a path specification. XED tries to open an input file with the name you specify, but if the file doesn't exist, it creates a new file with this name.

You should see something like this:

```
┌──────────────────────────────────────────────────────┐
│                                                      │
│                                                      │
│                                                      │
│                                                      │  <- Text Window
│                                                      │
│                                                      │
│                                                      │
│                                                      │
│|--/+---1---+---2---+---3---+---4---+---5---+---6^-|+│  <- Ruler
│                                                      │  <- Command Line
│ok  Cmd:7  Scoop:98  S#s -> ins  Rem:921 At:109 Col:9│  <- Status Line
└──────────────────────────────────────────────────────┘
```

Text is entered by simply typing. If you make a mistake, correct it with the Backspace key. Use the arrow keys to move anywhere in your text. New typing is inserted by shifting existing characters out of the way.

There are many editing commands, and even the most experienced user has trouble remembering them all. Shift+F1 displays the help screens. <Esc>/<Esc><Esc> prints them out. This makes a handy reference when first learning XED. [Printing is currently not implemented on the Raspberry Pi.]

There are two ways to exit from XED:

```
        <Esc>Q<Esc><Esc>
        <Esc>T<Esc><Esc>
```

The "Q" command is the normal way. This saves your text in the output file. The "T" command exits without saving your text. All commands are case-insensitive so "q" and "t" can also be used.

1.2 THE TEXT WINDOW

Most of the screen is devoted to text. Editing occurs at the cursor, which is a red block. The cursor is usually near the bottom. As it's moved up and down, the text scrolls down and up such that the cursor stays at the same line on the screen.

1.3 THE RULER

The ruler helps orient you. The numbers measure the text columns, for example, "2" indicates column 20. Other characters on the ruler indicate the following:

/	Marks the column where the first line of a paragraph is indented.
|	Marks the columns for the left and right margins. These are often hidden beneath the other characters.
^ or diamond	Mark the location of the bell column. If wordwrap is on, the diamond character is shown. Otherwise, the "^" character is shown.
smiley face	Indicates the location of the cursor within the edit buffer. If this is near the left side, the cursor is near the beginning of the buffer. If this is near the right side, the cursor is near the end of the buffer.

The edit buffer is the available RAM space in your computer. Files larger than this space are loaded a chunk at a time.

1.4 THE COMMAND LINE

The command line is used for macros. Macros are short programs that perform editing tasks. There are 20 command lines. The currently selected one is displayed.

1.5 THE STATUS LINE

The status line displays information about the state of XED. There are ten fields.

```
|ok  Cmd:7  Scoop:98  S#s -> ins  Rem:921 At:109 Col:9  Mod|
└──────────────────────────────────────────────────────────┘
 ^      ^        ^     ^  ^   ^      ^      ^       ^    ^
 1      2        3     4  5   6      7      8       9   10
```

1) Error Field: Displays error information. If the last operation caused an error, a message appears in this field. If there's no error then "ok" is displayed.

2) Command Line Number: Displays the number of the selected command line (0-19).

3) Scoop Free Space: Indicates the number of unused characters available in the scoop. The scoop is a temporary holding place. It's used to scoop up text and move it to another location. It's similar to what other editors call a clipboard that's used for cut, copy, and paste operations.

4) Search Case: Indicates whether search commands are case sensitive. If "S#s" is displayed, searches are case sensitive. If "S=s" is displayed, upper and lower case characters are treated as the same.

5) Search Direction: Indicates the direction of searches. If a right arrow appears, searches move forward from the cursor. If a left arrow appears, searches move backward.

6) Insert/Overwrite Mode: Indicates whether typing is inserted by shifting existing text out of the way or whether the existing text is overwritten. XED is normally used in insert mode. When in insert mode, "ins" appears. When in overwrite mode, a yellow "OVR" appears.

7) Remaining Free Space: Indicates the number of unused character spaces left in the edit buffer.

8) Cursor Location: Shows the cursor location, in characters, from the beginning of the edit buffer.

9) Column Count: Shows the cursor column position from the beginning of the line. The column number takes into account the effect of any tab characters. Note that the left-most column is column zero.

10) Modified: When "Mod" is displayed, this indicates that the text in the edit buffer has been changed.

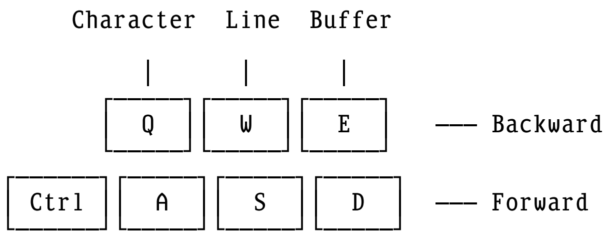
On wide screen displays the name of the file being edited is shown. If the output file name is different than the input file name, it's also shown.

2.0 DIRECT MODE VERSUS COMMAND MODE

XED has two kinds of commands: those that operate directly on the text (such as Ctrl+A) and those that are executed from the command line (such as <Esc>Q<Esc><Esc>). XED always starts in direct mode. Typing <Esc> changes from direct mode to command mode.

2.1 DIRECT MODE

Most of the commands used in direct mode are control characters. The most commonly used commands are positioned close to the left side of the keyboard so that the Ctrl key can be pressed with the little finger while the commands are typed. [This assumes that the Ctrl key and Caps Lock key have been swapped by changing the line in the file /etc/default/keyboard from XKBOPTIONS="" to XKBOPTIONS="ctrl:swapcaps"] This way your fingers never leave the home position. The following diagram shows the location of the main cursor movement keys:



Note that the commands are arranged in pairs. One key moves the cursor backward, while the other moves the cursor forward. As you see, the cursor can be moved a character at a time, a line at a time, or to the beginning or end of the edit buffer.

Years ago IBM moved the Caps Lock key to the Ctrl key position, which changed the key arrangement shown above. You can move it back as explained above.

In XED the line-movement commands are a little different than in most other editors. In most editors the line-movement commands leave the cursor on the same column, but in XED line-movement commands force the cursor to the beginning of the line. This helps orient you as you scroll through your text, but if you prefer that the cursor stays on the same column then use the arrow keys instead.

These commands are available in direct mode:

-- DIRECT COMMANDS --

	MOVE		ADDITIONAL CHARACTERS
^A	Ahead a character	^I	Tab
^Q	Back a character	^L	Form feed (new page)
^N ~N	Ahead to Next word	^U	Indent same as line above
^B ~B	Back a word	^Z	Shift case and move ahead
^S	Down a line	^T	Display control chars (on/off)
^W	Up a line	^^	Always insert next keystroke
^J	Jump ahead a page (PgDn)		SCOOP
^K	Kick back a page (PgUp)	^R	Start (or Resume) a scoop-up
^D	Ahead to end of edit buffer	^F	Finish a scoop-up
^E	Back to start of edit buffer	^Y	Insert scoop and empty it
	DELETE		COMMAND LINE
Bksp	Back a character (^H)	Esc	Enter command-line mode
Del	Ahead a character	^G	Re-execute command line (Go)
^C ~C	Back a word	^V	Copy command line into text
^X	Back a line	^P	Select command line (0-9)
~X	Ahead a line	^]	Change search case sensitivity
		^_	Reverse search direction

^ = Ctrl key   ~ = Alt key   Fn = Function key   SFn = Shifted

-- EXTENDED DIRECT COMMANDS --

F1	Get last command line	SF1	Display help summary
F2	Get next command line	SF2	Display ASCII chart
F3	Turn wordwrap mode on/off	SF3	Show status lines (off/on)
F4	Get next buffer from file	SF4	Highlight syntax (on/off)
F5	Insert scoop	SF5	Insert column scoop
F6	Margin paragraph		
F7	Display current line number	SF7	Scroll text (on/off)
F8	Erase scoop	SF8	Select font size (S/M/L)
F9	Mark location in text		
F10	Move to last marker		
		~R	Start column scoop-up
		~F	Finish column scoop & blank
		~Y	Insert column scoop & empty
^O ~O	Undelete (Oops!)		
Insert	Select insert or overwrite mode		
~D	Draw with box characters: ┼ ╪ ╫ ╬		
^Break	Break out of a command-line loop		

^ = Ctrl key   ~ = Alt key   Fn = Function key   SFn = Shifted

2.1.1 Control Characters in the Text

When control characters are inserted into the text they are usually highlighted. For example, Ctrl+L appears as a black-on-white "L". Some control characters, like carriage return and tab, position the text and don't appear as highlighted characters. Sometimes it's useful to see all the control characters. Ctrl+T highlights all control characters, including tabs and carriage returns. It also displays space characters as small dots so you can easily count them and distinguish them from tabs.

Occasionally you need to insert a control character, such as the bell (Ctrl+G), into your text. But typing most control characters executes a command rather than inserts it. The Ctrl+^ (same as: Ctrl+Shift+6) is used to solve this problem. Any keystroke that follows this command is inserted into the text.

2.1.2 Deleting and Undeleting

The Backspace key is often used for correcting mistakes. It deletes one character to the left of the cursor. The Delete key deletes the character under the cursor, which has the effect of deleting one character ahead. Ctrl+X deletes from the cursor back to the beginning of the line, or if the cursor is at the beginning of the line, it deletes the line above.

XED can "undelete" text that's accidentally deleted. If you delete some text using one of the delete commands (Ctrl+X, Delete, or Backspace), typing Ctrl+O (oops) restores it. You must use Ctrl+O immediately after deleting. If you move the cursor with anything other than a delete command, the text is lost and you cannot undelete it. Multiple deletes, even using different delete commands, can all be undeleted.

2.1.3 Text Markers

Text markers can be used to move back and forth between two locations. This is ideal when you need to compare two areas of text.

You can put an invisible marker in the text at any time by hitting the F9 key. You can move to another location and place a second marker by typing F9 again. Only two markers are active at one time. If you hit F9 a third time, the oldest marker is moved to the current position.

If you hit F10, the cursor moves to either the most recently set marker, or if you just moved to a marker, the cursor moves to the other marker.

2.2 COMMAND MODE

Command mode lets you write small programs that edit text. These programs are called macros. Commands can be combined in many ways to perform almost conceivable any editing task. Groups of characters can be searched for, then deleted, and new characters inserted. The operation can be repeated throughout the text. As stated before, the <Esc> key is used to move from direct mode to command mode. It's also used to separate certain commands on the command line. Typing two <Esc>s at the end of the command line executes it. This is a little confusing, so we'll go through it step by step.

When in direct mode, typing <Esc> puts you into command mode. An inverse video "[" and a red cursor appear on the command line indicating that you are in command mode. Typing another <Esc> executes the empty command line and puts you back into direct mode. Typing a third <Esc> takes you back into command mode. As you see, <Esc> takes you back and forth between direct mode and command mode.

In command mode, characters typed on the keyboard appear on the command line. <Esc> is used to separate commands. After the commands are entered, two <Esc>s are typed to execute them, and XED returns to direct mode. The command line remains, and it can be re-executed from direct mode by typing Ctrl+G. Reentering command mode clears the previous command line. You can leave command mode, back to direct mode, at any time by typing a Ctrl+X, since this deletes the command line. You also can leave command mode by using the Backspace key to delete all the characters on the command line.

XED has 20 command lines. You can select one and execute it at any time. Typing Ctrl+G re-executes the currently selected command line.

The function keys F1 and F2 are used to select different command lines. F1 decrements the command-line number and F2 increments it. In this way a single keystroke can be used change and examine each command line.

The Ctrl+P command also can be used to select a new command line. When you type Ctrl+P, a cursor appears on the status line over the command-line number. Type the number (0-9) of the command line you want.

To facilitate the use of multiple command lines, XED cycles between them. Hitting <Esc>, causes the command-line number to increment before the new command line is entered. This prevents the old command line from being overwritten. This forms the command lines into a circular buffer. Many times, you want to save the last command line. This way, you have a record of the last command lines (10-19) you've executed.

XED starts with automatic command-line cycling on. If you use the Ctrl+P command to select a specific command line, automatic cycling is turned off. Hitting the F1 or F2 keys to decrement or increment the command line turns command-line cycling back on.

Any command that works in direct mode also works in command mode. For example, Ctrl+A moves the cursor one character forward in both direct and command mode. If you use Ctrl+^ to insert Ctrl+X into the command line, it works like Ctrl+X in direct mode and deletes a line in the edit buffer.

There are commands available in command mode that cannot be done in direct mode. These extra commands are shown here:

-- COMMAND-LINE COMMANDS --

	DELETE
D	Back a character
R	Rest of line, including CR
E	Erase scoop
Z	Zap back to form feed (^L)
K	Kill entire edit buffer

	INSERT
I	String
X	Scoop
H	Character by its Hex value

	FILE
G	Get next buffer from file
A	Append from input file
W	Write to output file
O	Open input and output files
C	Close output file

	QUIT
T	Terminate and discard text
Q	Quit and save text

	MISCELLANY
/	List help summary on printer
L	List edit buffer on printer
M	Set left, right margins or bell
Y	Center line
V	Copy text into command line
J	Re-execute command line (Jump)
P	Execute command line (0-9)
Esc	Command separator
Esc Esc	Execute command line
{ }	Enclose a loop
;	Exit a loop
#	Repeat command indefinitely

-- EXTENDED COMMAND-LINE COMMANDS --

	EXPRESSION (sets variable)
( )	Enclose an expression
[ ]	Execute if variable is not 0
$	Hex constant
^	ASCII constant
%	The variable
@	Character under cursor
K	Keystroke (waits)
Z	Space remaining in buffer
.	Cursor location in buffer
C	Cursor column position
+ -	Add, Subtract
* /	Multiply, Divide
\	Remainder
& !	And, Or
= #	Equal, Not equal
> <	Greater than, Less than
>=	Greater than or equal
<=	Less than or equal

	THE VARIABLE
BA	Insert it as an ASCII character
BH	Insert it as a Hex number
BD	Insert it as a Decimal number
BN	Show on status line in decimal
BS	Show on Status line in hex
BT	Set it to scooped decimal value
BX	Set it to scooped hex value
:	Move to location in variable
_	Set loop count with variable

	SEARCH
S	For a string
	^W = Wild character
	~W = Word separator
N	Entire file (N edit buffers)
F	Find and replace
+ -	Search forward or backward

^ = Ctrl key   ~ = Alt key   Fn = Function key   SFn = Shifted

2.2.1 Executing Commands Multiple Times

All the commands available in command mode can be preceded by a number. The number causes the command to be executed that many times. For example, "5D" deletes five characters back from the cursor. If there's no number, or if it's zero, the effect is the same as 1.

Sometimes you want to execute a command over and over until the command fails. For example, you might want to search for and replace some string throughout the edit buffer. You could use a very large number, but the symbol "#" indicates that a command is to be executed repeatedly until it fails. For example, to find the final occurrence of the word "frog" type:

```
        <Esc>#sfrog<Esc><Esc>
```

2.2.2 Editing Command Lines

Macros are normally entered on the command line by typing them directly. However, sometimes when the macro is complex, it's easier to edit it using the capabilities of XED. There are two commands that let you do this. The first command, "V", is itself executed from the command line. It copies text from the edit buffer into the command line. This way, you can create a macro in the edit buffer then copy it into the command line for execution. The "V" command copies text starting at the cursor. It copies forward from the cursor to the first double <Esc>. (You can insert <Esc> into the edit buffer by typing Ctrl+^ followed by <Esc>.)

XED also lets you move macros that are already in the command line into the edit buffer by typing Ctrl+V. This can be used to modify or repair a macro that's in a command line.

2.2.3 Function Keys

The following command-line commands are used so frequently that they're duplicated on the function keys:

FUNCTION	ACTION	EQUIVALENT COMMAND
F4	Get next buffer from file	G
F5	Insert Scoop	X
F6	Remargin paragraph	M
F8	Erase scoop	E

3.0 SEARCHING

XED has several search commands that let you quickly scan for strings of characters. The basic search command looks like this:

```
        <Esc>Sanimals<Esc><Esc>
```

If you enter this on the command line, it searches forward from the cursor to the first occurrence of "animals". If you type Ctrl+G, it's re-executed, and the next occurrence of "animals" is located.

You can do several types of searches. For example, searches normally go forward from the cursor, but you can set the search to go backward by typing Ctrl+- (Ctrl+dash, but not on the keypad). Typing Ctrl+- a second time, changes back to forward searches. The search direction is indicated on the status line by an arrow. The arrow points to the right if the search is forward, and points to the left if the search is backward.

You also can force a search to be forward or backward by placing a "+" or "-" in front of it. For example:

```
        <Esc>-Sfrog<Esc><Esc>
```

This forces a backward search for the string "frog". This feature is useful with macros when you don't know the direction that was last set by the Ctrl+- command.

XED lets you change the case sensitivity of searches. Normally, searches are case sensitive, which means that uppercase and lowercase letters are treated as different characters--uppercase "A" does not match lowercase "a". Typing Ctrl+] turns off case sensitivity so, for example, searching for "COWBOY" also locates the string "Cowboy". The case sensitivity is shown on the status line. If "S#s" is shown, searches are case sensitive, and if "S=s" is shown, they are not.

Sometimes you want to search for a string, but the match doesn't need to be exact. For example, you might want to locate a string like "the high was 78 degrees", but the temperature could be any number. The "wildcard" character, Ctrl+W, is used to do this. When Ctrl+W is placed in a search string, it matches any character. As a result, the search string: "the high was <Ctrl+W><Ctrl+W> degrees" finds the string no matter which (two-digit) temperature was used.

If you want to search for a short word, such as "an" or the variable name "I" then you want to search for it as a whole word. This way you skip instances like "and" and "many" or "LINE" and "If". Placing Alt+W in a search string matches any non-alphanumeric character. So for example, the search string: <Alt+W>I<Alt+W> finds only instances of the variable "I".

The above search commands only search the current edit buffer. If you have a file that's larger than the edit buffer, you can search your entire file using the "N" search command. This works the same way as the "S" search, but when it reaches the end of an edit buffer, it automatically writes the current buffer to the output file, and then loads the next buffer and searches it. The search continues until either the string is found or the end of the file is reached. The "N" search should only be done in the forward direction.

Oftentimes you want to search for one string and replace it with another. The "F" command is like a search except that it replaces the string with a new string. The syntax of this command is:

```
        F<string1><Esc><string2><Esc>
```

XED searches for <string1> and, if found, replaces it with <string2>. The two strings do not need to be the same length. If the second string is empty, XED simply deletes the first string.

3.1 EXAMPLES

Here are some example command lines.

```
        <Esc>sprint<Esc>i line<Esc><Esc>
```

This searches forward for the word "print", and inserts a space and the word "line" immediately after it.

```
        <Esc>ftheir<Esc>they're<Esc>j<Esc><Esc>
```

This command line changes every occurrence of the word "their" to "they're".

```
        <Esc>S@<Esc>DRI<Enter><Esc>J<Esc><Esc>
```

This macro can be used to delete all comments from an assembly language program. It searches for each occurrence of a "@" character, deletes it and the rest of the line, then inserts a new carriage return to replace the one deleted by the "R" command. This is repeated until the end of the edit buffer is reached.

4.0 MARGINS

XED has several ways to format text. The first is the margin command. This formats a paragraph so that the margins are aligned to a specified column. The "M" margin command can be used with one, two or three arguments. If three arguments are used, the left and right margins and the paragraph indenting are set. For example:

```
        <Esc>M8,64,4<Esc><Esc>
```

The first argument is the left margin, the second is the right margin, and the third is the number of spaces to indent the first line of a paragraph. The third argument is optional. Entering the arguments sets the positions; it does not remargin a paragraph. Remargining is accomplished by entering the "M" command without arguments or by hitting F6.

As an example, if you select margin settings of M8,64,4 then hit F6, the paragraph at the cursor is formatted so that the left margin is eight spaces from the left side of the screen, the right margin ends in column 64, and the first line of the paragraph is indented four spaces, or 12 spaces from the left side. Once you have entered the margin specification, you can format other paragraphs. If you "margined" using the "M" command without arguments, you can margin other paragraphs using Ctrl+G to re-execute the command line. Hitting F6 margins a paragraph without requiring a command line.

This next text formatting feature is called "wordwrap". It automatically inserts a carriage return at the end of each line as you type it. This saves you from having to hit Enter. Wordwrap occurs when you are typing characters and the cursor reaches the right margin. At this point XED searches backward for the first space. If it finds one, it replaces it with a carriage return. If no space is found within 16 characters, no wordwrap occurs until a space is typed. Wordwrap only occurs on space characters.

Wordwrap mode is turned on and off with the F3 key. XED starts with wordwrap turned off. Striking F3 turns it on. Striking F3 again turns it off. When wordwrap is turned on, the up arrow (^) marker on the ruler (which indicates the bell column) is changed to a diamond.

When XED is in wordwrap mode it inserts spaces, if necessary, in front of each new line. This makes the wordwrapping consistent with the margin command.

Another text formatting feature is the bell column. When the cursor reaches the bell column, the computer beeps, warning you that the line might be too long. This is similar to the bell on a typewriter. The bell column is set with the "M" command. If you use the "M" command with a single argument, it sets the bell column. For example:

```
        <Esc>M65<Esc><Esc>
```

This sets the bell column to the 65th column from the left side of the screen.

Ctrl+U is a direct mode command that's used to indent the current line to the same column as the line immediately above it. It's frequently used by programmers to indent blocks of code.

The final text formatting command, Esc Y, is used to center a line of text. This is useful for titles. Space characters are inserted in front of the line so that it's evenly centered between the left and right margins.

5.0 THE SCOOP

The scoop is used to pick up segments of text and move them to other locations. Words, phrases, paragraphs, and even large blocks of text can be moved easily. The scoop also can be used to duplicate the same block of text over and over. The scoop is similar to what other editors call a clipboard used for cut, copy and paste operations.

5.1 LINE SCOOPS

With XED in direct mode, the scoop is turned on by typing Ctrl+R and turned off with Ctrl+F. When the scoop is on, text can be pulled into it by moving the cursor backward over the text. Any backward movement of the cursor, including deletes, scoops up text. You can delete and scoop at the same time.

While the scoop is on, moving the cursor forward causes the equivalent amount of text to be removed from the scoop. Be careful! Generally, the scoop should be turned off before resuming normal editing.

The text that's in the scoop can be inserted into the edit buffer at any time. Executing an "X" command on the command line or F5 from direct mode, causes the entire contents of the scoop to be inserted into the edit buffer at the cursor. The text remains intact in the scoop until it's erased with an "E" on the command line or by hitting F8. Many copies can be inserted into different locations in the edit buffer.

Text from the scoop also can be inserted into the edit buffer using Ctrl+Y. This inserts the text and erases the scoop, which is useful when moving many separate pieces of text.

You can scoop up to 99,999 characters, but the free space in the edit buffer can be as small as 10,000 characters. As a result, you can easily scoop a block that's too big to insert, so you must be careful. If you delete as you scoop, there's always enough space to insert the block.

When you have a file that's larger than the edit buffer and you need to edit it in several stages, you could run into a problem trying to carry a large scoop full of text from one buffer to another. XED protects you from this by checking the size of the scoop contents when it reads in the next text buffer. It always leaves a free space that's the size of the scoop contents plus about 1000 characters. This way you can always insert the scoop when you move to the next buffer.

5.2 COLUMN SCOOPS

It is often useful to take a block of text and insert it into the middle of another block of text. This works fine using the above scoop commands when you want to insert between lines, but it doesn't work at all when you want insert between columns. So XED provides a second way to insert text. Each line of the inserted text starts at the same column. If some of the lines you are inserting into are shorter than the specified column, then these short lines are extended with spaces to allow each inserted line to start in the same column. Any tabs on the line are taken into account.

This feature works like the line-insert command. You simply move the cursor to a location in the text and press Shift+F5. The block of text is inserted with each line starting at the cursor column. Here's an example:

```
-------------------------
| This is a small piece |
| of text that can be   |
| inserted anywhere.    |
-------------------------

SEE THE QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE -------------------------QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE | This is a small piece |QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE | of text that can be   |QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE | inserted anywhere.    |QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE -------------------------QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE QUICK BROWN FOX JUMP OVER THE LAZY DOG
```

As you see, the small block of text has been inserted into the middle of the larger block. It was scooped up the normal way with Ctrl+R some Ctrl+Ws and Ctrl+F, then the cursor was moved to the desired position where the block was inserted with Shift+F5.

It is also useful to cut blocks of text out of the middle of a larger block. The normal scoop command will not do this (except by picking up the whole block, copying it somewhere, trimming away any excess on the left and right sides, and then picking it up again).

The column scoop command lets you to pick up any rectangular block of text. The commands to do column scoops are similar to line scoops. You simply move the cursor to the upper-left corner of the rectangular area you want and press Alt+R. You then move the cursor to the lower-right corner and press Alt+F. Alt+F causes the text in the block defined by the two cursor positions to be copied into the scoop. Here's an example that illustrates the kind of block that can be scooped:

```
SEE THE QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE QU -------------------------  LAZY DOG
SEE THE QU |K BROWN FOX JUMP OVER T|  LAZY DOG
SEE THE QU |K BROWN FOX JUMP OVER T|  LAZY DOG
SEE THE QU |K BROWN FOX JUMP OVER T|  LAZY DOG
SEE THE QU -------------------------  LAZY DOG
SEE THE QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE QUICK BROWN FOX JUMP OVER THE LAZY DOG
SEE THE QUICK BROWN FOX JUMP OVER THE LAZY DOG
```

If you start at the lower-right corner of the block to be scooped instead of the upper-left corner then when the block is scooped with Alt+F, it will be replaced with space characters in the edit buffer. This is similar to what other programs call a "cut" operation.

If you want to copy a block into the edit buffer without shifting the existing text to the right, as is often the case with tables and drawings, then use overwrite mode (see 6.2).

6.0 EXTENDED ASCII CHARACTERS

XED supports the IBM PC's extended ASCII character set. It can display most of the special graphic characters available on the PC. The PC has symbols for the characters with values of 128 through 254. It also has symbols defined for control characters (1 through 31). XED can display all the symbols for the characters from 128 through 254. It does not display the symbols for control characters because it's more important to be able to recognize the control character than it is to view these symbols.

The complete set of ASCII characters can be displayed by typing Shift+F2. This also provides a convenient way to insert an extended ASCII character into your text. Simply type its hex code. For example, to insert the character "¼", type Shift+F2 followed by "AC".

6.1 BOX CHARACTERS

You can make nice looking charts and drawings using the box characters shown here:

```
        ┌───┬───┐   ╒═══╤═══╕   ╓───╥───╖   ╔═══╦═══╗
        │   │   │   │   │   │   ║   ║   ║   ║   ║   ║
        ├───┼───┤   ╞═══╪═══╡   ╟───╫───╢   ╠═══╬═══╣
        │   │   │   │   │   │   ║   ║   ║   ║   ║   ║
        └───┴───┘   ╘═══╧═══╛   ╙───╨───╜   ╚═══╩═══╝
            1           2           3           4
```

One way to get these extended ASCII characters is to use Shift+F2, but there's an easier way. Press the Alt+D key to enter box character mode. The center box character (┼) will appear in the middle of the status line:

```
ok        Cmd: 0  Scoop: 31971   S#s > ins ┼ Rem: 28342   At: 4220    Col: 0
```

In this mode, the numeric keypad (with Num Lock enabled) is used to type the box characters as follows:

```
┌─────╥─────╥─────╥─────┐
│7 ┌  ║8 ┬  ║9 ┐  ║+ │  │
╞═════╬═════╬═════╣     │
│4 ├  ║5 ┼  ║6 ┤  ║     │
╞═════╬═════╬═════╬═════╡
│1 └  ║2 ┴  ║3 ┘  ║     │
╞═════╩═════╬═════╣     │
│0 ─        ║     ║     │
└───────────╨─────╨─────┘
```

There are four sets of box characters consisting of combinations of single and double lines. Each time you press Alt+D, you select the next set. Pressing Alt+D five times sequences through the four sets of box characters and back to the normal numeric characters (whereupon the box character disappears from the status line).

6.2 INSERT AND OVERWRITE MODES

XED has two modes for entering text: insert mode, and overwrite mode. Insert mode is the normal mode. When XED is in this mode, existing characters are shifted to the right before a new character is inserted. This way no text is lost as new text is added. In overwrite mode, the character under the cursor disappears whenever a new character is typed.

XED starts in insert mode. Hitting the Insert key changes between insert and overwrite mode. When XED is in insert mode, the letters "ins" appear on the status line, and when it's in overwrite mode, the letters "OVR" appear on the status line. Because overwrite mode can be dangerous, the "OVR" symbol is outlined in yellow to alert you.

Overwrite mode is useful for making diagrams and drawing box characters. You can fill an area with spaces then type in the appropriate text. In overwrite mode the text won't shift.

When in overwrite mode, inserting the scoop overwrites the existing text. This can be useful in some instances, and a disaster in others. XED protects you as much as possible from overwrite errors. For example, it does not overwrite a carriage return. Also, it does not overwrite if the character you enter is a carriage return. This way, entering blank lines does not overwrite existing text.

The Backspace key works differently in overwrite mode. In overwrite mode, Backspace replaces the deleted character with a space. This way, the text doesn't shift. To prevent extraneous spaces from accumulating at the beginning of a line, Backspace does not replace carriage returns with a space--it just deletes them.

7.0 LINES

7.1 CARRIAGE RETURNS AND LINE FEEDS

XED automatically handles two conventions for terminating lines. The Windows convention terminates lines with a carriage return followed by a line feed, while the Linux convention terminates them with just a line feed.

When a file is read into the edit buffer, line terminators are changed to carriage returns. When the file is written back out, the carriage returns are converted back to the original termination method: either a carriage return converted to a line feed or a carriage return followed by a line feed. If a file is created from scratch (no input file) then the Linux convention is used.

7.2 COUNTING LINES

The line number that the cursor is on is displayed at the left end of the status line when the F7 key is pressed. Lines are counted from the beginning of the file, not just from the beginning of the current edit buffer. The open command "O" resets the line count, even if you are using multiple buffers. The first line is line 1.

7.3 LONG LINES

Some editors have trouble handling extremely long lines of text. This usually occurs when trying to edit some sort of hex loader file that has no carriage returns. XED handles lines of unlimited length. A long line appears as a single line. By moving the cursor forward or back, the line scrolls left or right.

To facilitate handling long lines, the Ctrl+W and Ctrl+S commands limit how far they move the cursor. These commands normally move backward or forward one complete line. However, if you have just one extremely long line, moving up or down a line would take you to the beginning or end of the text, making these commands useless. To deal with this, XED limits the number of characters that the cursor moves to 255. This lets you step through a long line 255 characters at a time.

8.0 FILE HANDLING

As previously explained, XED is normally executed from a command line by typing something like:

```
        xed tax.let
```

This opens an input file named tax.let and, when you exit from XED, creates a new copy of this file that contains any changes. XED normally creates backup files while it works. This protects you from losing data. XED never overwrites the input file. Instead, it does all writing to a temporary file. When you exit, the original input file is renamed to a .bak file, and the temporary file is renamed to match the original file. This way, there's always an unmodified copy of your file.

Normally, the input and output files have the same name. Sometimes however, it's useful to read from one file and write to another with a different name. You can do this by typing two file names after XED when you start it. For example:

```
        xed tax.let tax2015.let
```

This causes XED to read from tax.let and to write to tax2015.let.

When a file is bigger than the edit buffer, XED lets you edit the file in pieces. You do this by working your way through the file, from start to finish, one buffer at a time. This way XED can edit files that are bigger than memory.

When you start XED, it reads as much information from the input file as it can. If the file is larger than the edit buffer, it reads enough information to fill the buffer, leaving about 10,000 characters of free space for insertions. The "G" command from the command line or F4 from direct mode writes the current buffer to the output file, and reads the next section of the input file into the edit buffer. This way, you can work through a file of any size. Because of the backup file feature, you cannot move backwards from one buffer to another, only forwards.

8.1 FILE MANIPULATION

Most of the time, you edit a document using only the simplest file handling commands. In fact, most of the time, the only file handling command you use is "Q", which saves your file and returns to the OS. However, sometimes it's useful to move pieces of text from one file to another. XED has several command-line commands that can be used to manipulate files.

The first command is the "A" append command. The append command reads text from the input file. If the edit buffer is empty, the command fills the buffer, leaving space for at least 10,000 more characters. If the edit buffer is nearly full, the append command reads in only one line of text each time it's executed. All information read by the append command is inserted at the cursor. If you want the append to be inserted at the end of the edit buffer, the cursor must be at the end of the edit buffer.

XED has a feature that helps you deal with large files. Ordinarily when a file is too large to fit into a single edit buffer, you edit the file sequentially, one buffer at a time. But occasionally, the information you want to work with falls right on the border between two buffers. One way to solve this problem is to use the "A" command to append a few more lines into the current buffer. Unfortunately, every line you append decreases the free space available in the buffer.

This feature works like the "A" append command except that it writes one line out of the buffer for every line it reads in. This way, you can move the whole buffer forward a line at a time while preserving the free space. In effect, the whole buffer scrolls one line at a time through the text.

There are two commands that are used to move through the file this way: Ctrl+Page_Down and Ctrl+End. Ctrl+Page_Down moves the buffer forward one line at a time. Ctrl+End moves forward one page at a time. Each time one of these keys is struck, one line or one page from the beginning of the buffer is written to the output file and deleted from the buffer, and one line or one page is read into the end of the buffer. Since the lines written and deleted probably won't be the same size as the lines read into memory, the amount of free space will expand and contract accordingly. [This feature is currently not implemented.]

The next command is the "O" open command. The open command is used to reopen the current file or open a brand new file. For example:

```
        <Esc>O<Esc><Esc>              - Reopen the current files

        <Esc>O"cat.txt"<Esc><Esc>     - Open a new file called cat.txt
```

In the first example, the open command is used with no argument. This reopens the current input and output files. Reopening resets their pointers to zero. In effect, you are starting over with these files, and any changes that were written to the output file are discarded. This is dangerous, but it can be useful as long as you realize that you are starting over. Normally, this command is used with the "A" append command described above.

The second example illustrates opening a new file. If the "O" command is followed by a file name in quotes, a new file is opened. Again, both the input and output file pointers are reset, so that any information that was written to the output file is lost. However, information in the edit buffer and in the scoop is preserved making it possible to move information from one file to another. You can even open a series of files and combine pieces of information from each.

As an example, lets say you want to move a block of text from frog.fil to cat.fil. You begin by editing frog.fil normally:

```
        xed frog.fil
```

Next, scoop up the text you want then delete the rest of the buffer using the kill command:

```
        <Esc>K<Esc><Esc>
```

Now open and append cat.fil using the following command:

```
        <Esc>o"cat.fil"a<Esc><Esc>
```

You can now insert the text in the scoop anywhere you want in cat.fil. When you are done, exit normally using the "Q" command.

You can use this technique to move through any number of files picking up pieces of text as you go. Just keep opening new files until you have everything you need. When you are ready, open the destination file and insert the text into the proper location, then exit using the "Q" command. The information goes into the last file you edit.

8.2 CHANGING TERMINALS

Linux, being a multiprocessing system, lets you switch to another terminal at any time, even while editing a file. For example, if you press Alt+F2, it switches to a new screen where you can login and run a completely independent session. Here, you can even run XED on other files (but don't run it on the original file or you may lose changes). Pressing Alt+F1 takes you back to the original editing session. You must press another key, such as an arrow key, to refresh the screen. There are six of these virtual terminals available by pressing Alt+F1 through Alt+F6.

9.0 ADVANCED MACROS

9.1 COMMAND-LINE LOOPS

Loops repeatedly execute a group of commands on the command line. They are similar to a "for" loop in a program. Loops are created by enclosing a group of commands inside a set of braces "{ }". If a number is placed in front of the braces, every command inside the braces is executed that number of times. If an error is detected within the loop, the loop aborts immediately without executing any more commands between the braces. For example, if you are doing a search and the search fails, the loop stops before executing the rest of the commands inside the loop. Here's an example:

```
        <Esc>10{SJuly <Ctrl+W><Ctrl+W>, <Esc>I2015<Esc>}<Esc><Esc>
```

This command line searches for any date in July and adds "2015" after it. It executes 10 times or until it fails to find the search string.

Loops can be nested. For example:

```
        <Esc>20{10{Sfrog<Esc>Sdog<Esc>}Ipig<Esc>}<Esc><Esc>
```

This command does 10 searches for a complex string then inserts the string "pig". It does this whole operation 20 times.

If you use a "J" command within a loop the "J" causes the execution to jump to the beginning of the loop, not to the beginning of the whole command line.

Errors cause the current loop to abort. If a loop is nested or if there are other commands on the command line after the loop, XED continues to execute these commands. When there's an error within a loop, XED clears the error as soon as it exits from the loop so it can continue executing commands outside the loop. This is useful for things like searching for the last occurrence of a string. For example, this locates and marks the innermost "begin-end" pair in a block of code:

```
        <Esc>#{Sbegin<Esc>}I *<Esc>Send<Esc>I *<Esc><Esc>
```

Assuming you have a single procedure in the edit buffer, this command repeatedly searches for "begin". When the search fails, the cursor is located just after the last "begin", which is the innermost "begin". The code then places an asterisk after the "begin" and then searches for the next "end". This "end" is the corresponding innermost "end", and an asterisk is placed after it.

9.2 EXPRESSIONS AND CONDITIONALS

9.2.1 Expressions

XED can do complex editing tasks using expressions and conditional operations. The syntax is relatively simple.

Expressions are a series of mathematical or Boolean operations that, when evaluated, result in a single numeric value. XED uses expressions to test for special situations, like a unique combination of characters.

Expressions are always enclosed in parentheses "()". Expressions consist of two kinds of items: factors and operators. A factor is something that has a value, such as the number 2. Operators operate on factors, like the plus sign operates on 2 + 3. Here's a list of the factors and operators available in XED:

FACTORS

123	Any decimal number
$	Returns the value of the hex number that follows
^	Returns the ASCII value of the next character
%	Returns the value of the Variable (see 9.2.2)
@	Returns the value of the character under the cursor
K	Waits for then returns the value of a keystroke
Z	Returns the size of the free space in the edit buffer
.	Returns the cursor location in the edit buffer
C	Returns the column number of the cursor position

OPERATORS

+	Add
-	Subtract
*	Multiply
/	Divide
\	Remainder of a divide
&	Bit-wise Boolean AND
!	Bit-wise Boolean OR
=	Compare for equality
#	Compare for not equal
>	Compare for greater than
<	Compare for less than
>=	Compare for greater than or equal
<=	Compare for less than or equal

Note that all these factors and operators can only be used in an expression. In other words, they can only be used inside parentheses. Some of these characters have different meanings outside expressions. For example, "C" means "return the column number of the cursor" inside an expression and "close the file" outside an expression.

When XED evaluates an expression, it reduces all the operators and factors to a single numeric value. For example, the expression: (1+2+3) reduces to the value 6. Sometimes an expression is used as a Boolean value. In other words, sometimes the expression is reduced to a single true or false value. For example:

```
        (@=^A)
```

In this expression, if the character under the cursor is "A", then the expression evaluates to "true". In XED, "true" is the value -1, or any non-zero value, and "false" is 0.

9.2.2 The Variable

The result of the evaluation of an expression is stored into a special place called the "Variable". There's only one variable, and every time an expression is evaluated, the old value is overwritten by the new value.

You also can set the Variable with a number that's in the scoop. The number is converted from its ASCII representation. For example, the ASCII string "193" is converted to the numeric value 193. There are two commands that perform this conversion:

BX	Interprets text as hexadecimal
BT	Interprets text as decimal

These commands search from the beginning of the scoop for a decimal or hexadecimal string (which must be within the first 80 characters or a zero will be returned). The resulting converted number is stored in the Variable.

9.2.3 Conditionals

Conditionals are used to selectively perform an editing operation. Conditional operations are indicated with square brackets "[]". Commands enclosed in square brackets are only executed if the value of the Variable is true. For example:

```
        <Esc>#{<Ctrl+A>(@=$1B)[<Ctrl+A>D]}<Esc><Esc>
```

This command line searches for escape characters ($1B)--something that normally cannot be done--and deletes them. The command line starts by moving the cursor forward one character. If the cursor is on an escape character, the commands in the square brackets are executed. These commands move the cursor forward one character and then delete the escape. The "#" repeats the entire command until the end of the buffer is reached.

Conditionals can be nested, or placed inside loops. They cannot have a numeric value in front of them like other commands. For example, this would cause an error:

```
        <Esc>234[Ifrog<Esc>]<Esc><Esc>
```

9.2.4 Output Commands

XED has several commands to display or insert values:

BA	Inserts the Variable as an ASCII character
BD	Inserts the Variable as a decimal number
BH	Inserts the Variable as a hex number
BS	Shows the value of the Variable on the status line in hex
BN	Shows the value of the Variable in decimal

Three of these commands insert values into the edit buffer. As you will see in the examples below, this is useful for creating a series of numbers in your text. The "BS" command can be used to display the ASCII value of any character in your text.

9.2.5 Looping

If you combine expressions and conditionals with loops, you can create some powerful macros. To facilitate the use of conditionals with loops, XED has the semicolon ";" command. This forces a loop to terminate at the point it's encountered. This provides a way to terminate a loop based on a condition. For example:

```
        <Esc>#{<Ctrl+A>(@=$1B)[<Ctrl+A>D;]}<Esc><Esc>
```

Like above, this command searches through the text for an escape character ($1B); but when it finds one, it deletes it, and then stops.

The command iteration count is normally set by putting a number in front of a command. For example:

```
        <Esc>10Inow is the time<Esc><Esc>
```

This inserts ten copies of "now is the time" into the buffer. You also can set the loop iteration count using the Variable. This enables the number of iterations to be calculated with an expression or to be set by a condition. The underline character "_" copies the value of the Variable into the iteration count. Since the value of the Variable is set by the value of the last expression evaluated, any command can be executed a calculated number of times. For example, the following command can be used to pad a string with spaces:

```
        <Esc>S<Enter><Esc><Ctrl+Q>(8-C)<Ctrl+W>_I <Esc><Esc>
```

This command right-justifies a string that's up to eight characters long. The string is assumed to be on a line by itself. The first part searches for the carriage return that terminates the line. The Ctrl+Q backs up to the last character of the string. The column position is then subtracted from 8. This gives the number of spaces needed to right justify the string. The cursor is then moved to the beginning of the line, and the proper number of spaces are inserted.

9.2.6 Cursor Movement

This command moves the cursor to a location in the edit buffer depending on the value of an expression. The colon command ":" moves the cursor to the location specified by the Variable. For example, this moves the cursor 20 locations ahead if the character under the cursor is an "a".

```
        <Esc>(@=^a)[(.+20):]<Esc><Esc>
```

9.2.7 Examples

The following examples illustrate the power of expressions and conditionals.

Display the value of the character under the cursor in hex:

```
        <Esc>(@)BS<Esc><Esc>
```

Search for the first occurrence of a numeric character:

```
        <Esc>#{<Ctrl+A>(@>=$30 & @<=$39)[;]}<Esc><Esc>
```

Search for the word "the" and if the keystroke is "+" then delete it:

```
        <Esc>Sthe<Esc>(K=^+)[DDD]<Esc><Esc>
```

List the numbers 0 through 99 on the left side of the screen:

```
        <Esc>(0)100{bdi<Enter><Esc>(%+1)}<Esc><Esc>
```

Count the number of "a" characters beyond the cursor, and display the number on the status line:

```
        <Esc>(0)#{sa<Esc>(%+1)}bn<Esc><Esc>
```

Move to the 81st column on the next line with more than 80 columns:

```
        <Esc><Ctrl+S>#{<Ctrl+A>(C>80)[;]}<Esc><Esc>
```

Display the hex code and character for the full ASCII character set:

```
        <Esc>(0)256{BHI - <Esc>BAI<Enter><Esc>(%+1)}<Esc><Esc>
```

9.2.8 Break Command

It is possible to write macro commands that are infinite loops. You can also write commands that, although they are not infinite, take hours or days to complete. XED lets you exit from these loops by typing Ctrl+Break. When you hit Ctrl+Break to abort a loop, the message "break" appears on the status line.