Bedit Editor

Bedit Overview

The Bedit editor is a text editor designed to work with MS Windows 3.1. It also works with Windows NT and WorkGroup for Windows. Bedit is intended to be a powerful, feature rich editor that is consistent with Windows standards while being highly configurable. Features were selected from other editors found in Windows and in the UNIX operating system. These features and others were integrated together to form an editor that can be configured to be similar to some already familiar screen oriented editors, while maintaining consistency with Windows and being intuitive and powerful. In particular, users of the AT&T vi editor will find the Bedit editor pleasing indeed.

Files of almost unlimited size can be edited, but each line cannot exceed 1022 bytes. The only limit on the file size is the amount of "virtual" memory available to Windows. In most PCs this is at least 2 Megabytes.

The Bedit editor can open a file for editing in several different ways. These are all described under the Bedit Operators. However, two methods should be discussed here as well. First, when the Bedit editor is first run (executed), files may be passed as parameters. For example, to run Bedit, initially opening two files called file1 and file2, run:
bedit.exe file1 file2
In this way, the Bedit editor will come up and open file1 and file2 for editing. Alternatively, once the Bedit editor is running, files can be dragged and dropped onto the running editor. Files dropped onto Bedit will be opened just as if the Open File Operator were used. The details of how you can use drag and drop are dependent on which file manager is used on your PC.

The Bedit editor reads text from a file into high speed memory (RAM). The text is then modified by the Bedit editor in memory and is later written back to the file (saved) under user control. The Bedit editor bases most of its functionality on "operators". An example of an operator is the Copy Operator. The Copy Operator, like many of the Bedit operators, likes to operate on selected text. That is, if text is selected before executing the Copy Operator, the selected text is copied to a buffer. Later this text can be copied from the buffer to a selected location in the file text. The Bedit operators are described individually in Bedit Operators.

The Bedit editor depends on a configuration file to define its available functions and how these functions are associated with keyboard or mouse input. That is, the Bedit operators are assigned to key combinations and to "buttons" in the configuration file. The configuration file also defines settings for several editor options. Sample configuration files are distributed with Bedit as a starting point. One of these configuration files may be found satisfactory as shipped. If not, modifications can be made easily. The configuration files are normal (ASCII) text files, allowing the use of any text editor to customize them. The format and content of the configuration file are described in Bedit Configuration.

The Bedit editor has two modes. The mode most common for Windows editors is called Text Entry Mode. While in Text Entry Mode, all normal text is entered into the file as it is typed: just like on an old typewriter. The Bedit editor also has a Command Mode. While in Command Mode, all key strokes are interpreted as commands to execute Bedit Operators. Thus, while in Text Entry Mode the letters a through z are entered into the file, but in Command Mode these letters are interpreted according to the configuration file. Most of the key combinations on a modern keyboard do not represent text. Such key combinations include the control keys and the function keys. These key combinations are not entered into a file and are interpreted as commands to execute Bedit Operators in either mode. Command Mode provides a convenient way to execute most Operators with a single key stroke, and, in general, increases the number of Operators that can be easily and directly

utilized from the keyboard.

The Bedit editor has a button bar under the menus and above the edit windows. The buttons on the button bar are selectable with the mouse. Each operator has a unique looking button which can be placed anywhere on the button bar as assigned in the configuration file (for Professional Bedit). The use of these buttons is unaffected by the modes of the Bedit editor, and provide an easy way to execute Bedit Operators without touching the keyboard.

The Bedit keyboard input can be combined into "macros" that can, in turn, be assigned to key combinations. These macros can be very helpful when doing repetitive editing and can perform a series of Bedit editor operations. Macros are a common feature among the more powerful editors. However, the Professional Bedit editor can have up to 127 macros and each macro can be assigned to a key combination which remains in effect in future editing sessions. For example, the Symantec Deskedit (TM) editor has a function that repositions the current line in the center of the screen. Although the Bedit editor does not have that function, a Bedit macro can be recorded to perform exactly the same function. Thus, even in the unlikely event that the Bedit editor does not have your favorite functions as operators, you can easily customize it into a familiar, yet more powerful, Windows editor. A macro can also be modified (edited) after it has been recorded. This is not only useful when a mistake is made recording a macro, but also when small changes to an existing macro are desired during its use. For more information about Bedit editor macros, see Macros.

Two levels of the Bedit editor may be purchased. These are "Normal" and Professional. Normal Bedit may be purchased for $35. For $35 you receive a registration key that will eliminate the reminder screens. We can also send you a manual for $5 plus shipping and handling (which is also $5 within the continental US). We will send you a diskette with or without the manual for shipping and handle ($5). Thus, the prices for shareware Bedit registration are:
$35 - Registration key only
$40 - Registered copy of Bedit on a diskette
$45 - Registered copy of Bedit on a diskette with a manual

Professional Bedit costs $100. Professional Bedit comes with a disketteand a manual with no additional shipping and handling (within the continental US). A Normal Bedit editor can be upgraded to Professional Bedit for $65. State sales tax of 3% applies only to sales in Colorado. Call (303) 660-1796. You can pay by Visa or MasterCard, or you can send a check or money order to:

J & B Star Software
P.O.Box 878
Castle Rock, CO 80104

If you already have a registered copy of the Bedit editor, please call (303) 660-1796 if you have any questions, comments, or suggestions. We can also be reached on CompuServe with user ID 72640,3347 or on America Online with user ID "Bedit". An unregistered copy of Bedit can be found on shareware Bulletin Boards such as CompuServe's WINSHARE or WUGNET. (E.g. GO WUGNET). Please distribute the unregistered Bedit software to anyone who may be interested in a high quality, powerful, and configurable text editor for Windows 3.1+. Use of an unregistered Bedit editor for more than 30 days is illegal.

It is illegal to distribute a registered or Professional copy of the Bedit software. It is legal to use a registered copy of the Bedit editor only on one PC at a time.

Bedit Configuration

The Bedit editor configuration descriptions given here are most easily understood if you are viewing a sample configuration file. Two sample configuration files are distributed with the Bedit editor. These sample configuration files are called beditcfg.win and beditcfg.unx. The beditcfg.win configuration file is intended for those who are most familiar with another Windows text editor while the beditcfg.unx file is intended for those more familiar with the AT&T vi (TM) editor. The primary configuration file is called bedit.cfg. You should view or edit one of these files while reading about Bedit editor configuration files.

The Bedit editor configuration is specified in a text file called bedit.cfg. This file describes how the editor will function. It specifies initial states (most of which can be changed while running the editor). It also specifies the actions the editor should take when keys are typed. This is the most important aspect of the configuration file. In most editors the meaning of a particular key is always well defined and constant. For example, in the vi editor, while in Text Entry Mode the escape key always puts the editor into Command Mode. While in Command Mode, vi will save the text to the disk file when :w<Enter> is typed. The Bedit editor can be configured to interpret these keys in the same way (see the beditcfg.unx configuration file). However, the meanings of the keys can be defined to be whatever combination is most familiar or desirable to an individual. In fact, redundant key assignments may be most desirable, so that the same functionality may be obtained in several different situations by using different key combinations.

We recommend that one of the sample configuration files be used initially. Then the Bedit.cfg file can be edited to customize the Bedit editor. The structure and content of the configuration file are described in this help topic. After the configuration file has been changed, the Bedit editor must be restarted for the changes to take effect. In other words, the Bedit editor functions based on the configuration file as it was when the editor began to run and is unaffected by later changes to the file.

The configuration file and the help file are normally in the same directory as the Bedit.exe file. This directory should be in the PATH environment variable. It is possible to move either Bedit.cfg or Bedit.hlp to a directory not in the PATH by setting an environment variable as follows. The configuration environment variable is BEDITCFG and the help environment variable is BEDITHLP. Do not include the file names in the path given in these variables. The file names are always Bedit.cfg and Bedit.hlp.

Some examples may help to clarify how the configuration file works. The Bedit editor has a Find String Operator that can be used to find and select (highlight) text. The Find String Operator is assigned to the ^s (or Ctrl+s) key combination in the sample configuration file beditcfg.win by the following line:

```
^s = FINDST
```

With this configuration, whenever the s key is pressed while the control key is held down, the string search operator is "executed". Similarly, to assign the escape key to place the Bedit editor in Command Mode (from Text Entry Mode) as described above, the beditcfg.unx configuration file includes the assignment:

```
ESCAPE = CMDMODE
```

Bedit operators can also be assigned to normal keys such as the letters a through z. However, while in Text Entry Mode, these keys are used to enter text and therefore any operator assignments to these keys are ignored until the editor is put in Command Mode. Thus, if an operator (such as find string) should be available from Text Entry Mode, the operator must be assigned to a key combination not normally used for text entry (such as control s). A complete list of available key combinations is given later in this section.

Operators can only be executed using associated key combinations when there is an active edit window. For example, the Bedit editor Exit Operator can be assigned to function key F12 as follows.
F12 = EXIT
Then F12 can be used to exit a Bedit editing session, but only if there is at least one active edit window. If all edit windows have been closed, another method must be used to exit the editor. For this reason, we recommend that the Exit Operator be assigned to a button on the button bar. Button bar assignments are described later in this section.

The operators are listed under Bedit Operators. Each operator lists a "Config Name" which is used in the configuration file key assignments. Each assignment has the form:
<key combination> = <operator>
Choose a key combination from the table below and an operator "Config Name" from the Bedit Operators help pages. An operator can be assigned to any number of different key combinations, but of course each key combination must have no more than one operator assigned to it. Comments can be placed in the configuration file by preceding them on the same line with the number sign (#). A key combination can be specified in multiple ways. ^a can also be specified as Ctrl+a. ^A can also be specified as Ctrl+Shift+a.

The standard motion keys on an extended keyboard are not assignable. Their actions are fixed to maintain Windows standards. These keys are:
Up arrow: move caret up a line
Down arrow: move caret down a line
Left arrow: move caret left in line
Right arrow: move caret right in line
Ctrl+left arrow: move caret left a word in line
Ctrl+right arrow: move caret right a word in line
Home: move caret to beginning of current line
End: move caret to end of current line
Ctrl+Home: move caret to beginning of file
Ctrl+End: move caret to end of file
Page Up: display previous page of file
Page Down: display next page of file
Ctrl+Page Up: move caret to first line of current page
Ctrl+Page Down: move caret to last line of current page

Some other key combinations used by Windows are:
Ctrl+F4: Closes the current file
Ctrl+F6: Moves to the next file (if more than one file is open)
Ctrl+Tab: Moves to the next file (if more than one file is open)
Any combination with F10: opens the File menu

Many keyboards have a number pad with a NumLock key. When the NumLock light is not lit, the NumPad keys have the labeled insert, delete, and motion functions except the NumPad/, NumPad*, NumPad-, and NumPad+ keys. These can be given assignments in the Bedit configuration file that will be active while the NumLock light is not lit whether in Command or in Text Entry Mode. When the NumLock light is lit, the keys on the number pad have the labeled numeric functions, where NumPad0 is a 0, NumPad1 is a 1, etc. and NumPad/ is a slash (/), etc. The initialization parameters used in the configuration file are as follow:

Initialization Parameters:

Parameter	Explanation
BufferSize = 5	# Buffer size to use for files

```
InitCmdState = Cmd          # Entry if Text Entry Mode, Cmd for Command Mode
InitShiftWidth = 4          # width in characters of a displayed tab
InitAutoIndent = False      # True or False
InitOverStrike = False      # True or False
InitMatchCase = True        # True or False
InitMatchWholeWord = False          # True or False
InitWordWrap = True         # True or False
InitWrapCol = 68            # word wrap column
InitAutonextline = True     # Initial check box state (macro iteration)
InitWatchChanges = False    # Initial check box state (macro iteration)
MaxUndoes = 200             # Max levels of undo/redo
BringUpWithFiles = True     # True or False
AutoSaveTime = 10           # autosave time in minutes (0 is never save)
Backups = True              # True or False
BackupChar = $              # First char of backup file extension
```

The BufferSize parameter specifies how text is buffered. This is given in KB and ranges from 4KB to 10KB. The only advantage to small buffers (e.g. 4KB) is slightly faster text entry. Larger buffers have many advantages, including: 1) ability to display more long lines in an edit window and 2) less frequent screen flashing. On a 486 or faster machine, the 10KB buffer size is recommended. On a 386 the text entry speed may be more important and the 4KB size is recommended.

The Bedit editor can be configured to have one or two states. To be an editor, it must have the Text Entry Mode to allow keyboard text entry. The editor may also have a Command Mode which simplifies some editing and increases the number of easily used operators. The InitCmdState specifies whether the Bedit editor will begin in Text Entry Mode or in Command Mode. No matter which state is initially chosen, Bedit Operators may be used to switch between Command and Text Entry Modes. For more on this see the Command Mode Operator and the Insert Operator.

InitShiftWidth specifies the initial apparent size of a tab character. The value of 4 specifies that a tab will be the size of 4 average characters. This value is also used to determine the number of characters to shift lines using the Bedit Shift Operators. The shift width can be changed for the current editing session by selecting the Options menu, then the Shift Width menu item. The Set Shift Width Operator can also be used to change the shift width for the current editing session.

InitAutoIndent, if True, tells the Bedit editor to automatically indent new (empty) lines to match that of a previously existing line. AutoIndent can be changed for the current editing session by selecting the Options menu, then the Auto indent menu item. The Autoindent Operator can also be used to change AutoIndent for the current session.

Many MS Windows editors have two types of text entry. One is insert mode, where text is only replaced if it is selected (highlighted). The other mode is "overstrike", where the character immediately following the caret is replaced by the next typed character. InitOverStrike sets the initial state for Text Entry Mode. This state can be changed later from the Options menu or with the Overstrike Operator .

The Find String operator uses the InitMatchCase and InitMatchWholeWord settings for initial values in the dialog box it uses. If True, InitMatchCase implies that a search for AbC will not match abc. Otherwise, AbC will match abc. If True, InitMatchWholeWord implies that a search for the string ack will not match the ack in the word back. Otherwise it will match.

The Bedit editor can automatically begin a new line when a maximum column is reached while doing text entry. This is called Word Wrap, since the last word typed is "wrapped" to the next line. Word Wrap is initially on if InitWordWrap = True. Word Wrap can be turned

on or off by selecting the Options menu, then the Word Wrap menu item. The column at which words are wrapped is specified by InitWrapCol. Word Wrap can also be turned on or off using the Word Wrap Operator.

The macro iteration dialog box has a check box option called "Auto next line". If InitAutonextline = True, the initial state of the check box is checked. The default is True. See Macros for more information.

InitWatchChanges determines the initial state of the "Watch Changes" check box of the macro iteration dialog box. When this box is checked during macro iteration, the changes made are displayed as they are made. This is nice for some purposes, but slows down the macro iteration by approximately a factor of ten!

You can "undo" changes that are made to the file text during a single edit session using the Edit menu Undo or the Undo Operator. More than one change can be undone in succession. After changes have been undone, the Redo Operator can be used to remake the same changes. For example, if you replace a character, this can be undone by executing Undo. If you search for and replace every occurrence of a string in the file, this can also be undone with a single execution of Undo. In Professional Bedit the default for the maximum number of successive undo (or redo) operations that can be performed is 200 (MaxUndoes = 200). Since computer memory is used to keep track of undo/redo information, we recommend that, for machines with minimal amounts of RAM (i.e. 2MB), you keep MaxUndoes less than 1000. However, with more than 8MB of RAM and at least 8MB for a Windows swap file, you should be able to set MaxUndoes as large as you like. MaxUndoes can be set to 0 to provide unlimited undo/redo or to any number from 2 to 2,147,483,647. One command normally results in one level of undo/redo: including a complex Command Line command, a "Replace All", or iteration of a macro. Thus, 1000 levels of undo/redo represents a large amount of control (usually 1000 Bedit Operators).

When the Bedit editor is brought up (i.e. executed) it can come up with a record of the files that were being edited when the editor was last terminated (i.e. exit). This can be very useful if a particular file (or set of files) is frequently edited. BringUpWithFiles specifies whether this option is desired or not.

The Bedit editor can make a backup of a file before saving changes back to the file. If Backups = True, a backup is made the first time a file is saved in each editing session. The default is Backups = False. Saving will be done automatically every AutoSaveTime minutes if this is greater than zero. If AutoSaveTime = 0, no automatic saving is done. "New" files will not be automatically saved until they have been saved manually and given a name. The first character of the extension of the backup file is given by the BackupChar parameter.

ACCESS Assignments:

The Bedit editor provides something called an Access Operator. This can be quite important, depending on how the editor is to be used. The Access Operator provides second level (i.e. two key combination) access to the Bedit operators. That is, after the Access Operator is executed, the next key combination is assumed to have an operator assignment in the configuration file as follows:
ACCESS+<key combination> = <operator>
The beditcfg.unx sample configuration file has the assignments:
^k = ACCESS
ACCESS+w = SAVE
The Access Operator can be thought of as providing a second meaning for key combinations. There is no overhead for unused key assignments, so feel free to assign as many operators to Access key combinations as desired.

Button Bar Assignments:

The Bedit editor supplies a button bar (just below the Bedit menus and above the edit windows). The buttons on this bar can be selected with the mouse to execute Bedit

Operators. The placement of the Bedit Operators on the button bar is configurable only in Professional Bedit. (Other versions of Bedit use a fixed assignment of Bedit Operators to the button bar.) The buttons on the button bar are numbered from 1 to 60. The actual number of buttons that can be used is dependent on the size of the Bedit window and the resolution used in Windows. At a resolution of 1024 X 768, when Bedit is full screen approximately 36 buttons can effectively be used on the button bar. When the Bedit window is smaller or the resolution is coarser (e.g. 800 X 600), fewer buttons will be useable (visible on the button bar). However, any number of Bedit Operators (up to 60) may be assigned to the button bar independent of the number that will actually be visible. When the mouse is over a button, the Operator assigned to the button is indicated on the status line at the bottom of the Bedit editor window. The button for each Operator is shown in the help for that Operator. (Professional Bedit) button bar assignments in the configuration file have the following form:
Button<number> = <operator>
For example, to assign the WORD operator to button 5, the following assignment would be needed in the configuration file (for professional Bedit):
Button5 = WORD

NOTE: the Repeat Change Operator cannot be assigned to the button bar.

Operator Assignable Key Combinations

All key combinations listed in the following table can be used for Operator and Macro assignments that will function while in Command Mode. The table indicates which key combinations are also active while in Text entry Mode.

Key Combo	TextMode	Key Combo	TextMode
a	no	A	no
b	no	B	no
c	no	C	no
d	no	D	no
e	no	E	no
f	no	F	no
g	no	G	no
h	no	H	no
i	no	I	no
j	no	J	no
k	no	K	no
l	no	L	no
m	no	M	no
n	no	N	no
o	no	O	no
p	no	P	no
q	no	Q	no
r	no	R	no
s	no	S	no
t	no	T	no
u	no	U	no
v	no	V	no
w	no	W	no
x	no	X	no
y	no	Y	no
z	no	Z	no
^a	yes	^A	yes

^b	yes	^B	yes
^c	yes	^C	yes
^d	yes	^D	yes
^e	yes	^E	yes
^f	yes	^F	yes
^g	yes	^G	yes
^h	no	^H	no
^i	no	^I	no
^j	yes	^J	yes
^k	yes	^K	yes
^l	yes	^L	yes
^m	no	^M	no
^n	yes	^N	yes
^o	yes	^O	yes
^p	yes	^P	yes
^q	yes	^Q	yes
^r	yes	^R	yes
^s	yes	^S	yes
^t	yes	^T	yes
^u	yes	^U	yes
^v	yes	^V	yes
^w	yes	^W	yes
^x	yes	^X	yes
^y	yes	^Y	yes
^z	yes	^Z	yes
^1	yes	!	no
^2	yes	@	no
^3	yes		
^4	yes	$	no
^5	yes	%	no
^6	yes	^	no
^7	yes	&	no
^8	yes	*	no
^9	yes	(	no
^0	yes	)	no
0	yes		
`	no	~	no
-	no	_	no
=	no	+	no
\	no	|	no
[	no	{	no
]	no	}	no
;	no	:	no
'	no	"	no
,	no	<	no
.	no	>	no
/	no	?	no
^`	no	^~	no
^-	no	^_	no
^=	no	^+	no
^\	no	^|	no
^[	no	^{	no
^]	no	^}	no
^;	no	^:	no
^'	no	^"	no

^,	no	^<	no
^.	no	^>	no
^/	no	^?	no
F1	yes	SHIFT+F1	yes
F2	yes	SHIFT+F2	yes
F3	yes	SHIFT+F3	yes
F4	yes	SHIFT+F4	yes
F5	yes	SHIFT+F5	yes
F6	yes	SHIFT+F6	yes
F7	yes	SHIFT+F7	yes
F8	yes	SHIFT+F8	yes
F9	yes	SHIFT+F9	yes
F11	yes	SHIFT+F11	yes
F12	yes	SHIFT+F12	yes
Tab	no	Shift+Tab	no
Escape	yes	SHIFT+Escape	yes
Ins	yes	SHIFT+Ins	yes
Delete	yes	SHIFT+Delete	yes
Enter	no	SHIFT+Enter	no
Space	no	SHIFT+Space	no
^F1	yes	^SHIFT+F1	yes
^F2	yes	^SHIFT+F2	yes
^F3	yes	^SHIFT+F3	yes
^F5	yes	^SHIFT+F5	yes
^F7	yes	^SHIFT+F7	yes
^F8	yes	^SHIFT+F8	yes
^F9	yes	^SHIFT+F9	yes
^F11	yes	^SHIFT+F11	yes
^F12	yes	^SHIFT+F12	yes
^Ins	yes	^SHIFT+Ins	yes
^Delete	yes	^SHIFT+Delete	yes
^Enter	yes	^SHIFT+Enter	yes
^Space	yes	^SHIFT+Space	yes
NumPad/	yes	SHIFT+NumPad/	yes
NumPad*	yes	SHIFT+NumPad*	yes
NumPad-	yes	SHIFT+NumPad-	yes
NumPad+	yes	SHIFT+NumPad+	yes
^NumPad/	yes	^SHIFT+NumPad/	yes
^NumPad*	yes	^SHIFT+NumPad*	yes
^NumPad-	yes	^SHIFT+NumPad-	yes
^NumPad+	yes	^SHIFT+NumPad+	yes

The Mouse in Bedit

The Bedit editor is intended to be used with a mouse or other pointing device. Some things cannot be done without it, such as some dialog box usage. Furthermore, since the mouse can simplify text editing, the Bedit editor is designed to take advantage of it. The left button of the mouse is used just as with most MS Windows based editors. When the mouse cursor is moved to where the text caret is desired and the left button is pressed and released, the caret is placed at the desired character position in the text. If the mouse is moved while the left button is held down, text is selected (visibly highlighted). The next operator can then affect the selected text to change it, delete it, copy it, etc. Text is also selected if the shift key is held down when a new caret position is selected using the left button. This method of selection is recommended when a large block of text is being selected.

The right mouse button is used to copy or move text. That is, if text is selected in the active Bedit window before the right mouse button is pressed, the selected text will be unhighlighted and copied to the Clipboard. If the Ctrl key is held down while the right mouse button is pressed, the selected text will be unhighlighted, copied to the Clipboard, and then deleted (the text is "moved"). If no text is highlighted when the right button is pressed, the text in the clipboard is pasted where the mouse cursor is at the time. This is always true with or without the Shift and Cntl keys held down. If text is selected and the Shift key is held down when the right button is pressed, the text in the clipboard is pasted, replacing the selection. These methods provide effective ways of copying or moving text from one place (or file) to another without touching the keyboard. In fact, significant editing can often be done with the mouse alone. A tabular listing of these mouse functions follows:

Table of Mouse Functions

Function	Mouse Method
copy	Right button + selected text
paste at	Any+Right button + no text selected
paste over	Shift+Right button + selected text
cut	Ctrl+Right button + selected text

Since text searches are needed so frequently, the Find String Operator can be executed by pressing the middle mouse button on a 3 button mouse.

The mouse can be used to execute operators that are assigned to buttons on the button bar in the configuration file. See Configuration File for details of the configuration. In this way, text can be selected, cut, copied, saved, found, shifted, etc. without touching the keyboard.

The mouse has many other uses such as menu command selection, minimizing or maximizing windows, scrolling by clicking in the scroll bars to the right or at the bottom of the window, etc. Some experimentation is recommended to see how useful the mouse can be.

The Bedit Status Bar

The Bedit editor maintains a status bar at the bottom of the main Bedit window. Various types of information are displayed on the status bar to assist the user. While in Command Mode, file status information is maintained as discussed below. While in Text Entry Mode, the file status is updated periodically or, for example, when mouse selections are made. Other types of information are also presented on the status bar for macro recording and button bar assignments as discussed below. But first, the file status information:

The first two entries in the status line are the line and column numbers. These show the location of the beginning of the current selection or the position of the caret.

After the Col number, in parentheses, is the hexadecimal code of the character following the caret. This will mostly be useful to programmers or others who need to know the character representation in the file.

The next part of the status line shows "Enter: TEXT" or "Enter: CMDS". This indicates whether the Bedit editor is in Text Entry Mode or in Command Mode, respectively.

Following this mode indicator is the "command count" in parentheses. This applies to Command Mode Operator execution. Many of the Bedit Operators take a count preceding the operator to represent a line number, column number, repetition count, etc. This is also known as the command count.

While in Text Entry Mode, the Bedit editor can be in either insert or in overstrike mode. The next part of the status line shows "Emode: INS" or "Emode: OVR". This indicates the method of text entry when the editor is in Text Entry Mode.

The next part of the status line may be the most useful for many people. It indicates what the Bedit editor expects the user to type next. This field is "Type: Anything" when the editor does not expect any special input. However, when the next input will be interpreted in some special manner, this will indicate what is required. For example, "Type: Same key or motion" appears in this field for many operators to indicate that the next entry will be interpreted in the context of the operator that was just executed.

The last three entries in the status line indicate the size and status of the current file. The size is shown both in lines and in bytes. The status of the file is the last modification date and time of the file if the file has not been modified since it was last saved. If the file has been modified and not saved, this simply displays "Modified".

Other status information is sometimes displayed on the status bar when appropriate. When a macro is being recorded, this fact is displayed on the status bar before the number of lines in the file.

Another importantuse of the status bar is for the buttons. When the mouse is moved over a button, the operator assigned to the button is displayed on the status bar. Thus, even if you cannot remember the meaning of a button symbol, you can find the correct button by simply moving the mouse cursor over the buttons.

Yet another use of the status bar is while entering a string using the Set Find String Operator. As the new find string is being entered, it appears on the status bar until the Enter key is used to end the string entry.

Bedit Macros

A Bedit editor macro is a recorded sequence of key combinations that can be played back. Playing a macro is equivalent to typing the sequence of key combinations again. For convenience, when the macro includes Find Operators, if a Find fails, the playback is stopped. A macro can be used to simplify repetitive editing, reducing boredom and increasing productivity. Macros can also be used to customize the Bedit editor (as discussed later). First a macro is recorded and named. Then the macro can be assigned to a key combination so that when that key combination is typed, the macro is played. This is just like using key combinations to execute Bedit Operators. If a somewhat different sequence of key combinations is desired, Bedit allows the macro to be edited. Professional Bedit allows 127 macros to be defined simultaneously. Other versions of the Bedit editor allow 2 macros to be defined simultaneously.

A macro can be recorded using menu commands or using the macro operators. The Record Macro menu item will begin recording a macro. The Start Record Operator performs the same function. While recording a macro, this fact is displayed on the status bar. When the desired sequence of key combinations has been entered, the menu item Stop Rec Macro can be used to stop recording the macro. The Stop Record Operator performs the same purpose as the menu item.

When recording is stopped, a dialog box is brought up to allow naming the macro and assigning the macro to a key combination. At that time, the macro must be given a unique name, but it need not be assigned to a key combination. A macro can be played without being assigned to a key combination by using the Play Macro menu item. At a later time, the Assign Macro menu command or the Macro Assign Operator can be used to assign a macro to a key combination. A macro can be assigned to any number of different key combinations in this manner. You can cycle through the key assignments for a macro by selecting the macro in the list multiple times in succession.

As with Bedit Operators, a key combination associated with a macro is only effective when there is an active edit window. Thus, if all files are closed and a key combination is then entered, no macro or Bedit Operator will be executed.

When a macro is assigned to a key combination, iteration can be specified. That is, when the macro is played using the assigned key combination (with iteration), a dialog box is brought up to specify a line number range and maximum number of iterations to play the macro. The first and last line numbers in the line number range can be given simply as numbers or if they have been marked using the Mark Operator the mark letter may be used to specify the first and or last line number in the range. If the "Watch Changes" box is checked, all changes made by iterating the macro will be shown when they are made. (This slows down the execution of the macro by approximately a factor of ten!) If the "Auto next line" box is checked, each iteration of the macro will begin on a later line than the previous iteration. The following discussion should clarify this:

The first iteration of the macro will always begin in the first line specified in the dialog box. If the caret is already in this first line, the macro is played with the pre-existing selection. Otherwise, the selection is changed to column 1 of the first specified line. For each subsequent iteration of the macro, if the caret is in a later line than for the previous iteration, the selection is not changed. If the "Auto next line" box is not checked, the selection is not changed no matter where the caret is (within the allowable line range). However, if the "Auto next line" box is checked, the selection is changed to column 1 of the next line before beginning to play the macro again. In each iteration of the macro, the macro is played as it was recorded. When the line number exceeds the maximum line number or the specified maximum number of iterations have been done, the macro iteration

ends. An example may also help to clarify this.

> Suppose the macro (called "stuff") goes to the beginning of the current line, moves one word to the right and inserts the number 0. Suppose stuff is assigned to function key f2 with iteration. When f2 is pressed on the keyboard, a dialog box comes up to specify the line number range where the macro can execute and the maximum number of iterations. For example, the first line number could be set to 5 and the last (maximum) line number to 10, with a maximum of 5 iterations and "Auto next line" checked. Then stuff will be played on lines 5, 6, 7, 8, and 9. If the maximum number of iterations had been set to 6 or more, stuff would also have been played on line 10. If stuff had changed lines by itself (in the forward direction), Bedit would not change the position of the caret between iterations even with "Auto next line". Since stuff did not change lines and "Auto next line" was checked, each iteration after the first one begins at column 1 in the next line.

As you can see, the "Auto next line" feature is mostly intended for macros that are written to affect one line at a time. It is often more convenient to write a macro without needing to be concerned about moving to the next line. However, some macros are intended to affect text in the vacinity of other text, whether the relevant text is in the same line or not. For these macros, the "Auto next line" box should not be checked.

While iterating a macro, clicking the left mouse button anywhere on the screen interupts the macro execution. This is sometimes very useful when you say "Oops!".

Several Bedit editor features are particularly useful in macros. These include the Set Find String Operator and the Unselect Operator. The Find String Operator brings up a dialog box that can be inconvenient when iterating a macro. The Set Find String Operator sets the string to be found (by a Find Next or Find Previous Operator) without using a dialog box. (String entry is ended with Enter while recording or by \n while editing a macro.) This allows searches in either direction through the text for several different strings within a macro. The Find Operators select the text that is found. A macro may need to make changes starting from the beginning of the selected text. For example, you may need to paste text after the first character in the selected (e.g. found) text. The Unselect Operator can be used to unselect text, leaving the caret at the beginning of the (previous) selection. This can simplify macros.

If a macro must be changed or deleted, the Macro Maintenance menu command or the Macro Maintenance Operator can be used to edit the macro. This brings up a dialog box that allows macros to be edited or deleted. A mouse or other pointing device is required when editing a macro. After edit changes have been made to a macro, it must be "Saved" before selecting another macro or selecting "Done". Otherwise the changes are lost. (NOTE: Saving edit changes for a macro does not save the macro to disk.) When a macro is deleted or unassigned, all key assignments associated with that macro are changed back to their normal configured assignments. For example, the Find String Operator might be assigned to ^s (control s) in the configuration file. A macro could be recorded and assigned to ^s. Later the macro could be deleted or unassigned, resulting in the Find String Operator again being assigned to ^s.

Professional Bedit saves all existing macros to disk when Bedit is closed (exited). The macro save file is named bedit.mac in the Windows directory. Professional Bedit restores previously saved macros from the bedit.mac file when it begins to run (executes). Other versions of Bedit maintain up to 2 macros for the current edit session only.

Some macros are included with the Professional Bedit editor. These macros provide additional vi emulation as well as examples of macros. These macros include (but may not be limited to) the following.

Supplied Macros

Assigned Key	Macro Purpose
0	Place caret before first column of line
p	Paste (put) text after the next character
A	Append text at the end of the current line
I	Insert text at the beginning of the current line
Ctrl+w	Delete the word to the left of the caret

The functions performed by the above macros are not available as operators, but these macros together with the provided key assignments increase the functionality of the Bedit editor.

Bedit Operators

Access
Append
AutoIndent
Back a Word
Back Big Word
Buffer
Cascade
Char Delete Left
Char Delete Right
Char Delete RAny
Close All
Close File
Command Line
Command Mode
Copy
Copy Line
Char Replace
Cut
Cut Line
Down
End
End Big Word
End Line
Exit
Export
Export As
File Insert
Find Char
Find Char Back
Find String
Find Next
Find Previous
Goto a Line
Goto Beginning of Line
Goto Marked Char
Goto a Column
Goto Marked Line
Help
Icon Arrange
Import
Insert
Join
Left
Macro Assign
Macro Maintenance
Mark Position
Match
New
New Next Line
New Previous Line
Next Line
Open File
Overstrike
Page Bottom

Page Down
Page Home
Page Middle
Page Up
Paste
Previous Line
Print
Quote
Redisplay
Redo
Repeat Change
Repeat Find Char Left
Repeat Find Char Right
Replace
Replace Line
Reposition
ReRead
Right
Save
Save All
Save As
Scroll Down
Scroll Up
Select
Select All
Set Find String
Set Font
Set Shift Width
Shift Left
Shift Right
String Replace
Start Macro Record
Stop Macro Record
Substitute Chars
Tile
Toggle Case
Undo
Unselect
Up
Word Right
Word Big Right
Word Wrap

Bedit Operators

Operator	Button	Brief Description
Access		Access other operators
Append		Text Entry Mode after this character
AutoIndent		Toggle autoindent on and off
Back a Word		Move caret back one normal word
Back Big Word		Move caret back one white space separated word
Buffer		Use a Bedit buffer with an Operator such as Copy
Cascade		Arrange edit windows in an overlapping cascade
Char Delete Left		Delete the character to the left on this line
Char Delete Right		Delete the character to the right on this line
Char Delete RAny		Delete the character (or EOL) to the right
Close All		Close all open files (Bedit edit windows)
Close File		Close the current file
Command Line		vi command line emulation
Command Mode		Change to Command Mode
Copy		Copy text to clipboard or buffer
Copy Line		Copy line(s) to clipboard or buffer
Char Replace		Replace one or more characters
Cut		Cut text to clipboard or buffer
Cut Line		

 Cut line to clipboard or buffer

Down

 Move caret down one or more lines

End

 Move caret right to the end of normal word(s)

End Big Word

 Move caret right to the end of big word(s)

End Line

 Move caret right to the end of this line

Exit

 Exit the Bedit editor, closing files as needed

Export

 Save to the current UNIX file

Export As

 Save to a UNIX file, specifying the name

File Insert

 Replace selected text with contents of a file

Find Char

 Find a char to the right in this line

Find Char Back

 Find a char to the left in this line

Find String

 Find a string

Find Next

 Repeat string find downward in file

Find Previous

 Repeat string find upward in file

Goto a Line

 Go to specified (or last) line number

Goto Beginning of Line

 Go to 1st non-white char in line

Goto Marked Char

 Go to marked char position

Goto a Column

 Place caret at specified column in this line

Goto Marked Line

 Go to marked line

Help

 Display help for last Operator

Icon Arrange

Arrange any ICONs in the Bedit window

Import

Open a UNIX file

Insert


Change to Text Entry Mode

Join

Join this line and the next one

Left

Move caret to the left

Macro Assign

Assign a macro to a key combination

Macro Maintenance

Change or delete macros

Mark Position

Mark a position in the file

Match

Match (,), {, }, [, or]

New


Open a new empty file

New Next Line

Create a next line, change to Text Entry Mode

New Previous Line

Create a previous line, change to Text Entry Mode

Next Line

Move caret to 1st nonwhite space char in next line

Open File


Open a file

Overstrike

Toggle between insert and overstrike modes

Page Bottom


Move caret to bottom of page

Page Down


Display next page of text

Page Home


Move caret to top of page

Page Middle


Move caret to middle of page

Page Up

 Display previous page of text

Paste

 Paste text from clipboard or buffer

Previous Line

 Move caret to 1st non-white space char in previous line

Print

 Print selected or all text of file

Quote

 In Text Entry Mode next char is entered literally

Redisplay

 Refresh display of current file

Redo

 Do again what was just undone (i.e. undo of undo)

Repeat Change Make same text change here

Repeat Find Char Left

 Find char left again

Repeat Find Char Right

 Find char right again

Replace

 Search for and replace strings

Replace Line

 Change one or more lines (Text Entry Mode)

Reposition

 Change location of current line as displayed

ReRead

 Ignore modifications and read file from disk

Right

 Move caret right

Save

 Save current file to disk

Save All

 Save all open files to disk

Save As

 Save current file to disk, specifying name

Scroll Down

 Display next half page of text

Scroll Up

 Display previous half page of text

Select

 Select text without affecting anything else

Select All

 Select all text without affecting anything else

Set Find String

Set string to be found (for next find command)

Set Font

Change the font used to display the current file

Set Shift Width

Set the shift width (tab size)

Shift Left

 Shift text left by shift width chars

Shift Right

 Shift text right by shift width chars (using tabs)

String Replace

 Delete (clear) specified text, change to Text Entry Mode

Start Macro Record

Start recording a new macro

Stop Macro Record

Stop recording a new macro

Substitute Chars

 Delete next character(s), change to Text Entry Mode

Tile

 Display edit windows tiled: non-overlapping

Toggle Case

 Change case of next or selected text

Undo

 Undo last text change

Unselect

 Unselect any current text selection

Up

Move caret up one line

Word Right

Move caret to beginning of next normal word

Word Big Right

Move caret to beginning of next big word

Word Wrap

Toggle word wrap on and off

Operator: Access

Config Name: ACCESS

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Access operator provides access to any other operators. That is, the key typed after the Access operator determines the next operator (assuming appropriate assignments in the configuration file). This is useful when you wish a key to have two meanings. For example take the assignments in the beditcfg.unx sample configuration file. The Access operator is assigned to Ctrl+k and the Word Operator is assigned to the w, but you want to be able to save the file using w. The access assignment for w is assigned to the Save Operator so that Ctrl+k w writes (saves) the file. Thus, the w has a different meaning after the Access Operator.

Well over 100 key combinations can be used for operator assignments that can be used while in either Command Mode or Text Entry Mode. (See Configuration File for allowable key combinations.) The Access Operator increases access to other operators from either Command Mode or Text Entry Mode.

Most computer users have difficulty remembering a large number of arbitrary key assignments, but find it easier to remember mnemonic assignments such as w for write, c for change, e for end, etc. Since there are a limited number of available easy to remember assignments, the Access Operator provides a means to effectively double the use of each key combination. This is the primary purpose of the Access Operator.

The configuration assignments described above are found in the beditcfg.unx sample configuration file:

^x = ACCESS
w = WORD
ACCESS+w = SAVE

Operator: Append

Config Name: APPEND

vi Emulation: a

Button for button bar: 

Description:
The Append Operator places the caret after the next character in the current line and enters Text Entry Mode. If it is preceded by a count, the Append Operator duplicates the entered text "count" times at each point where text is entered until Command Mode is changed to.

See Also: Insert, Command Mode

Operator: **Autoindent**

Config Name: **AUTOINDENT**

vi Emulation: **Not A vi Command (See Command Line Operator)**

Button for button bar: 

Description:
The Autoindent Operator toggles auto indent. When auto indent is on, each new line is indented the same as the previous line. (For the New Previous Line Operator indent is the same as the line the caret was originally on.) Autoindent can also be turned on and off with the Command Line Operator using "set ai" and "set noai", respectively.

See Also: **Command Line Operator**

Operator: Back

Config Name: BACK

vi Emulation: b

Button for button bar: 

Description:
The Back Operator moves backwards one (or more) words separated by white space or non-alphanumeric characters. If this operator is preceded by a count, the caret is moved the specified number of words backwards. Otherwise, the caret is moved one word backwards.

See Also: Backbig, Word, Wordbig

Operator: Backbig

Config Name: BACKBIG

vi Emulation: **B**

Button for button bar: 

Description:
The Backbig Operator acts like the Back Operator, but bases its motion on white space separated words. If this operator is preceded by a count, the caret is moved the specified number of white space separated words backwards.

See Also: Back, Word, Wordbig

Operator: Buffer

Config Name: BUFFER

vi Emulation: "

Button for button bar: 

Description:
The Bedit editor provides 26 buffers in addition to the Clipboard to store strings. A buffer is referenced by a letter from a through z. The Buffer Operator is the means for storing and retrieving text from/to these 26 buffers. Execute the Buffer Operator, then enter a letter to specify the buffer. These buffers can be used in conjunction with the following operators:

1) Copy
2) Copy Line
3) Cut
4) Cut Line
5) Paste

Example: Using the beditcfg.unx sample configuration file, to copy the current line to the Bedit editor buffer referenced by the letter z, you would type:

"zY

The Buffer Operator is assigned to the " key and the Copy Line Operator is assigned to the Y key (Shift+y) so that this says to use a buffer, the buffer to use is z, and the text to copy to buffer z is the current line.

See Also: Copy, Copy Line, Cut, Cut Line, Paste

Operator: Cascade

Config Name: CASCADE

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Cascade Operator arranges the Bedit windows in an overlapping cascade.

Operator: **Char Delete Left**

Config Name: **CDELETEL**

vi Emulation: **X**

Button for button bar: 

Description:
The Char Delete Left Operator deletes (clears) one (or more) characters to the left of the caret, but remains on the same line if no text is selected. A count preceding the operator specifies a number of characters to delete. If text is selected, the selected text is deleted.

See Also: **Char Delete Right**, **Char Delete Right Any**

Operator: Char Delete Right

Config Name: CDELETER

vi Emulation: X

Button for button bar: 

Description:
The Char Delete Right Operator cuts one (or more) characters to the right of the caret, but remains on the same line if no text is selected. The characters are cut into the Clipboard. A count preceding the operator specifies a number of characters to delete. If text is selected, the selected text is deleted.

See Also: Char Delete Left, Char Delete RAny

Operator: **Char Delete Right Any**

Config Name: **CDELETERANY**

vi Emulation: **Not A vi Command**

Button for button bar: 

Description:
The Char Delete Right Any Operator deletes one (or more) characters to the right of the caret, including the end of line sequence. That is, this operator does not "stay on the same line", but will delete any number of characters.

See Also: **Char Delete Left Operator**, **Char Delete Right Operator**

Operator: **CloseAll**

Config Name: **CLOSEALL**

vi Emulation: **Not A vi Command**

Button for button bar: 

Description:
The Close All Operator closes all files currently open in the Bedit editor.

Operator: CloseFile

Config Name: CLOSEFILE

vi Emulation: :q or :q!

Button for button bar: 

Description:
The Close File Operator closes the current open file.

Operator: Command Line

Config Name: CMDLINE

vi Emulation: :

Button for button bar: 

Description:
The Command Line Operator emulates most of the vi command line features (such as :w). The Command Line Operator is only available in Professional Bedit. The Command Line Operator can be assigned to key combinations just like any other Bedit editor operator. However, since the command line commands can be abbreviated, they are not assignable within the Command Line Operator. Much of the functionality available through this operator is available through other Bedit Operators as well. However, some of the global editing functions available through the Command Line Operator are powerful and useful in their own right. In fact, since some of the Command Line functions are so powerful, the Bedit editor allows them to be interupted. While such a function is executing, a dialog box is shown, having a "Cancel" button. If the left mouse button is clicked anywhere on the screen (during execution), the operation will be interupted (canceled).

The functions available through the Command Line Operator are listed in the following table and then described in more detail later.

Table of Command Line Functions

Name/Abbreviation	Meaning
append/a	Append text after specified lines
args/ar	Display the Bedit editor run time arguments
change/c	Change specified lines
chdir/chd	Change current directory
copy/co or t	Copy specified lines after a specified line
delete/d	Delete specified lines
edit/e	Edit a named file or a new file
insert/i	Insert text before specified lines
join/j	Join specified lines
map	Record a new macro
mark/ma or k	Mark specified line
move/m	Move specified lines after the specified line
put/pu	Put (paste) previously copied text after specified line
quit/q	Close the file
read/r	Read lines from another file after specified line
redo	Redo changes
set/se	Set some Bedit parameters
substitute/s	Substitute for matching text within lines
undo	Undo changes
unmap	Unassign macros
version/ve	Display Bedit editor version
write/w	Write (save) the file
xit/x	Save the file as needed and close it
yank/y	Yank lines(i.e. copy to clipboard or buffer)
z	Reposition the current line
<	Shift lines left
>	Shift lines right

Purhaps the most powerful aspect of the Command Line Operator is the line addressing scheme. This is also somewhat complex and can be confusing. This is not intended to be a treatise on the subject but we will try to give

a useful description. Commands can have zero, one, or two line specifiers. When no line specifiers are given, the current line is assumed. A line specifier can be any of the following:

Line Specifiers

Specifier	Meaning
.	The current line
$	The last line of the file
'x or `x	The line previously marked as x (any of a-z)
%	All lines (same as 1,$)
/string/	Lines that match the search string

In addition, an offset can be specified using + or - optionally followed by a number. If no line specifier is present, but the offset is given, the offset is taken from the current line.

Operations can be specified to be global to the file by preceding the first search string with g or v. The g means include all matching lines in the file while v is the inverse, meaning include all line in the file that do not match.

Some examples of line addressing follow. 1,5 means lines 1 through 5, inclusive. +,- means the line before the current one, the current line, and the following line. $ means the last line of the file. /this one/ is any line containing the text "this one". g/start-of-block/end-of-block/ is the sequence of lines that starts with a line containing the string start-of-block and ends with the a line containing the string end-of-block. v/^1/ is the lines in the file that do not begin with the number 1.

Note: the search strings used for the Command Line Operator are syntactically the same as those used by the Find String Operator. This limits the vi command line emulation somewhat, but maintains consistency some users will appreciate.

Commands vary as to how many line specifiers they need or will pay attention to. This will be shown for each command below. Also note that spaces are usually ignored within a command line.

append:

Syntax: line1 append
Description: line1 is a line specifier. A new line is opened following line1.
Example: /append here/a
Appends text after the next line containing "append here".

args:

Syntax: args
Description: Display the Bedit editor run-time arguments.

change:

Syntax: line1, line2 change
Description: line1 and line2 are line specifiers, optionally preceded by g or v. The specified lines are deleted, replaced by new text.
Example: 5,12c
Changes lines 5 through 12.

chdir:

Syntax: chdir dirname
Description: dirname is the name of a directory. Change the current directory to the named directory. This directory is the starting point for all file operations, including (for example) the open file dialog.
Example: chdir c:\
Changes the current directory to c:\.

copy or t:

Syntax: line1, line2 copy line3
Syntax: line1, line2 t line3
Description: line1, line2, and line3 are line specifiers. line1 can optionally be preceded by g or v. The specified lines, line1 through line2 are copied after line3. Long copy operations are interuptable.

Note: this copy function does not act the same as the corresponding vi function (:copy). Unlike vi, the Bedit editor always preserves the original order of the lines during copies and moves. We think this is more useful than reversing the order when copying. If you disagree, we would like to hear from you.
Example: 5,12copy30
Copies lines 5 through 12 after line 30.

delete:

Syntax: line1, line2 delete
Description: line1 and line2 are line specifiers, optionally preceded by g or v. The specified lines are deleted. Long delete operations are interuptable.
Example: -,/endofdelete/d
The current line through the next line containing endofdelete are deleted.

edit:

Syntax: edit[!] filename
Description: The filename file is opened for editing. If filename is the single character %, the current file is specified. This will cause the current file to be re-read. If the file has been changed, the changes can be discarded by using the ! after the command.
Example: e! %
Re-read the current file, discarding any changes that have been made to it.Example: e myfile
Open the file "myfile" for editing in its own window.

insert:

Syntax: line1 insert
Description: line1 is a line specifier. A new line is opened before line1, where new text can be entered.
Example: 5i
Insert text before line 5.

join:

Syntax: line1, line2 join
Description: line1 and line2 are line specifiers, optionally preceded by g or v. The specified lines are joined. Long join operations are interuptable.
Example: 5,12j
Lines 5 through 12 are joined to form a single line.

mark or k:

Syntax: line1 mark letter
Description: line1 is a line specifier. letter is any of the letters a through z. Line1 is marked with the given letter.
Example: k d
Mark the current line with the letter d.

map:

Syntax: map
Description: Begin recording a new macro. See the Start Record Operator for details.

move:

Syntax: line1, line2 move line3
Description: line1, line2, and line3 are line specifiers. Line1 can optionally be preceded by g or v. The lines line1 through line2 are moved after line3. Long delete operations are interuptable.

NOTE: This move function does not act like the vi :move function. Unlike vi, the Bedit Command Line move function preserves the original order of lines that are moved or copied. We believe that this is more useful than reordering lines when moved. If you disagree, we would like to hear from you.
Example: 5,12m41
Moves lines 5 through 12 after line 41.

put:

Syntax: line1 put [letter]
Description: line1 is a line specifier. letter is a letter from a to z. If the letter parameter is missing, the contents of the Windows clipboard is pasted after line1. Otherwise the contents of the buffer specified by the letter is pasted after line1.
Example: 5,12pu b
Pastes the text from buffer "b" after line 12.

quit:

Syntax: quit[!]
Description: Close the current file. If changes have been made to the file, the changes can be discarded by following the command with the character "!".
Example: q!
Close the file without asking about any changes that have been made since the last file save.

read:

Syntax: line1 read [filename]
Description: Read text from filename and insert it after line1. If no filename is specified, this brings up a dialog to choose which file to "open" for insertion. Also see the File Insert Operator.
Example: 5,12r
Changes lines 5 through 12.

redo:

Syntax: redo
Description: The undo command reverts the text to a state before a change was made. The redo command changes the text back to what it was before the last undo if no changes have been made since. If changes have been made to the text after the last undo, redo has no effect. See the Undo and Redo operators for more information.
Example: 5,12r
Changes lines 5 through 12.

set:

Syntax: set parameter
Description: parameter can be any of "ai", "noai", "sw=number-of-columns", or "wc=wrap-column-number". (The quotes are not part of the command line.) This command sets Bedit editor parameters. The examples describe their meanings.
Example: set ai
Set auto indent to on.
Example: set noai
Set auto indent to off.
Example: set sw=4
Set the shift width to 4. See the Set Shift Width Operator for details.
Example: set wc=68
Set the wrap column to 68. This only has an effect while word wrap is turned on. See the Word Wrap Operator for details.

substitute:

Syntax: line1, line2 substitute/srchstring/replstring[g]
Description: line1 and line2 are line specifiers, optionally preceded by g or v. The specified lines are searched for

occurrences of the srchstring string. The first occurrence of srchstring is replaced by the replstring string in each specified line. If the letter g follows the replstring string, all occurrences of srchstring are replaced by replstring in each line. Note: sometimes it is desirable to use a different delimiter than "/". Any non-letter and non-number other than <, >, &, or ~ can be used instead of /.
Long substitute operations are interuptable.
Example: 5,12s/xyz/abc/g
Searches lines 5 through 12 for occurrences of the string "xyz" and replaces every xyz string with "abc".
Example: g;this one;s,x/z,a/c,g
Searches all lines containing the string "this one" for occurrences of the string "x/z" and replaces every x/z string with "a/c".

undo:

Syntax: undo
Description: Undo a text change. Unlike vi, the Bedit editor has multiple levels of undo and redo. If some changes have already been undone, this undo undoes another change. See the Undo and Redo operators for more information.

unmap:

Syntax: unmap
Description: Brings up a dialog box allowing macros to be assigned or unassigned to key combinations. See the Macro Assign Operator for details.

version:

Syntax: version
Description: Displays current Bedit editor version information

write:

Syntax: write filename
Description: filename is a file name that will be used to save the current file text. If no filename is specified, the current file name is used. If followed by an exlamation mark (!) the text is saved in the named file even if it already exists and is not the current file.

wq:

Syntax: wq filename
Description: filename is a file name that will be used to save the current file text. If no filename is specified, the current file name is used. If followed by an exlamation mark (!) the text is saved in the named file even if it already exists and is not the current file. After successfully saving, the file (edit window) is closed.

xit:

Syntax: xit
Description: Saves the file if changes have been made and then closes the file (edit window).

yank:

Syntax: line1, line2 yank letter
Description: line1 and line2 are line specifiers, optionally preceded by g or v. letter is any letter from a to z, specifying a buffer. If the letter parameter is present, the lines from line1 through line2 are copied to the specified buffer. Otherwise they are copied to the Windows clipboard.
Example: 5,9y
Copies lines 5 through 9 to the Windows clipboard.

Example: /abc/,/xyx/y a
Searches for the next line containing the text "abc" and then the line containing the text "xyz" and copies these lines and all lines between them into the "a" buffer.

z:

Syntax: z[+-.]
Description: Repositions the current line in the edit window. If the z is followed by a "+", the current line is the first displayed line. If it is followed by a "-", the current line is the last displayed line. If the z is followed by a ".", the current line is displayed in the center of the window.

<:

Syntax: line1, line2 <
Description: Shifts all lines from line1 through line2 to the left by the current shift width. Note: a tab is always considered to be the size of the shift width. Thus, if a line begins with tabs, the last tab before visible text is deleted. If a line begins with spaces, the last shift-width number of spaces are deleted. Long shift operations are interuptable.
Example: 3,/xyz$/<
This shifts line 3 through the next line containing the string xyz, to the left.

>:

Syntax: line1, line2 >
Description: Shifts all lines from line1 through line2 to the left by the current shift width. Note: a tab is always considered to be the size of the shift width. Thus, a single tab is inserted at the beginning of each line. Long shift operations are interuptable.
Example: $>
This shifts the last line of the file to the right.

Some of the features of the vi command line are already available in easier forms or not as meaningful in the Windows environment. For example, the "file" and "=" features display information that is always visible on the status line of Bedit. The "next" and "rewind" features have little value when all desired files are already in edit windows. The "stop" feature is not needed in a windowing environment.

Some of the vi command line features have not yet been implemented in the Bedit editor, but would be considered for Version 2.0 if users desire them. These include: repeat substitute, number lines, list, vi style macros, abbreviations, "so", print (p), "!", and changes in the search string syntax.

Operator: Command Mode

Config Name: CMDMODE

vi Emulation: ESCAPE

Button for button bar: 

Description:
The Bedit editor has two basic modes. They are Command Mode and Text Entry Mode. While in Text Entry Mode, text entered from the keyboard is entered in the file being edited (actually in buffers ready to save to the file). This is much like typing on a typewriter. Keys that don't make sense as ASCII characters (such as function or control keys) are not entered in the file, and may be assigned to Bedit operators. While in Command Mode, typed characters are interpreted as potential operators and are not entered in the file. Thus, Command Mode provides the ability to have more single key stroke operators (i.e. commands).

The Command Mode Operator places the Bedit editor in Command Mode and initializes the editor state. The primary use for this operator is to change from Text Entry Mode to Command Mode. However, if the "Type:" entry in the status line (at the bottom of the editor window) shows something other than "Anything", the Command Mode Operator will usually reset this to "Anything" so that further keyboard input will be interpreted as operators.

Operator: Copy

Config Name: COPY

vi Emulation: y

Button for button bar: 

Description:
The Copy Operator copies selected text to the clipboard (the default) or to a Bedit buffer, if used in conjunction with the Buffer Operator. If text was selected before executing the Copy Operator, the selected text is copied. If the Copy Operator is executed twice in a row (without selected text), the current line or the specified number of lines are copied (the count may be entered before the Copy Operator). If the Copy Operator is executed once without previously selected text, the text from the current caret position to the next position is selected. The "next" caret position can be selected with the mouse or by using a motion operator such as the Back Operator.

See Also: Cut, Copy Line, Cut Line, Paste, String Replace, Buffer

Operator: CopyLine

Config Name: COPYLINE

vi Emulation: Y

Button for button bar: 

Description:
The Copy Line Operator copies the current line to the clipboard (the default) or to a Bedit buffer, if used in conjunction with the Buffer Operator. A preceding count can specify more than one line to copy.

See Also: Cut, Copy, Cut Line, Paste, String Replace, Buffer Operator

Operator: Char Replace

Config Name: CREPLACE

vi Emulation: r

Button for button bar: 

Description:
The Char Replace Operator replaces one or more characters with a specified character. If the Char Replace Operator is preceded with a count, that many characters are replaced with the character that follows the operator. The default is that the single character after the caret is replaced.

Operator: Cut

Config Name: CUT

vi Emulation: d

Button for button bar: 

Description:
The Cut Operator copies text to the clipboard or to a named buffer and then deletes the text. The text to be cut may be specified in the same ways as for the Copy Operator. That is, text may be preselected (e.g. with the mouse), the Cut Operator may be executed twice in a row (with or without a preceding count), or the Cut Operator may be followed by a motion operator or mouse selection.

See Also: Copy, Copy Line, Cut Line, Paste, String Replace, Buffer

Operator: Cut Line

Config Name: CUTLINE

vi Emulation: D

Button for button bar: 

Description:
The Cut Line Operator cuts text from the current caret position to the end of the line. If a count was entered before the Cut Line Operator was executed, the count specifies the number of lines to cut. The text is either cut to the Clipboard (the default) or to a Bedit buffer if the Buffer Operator is used to specify the buffer.

See Also: Copy, Copy Line, Cut, Paste, String Replace, Buffer

Operator: Down

Config Name: DOWN

vi Emulation: j

Button for button bar: 

Description:
The Down Operator moves the caret down one line. If a count is specified, the caret is moved down the specified number of lines. In most situations, while in Command Mode the column number is maintained for up and down motion.

See Also: Up, Left, Right

Operator: End

Config Name: END

vi Emulation: e

Button for button bar: 

Description:
The End Operator places the caret at the end of the closest word to the right of the caret. If a count is given before the End Operator, it specifies a number of words, the last of which the caret is placed after.

See Also: End Big, Back, Back Big, Word, Word Big

Operator: End Big

Config Name: ENDBIG

vi Emulation: E

Button for button bar: 

Description:
The End Big Operator places the caret at the end of the closest white space separated word to the right of the caret. If a count is given before the End Big Operator, it specifies a number of big words, the last of which the caret is placed after.

See Also: End, Back, Back Big, Word, Word Big

Operator: End Line

Config Name: ENDLINE

vi Emulation: $

Button for button bar: 

Description:
The End Line Operator places the caret at the end of the current line.

Operator: Exit

Config Name: EXIT

vi Emulation: :x or :xit

Button for button bar: 

Description:
The Exit Operator causes the Bedit editor to close all open files and to exit. If some files have not been saved after they were modified, you are given the option of saving (or not) each file or canceling the exit.

The Command Line Operator "exit" command saves the file if it has changed and then exits.

See Also: Command Line

Operator: Export

Config Name: EXPORT

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Export Operator saves the current file to disk, changing the end-of-line sequences to UNIX style. The Export, ExportAs and Import Operators are particularly useful when editing files on a network with UNIX machines. This operator is only available in Professional Bedit.

The Bedit editor always assumes DOS end-of-line sequences for internal purposes. Attempting to edit a UNIX file without Importing it can have unpredictable results. A UNIX file may be opened in the same ways as a DOS file is opened. However, the UNIX file has UNIX style end-of-line sequences and the user will be asked if the file should be Imported. If the user answers yes to this question, the UNIX file will be Imported without explicitly executing the Import Operator. The Import Operator explicitly imports a UNIX file, translating the UNIX style end-of-line sequences to DOS end-of-line sequences for internal use. When the file is saved, the end-of-line sequences are translated back to the UNIX style.

See Also: ExportAs, Import, Save, Save All, Open, SaveAs

Operator: ExportAs

Config Name: EXPORTAS

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The ExportAs Operator writes the current file to disk changing the end of line sequences to UNIX style and allowing a different file name to be specified. The Export, ExportAs and Import Operators are particularly useful when editing files on a network with UNIX machines. See the Export Operator for more information about UNIX file editing. This operator is only available in Professional Bedit.

See Also: Import, Export, Save, Save All, Open, SaveAs

Operator: File Insert

Config Name: FILEINSERT

vi Emulation: :r

Button for button bar: 

Description:
The File Insert Operator copies the text from another file into the current file (i.e. into the editor buffers) at the caret position. Any selected text is replaced.

The Command Line Operator (read command) can also be used to insert text from other files.

See Also: Command Line

Operator: Find Char

Config Name: FINDCH

vi Emulation: f

Button for button bar: 

Description:
The Find Char Operator moves the caret to the right on the current line to the first occurrence of the specified character. The character entered after the Find Char Operator specifies the character to find.

See Also: Find Char Back, Repeat Find Char Left, Repeat Find Char Right

Operator: Find Char Back

Config Name: FINDCHBACK

vi Emulation: F

Button for button bar: 

Description:
The Find Char Back Operator moves the caret to the left on the current line to the first occurrence of the specified character. The character entered after the Find Char Back Operator specifies the character to find.

See Also: Find Char, Repeat Find Char Left, Repeat Find Char Right

Operator: Find String

Config Name: FINDST

vi Emulation: / or ?

Button for button bar: 

Description:
The Find String Operator brings up a dialog box, allowing the entry of a string to find. The string length is limited to 127 characters. If longer find strings are needed (within a single line) use the Set Find String Operator. The dialog box also provides several options for the search. These options include:

1) Match Whole Words Only (to disallow partial word matches)
2) Match Case (to match upper/lower case letters)
3) Search up or down

The search string is interpreted as a regular expression. That is, some characters have special meaning as follows:

1) ^ as the first character of the string means that the string only matches beginning in the first column.
2) **$** as the last character of the string means that the string only matches if ending at the end of a line.
3) **?** matches any character.
4) \ escapes the next character in the string. That is, the next character will match that character. E.g. **\?** will match a question mark.
5) ***** matches any string of zero or more characters.
6) **[string]** matches any character in **string**. Beginning the **string** with ^ matches any character not in the **string**. A **-** in the **string** specifies a range of values. For example, **c-f** matches any letter from c through f, inclusively.

Some examples of a search string are:
Jane

Matches the name Jane.
Ja?e

Matches the strings Jane, or Jake, or Ja_e, etc.
Ja[nk]e

Matches the strings Jane or Jake.
Ja[a-z]e

Matches the strings Jaae, Jabe, Jace, through Jaze.
Ja[^A-Z]e

Matches strings beginning with **Ja** not having a capital letter in the third position and ending with **e**.
the*place
Matches strings such as **the right place** or **the Rogers place**.

See Also: Find Next, Find Previous, Set Find String, Search and Replace

Operator: Find Next

Config Name: FNEXT

vi Emulation: n

Button for button bar: 

Description:
The Find Next Operator searches forward (down) in the current file for the current "find string". The find string is set by the Find String Operator, by the Set Find String Operator, or by the Search and Replace Operator.

See Also: Find Previous, Find String, Set Find String, Search and Replace

Operator: Find Previous

Config Name: FPREV

vi Emulation: N

Button for button bar: 

Description:
The Find Previous Operator searches backwards (up) in the current file for the current "find string". The find string is set by the Find String Operator, by the Set Find String Operator, or by the Search and Replace Operator.

See Also: Find Next, Find String, Set Find String, Search and Replace

Operator: Goto

Config Name: GOTO

vi Emulation: G

Button for button bar: 

Description:
The Goto Operator goes to (moves the caret to) the first nonwhite space character in the specified line. The default is the last line of the file. If a count is entered before the Goto Operator, the count specifies the desired line number where the first line is line 1.

Operator: Goto Beginning of Line Operator

Config Name: GOTOLINESTART

vi Emulation: ^

Button for button bar: 

Description:
The Goto Beginning of Line Operator goes to (moves the caret to) the first non-white-space character in the current line. In most cases, this is themost useful "beginning of the line".

Operator: Goto Marked Char

Config Name: GOTOCHAR

vi Emulation: `

Button for button bar: 

Description:
The Goto Marked Char Operator goes to a previously marked position in the text. The marked location is specified by a letter entered following the Goto Marked Char Operator. The letter is the name used with the Mark Operator.

See Also: Mark, Goto Marked Line

Operator: Goto Column

Config Name: GOTOCOL

vi Emulation: |

Button for button bar: 

Description:
The Goto Column Operator goes to a column in the current line. The column is specified by a count preceding the Goto Column Operator. The default is the last column. The first column is column 1.

Operator: Goto Marked Line

Config Name: GOTOLINE

vi Emulation: '

Button for button bar: 

Description:
The Goto Marked Line Operator goes to the first nonwhite space character in a previously marked line of the text. The marked location is specified by a letter entered following the Goto Marked Line Operator. The letter is the name used with the Mark Operator.

See Also: Mark, Goto Marked Char

Operator: Help

Config Name: HELP

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Help Operator brings up context sensitive help. One way to learn about the Bedit operators is to execute one and then execute the Help Operator. Help will be displayed on the previously executed Operator. This may be quite useful if you do not know the meaning of a button on the button bar.

Operator: **Icon Arrange**

Config Name: **ICONARRANGE**

vi Emulation: **Not A vi Command**

Button for button bar: 

Description:
The Icon Arrange Operator arranges any icons in the Bedit editor.

Operator: Import

Config Name: IMPORT

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Import Operator reads a UNIX file from disk, changing the end-of-line sequences to DOS style for internal editor use. The Export, ExportAs and Import Operators are particularly useful when editing files on a network with UNIX machines. However, a file may be Opened using the Open Operator or a file may be "dragged and dropped" onto the Bedit editor to Open it. For UNIX files, Bedit will recognize the UNIX end-of-line sequences and ask if the file should be Imported. See the Export Operator for more information about UNIX file editing in Bedit. See the Open Operator for more information about opening files in the Bedit editor. An Imported file may be saved to disk by any of several different means. If Autosave is turned on in the configuration file, the file will be saved as a UNIX file automatically as often as specified in the configuration file. Using Autosave is exactly like using the Save Operator. The Save Operator saves any Imported or Exported file as a UNIX file. The Import Operator is only available in Professional Bedit.

See Also: Export, ExportAs, Save, Save All, Open, SaveAs

Operator: Insert

Config Name: INSERT

vi Emulation: i

Button for button bar: 

Description:
The Insert Operator puts the Bedit editor in Text Entry Mode without moving the caret. Any selected text is unselected (to insert text at the beginning of the selection range). See the Command Mode Operator for a description of Text Entry and Command Modes within the Bedit editor. If it is preceded by a count, the Insert Operator duplicates the entered text "count" times at each point where text is entered until Command Mode is changed to.

See Also: Append, Command Mode

Operator: Join

Config Name: JOIN

vi Emulation: J

Button for button bar: 

Description:
The Join Operator joins the current line and the next line by removing the end of line sequence from the current line. This forms one line from the two lines.

Operator: Left

Config Name: LEFT

vi Emulation: h

Button for button bar: 

Description:
The Left Operator moves the caret one character position to the left. If a count is entered before the Left Operator, the caret is moved the specified number of positions to the left.

See Also: Up, Down, Right

Operator: Macro Assign

Config Name: MACASSIGN

vi Emulation: Not A vi COmmand

Button for button bar: 

Description:
The Macro Assign Operator brings up a dialog box to allow assignment of a previously defined macro to a key combination. A macro may be assigned to a key combination, such as the letter x or the combination: Ctrl+shift+b. After the key assignment has been made, each time the key combination is entered the associated macro will be replayed. A function key or control key combination can be used while in either Command or Text Entry Mode. Most other key combinations cannot be used to play a macro while in Text Entry Mode.

When a macro is assigned to a key combination, the Iterate box can be checked. If the Iterate box is checked, each time the key combination is entered, the macro will be played one or more times within a selected line range up to a selected maximum number of times. The First and Last Lines, and the Maximum Iterations are entered in a dialog box that is brought up when the key combination is entered. "Auto next line" can also be specified for the iteration.

If a macro is assigned to more than one key combination, each key combination can be observed (and potentially modified or unassigned) by selecting the macro multiple times from the list of macros.

For more information about Bedit macros see Macros.

The Command Line Operator command "unmap" can also be used to enter this dialog.

See Also: Macro Maintenance, Start Record, Stop Record, Command Line Operator

Operator: Macro Maintenance

Config Name: MACMAINT

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Macro Maintenance Operator brings up a dialog box to allow editing of previously defined macros. To edit a macro, select the macro from the list, then select key combinations you wish to change or select before, between, or after displayed key combinations to add to the macro. All selections must be made with the mouse, since keyboard entry is assumed to modify the macro being edited. To add an Enter type \n. To add a Tab type \t.

For more information about Bedit macros see Macros.

See Also: Macro Assign, Start Record, Stop Record

Operator: Mark

Config Name: MARK

vi Emulation: m

Button for button bar: 

Description:
The Mark Operator marks a position in the file so that the line (Goto Marked Line) or specific character position (Goto Marked Line) may be returned to, later in the same edit session. A letter must be entered following the Mark Operator to name the mark.

See Also: Goto Marked Char, Goto Marked Line

Operator: Match

Config Name: MATCH

vi Emulation: %

Button for button bar: 

Description:
The Match Operator moves the caret to the matching bracket [], brace {}, or parenthesis () if the caret is immediately before one of these characters.

Operator: New

Config Name: NEW

vi Emulation: :e

Button for button bar: 

Description:
The New Operator brings up a new empty edit window. This can be used to create a new file.

Operator: New Next Line

Config Name: NEWNEXTL

vi Emulation: O

Button for button bar: 

Description:
The New Next Line Operator creates an empty line after the current line.

See Also: New Previous Line

Operator: New Previous Line

Config Name: NEWPREVL

vi Emulation: O

Button for button bar: 

Description:
The New Previous Line Operator creates an empty line before the current line.

See Also: New Next Line

Operator: Next Line

Config Name: NEXTLINE

vi Emulation: + or ENTER

Button for button bar: 

Description:
The Next Line Operator moves the caret to the first nonwhite space character in the line following the current line.

See Also: Previous Line

Operator: Open File

Config Name: OPENFILE

vi Emulation: :e filename

Button for button bar: 

Description:
The Bedit editor can open files for editing in several different ways. Although the Open File Operator is a simple method for opening a file, more experienced users will prefer the MS Windows "drag and drop" method. Many MS Windows file managers allow a file to be selected from a list of files, and then "dragged" to a different part of the screen and "dropped" on an accepting application, such as the Bedit editor. See your file manager for details on the drag and drop method you can use. NOTE: once the Bedit editor is running, files must be dropped onto the running editor and NOT on the Bedit ICON. Dropping a file on the Bedit ICON (normally to execute or run the editor) will bring up the Bedit editor with the dropped file(s) if the editor is not yet running. If the editor is already running, dropping files on the ICON will bring the editor window "to the top", but will not add the files to those being edited. When files are dropped on the running editor, Bedit opens those files in addition to those already open. Any number of files can be dropped on the running Bedit editor at a time.

The Open File Operator brings up a dialog box to choose one or more existing files to open for editing. The file names may be typed in or selected from the lists of files. If a UNIX file is chosen(i.e. has UNIX end-of-line sequences), you will be asked if you wish the file to be Import'ed. Importing is recommended since editing without proper end-of-line sequences have unpredictable results. When an Import'ed file is Save'd, it will be Export'ed. This converts the end-of-line sequences to the UNIX style.

While using the Open dialog box multiple files can be chosen from a directory in two ways. A sequence of files can be selected by holding down the Shift key while selecting the second end of the sequence of files. A file can be added to or removed from the list of chosen files by holding down the Control key while selecting the file. In this way, an arbitrary set of files may be chosen to open from a single directory. However, if files from another directory are needed, these must be chosen in a separate step.

The Command Line Operator can also be used to open files using the "edit" command.

See Also: Save, Save All, SaveAs, Import, Export, ExportAs, Command Line

Operator: Overstrike

Config Name: OVERSTRIKE

vi Emulation: **Similar to R**

Button for button bar: 

Description:
The Overstrike Operator toggles between overstrike and insert modes. This only has an effect while in Text Entry Mode. While in insert mode, text is inserted after the caret. While in overstrike mode, the typed characters replace (overstrike) the characters after the caret.

Operator: Page Bottom

Config Name: PAGEBOT

vi Emulation: L

Button for button bar: 

Description:
The Page Bottom Operator moves the caret to the last line of the displayed text.

See Also: Page Home, Page Middle

Operator: Page Down

Config Name: PAGEDN

vi Emulation: CNTL+f

Button for button bar: 

Description:
The Page Down Operator displays the next page of text.

See Also: Page Up

Operator: Page Home

Config Name: PAGEHOME

vi Emulation: H

Button for button bar: 

Description:
The Page Home Operator moves the caret to the first line of the displayed text.

See Also: Page Middle, Page Bottom

Operator: Page Middle

Config Name: PAGEMID

vi Emulation: M

Button for button bar: 

Description:
The Page Middle Operator moves the caret to the middle line of the displayed text.

See Also: Page Home, Page Bottom

Operator: Page Up

Config Name: PAGEUP

vi Emulation: CNTL+b

Button for button bar: 

Description:
The Page Up Operator displays the previous page of text.

See Also: Page Down

Operator: Paste

Config Name: PASTE

vi Emulation: P

Button for button bar: 

Description:
The Paste Operator copies text from the Clipboard or from a Bedit buffer to replace the selected text. If no text is selected, the copied text is inserted at the caret. The "See Also" topics describe operators that copy text into the Clipboard or Bedit buffers. The Buffer Operator can be used to specify a Bedit buffer (preceding the Paste Operator).

See Also: Copy, Copy Line, Cut, Cut Line, String Replace, Buffer

Operator: Previous Line

Config Name: PREVLINE

vi Emulation: -

Button for button bar: 

Description:
The Previous Line Operator moves the caret to the first nonwhite space character in the line before the current line.

See Also: Next Line

Operator: Print

Config Name: PRINT

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Print Operator prints the current file if no text is selected. If text is selected, the selected lines can be printed. The option is also available to select the printer or to modify the printer setup.

Operator: Quote

Config Name: QUOTE

vi Emulation: CNTL+V

Button for button bar: 

Description:
The Quote Operator causes the next character to be put in the text verbatim, if possible. This is most often useful when placing control characters in the file.

Operator: Redisplay

Config Name: REDISPLAY

vi Emulation: CNTL+l

Button for button bar: 

Description:
The Redisplay Operator refreshes the display from the memory copy of the text. Although this is seldom needed, the Redisplay Operator can be used to verify the state of the text.

Operator: Redo

Config Name: REDO

vi Emulation: Not A vi Command (default is CTRL+BACKSPACE)

Button for button bar: 

Description:
The Redo Operator will redo changes that were previously undone using the Undo Operator (unless other edit changes have been made). The Undo Operator is the "Uh Oh!" Operator while the Redo Operator is the "Oh yah, that OK after all" Operator. More specifically, after an edit change has been made, the Undo Operator can be used to change the text back the way it was before the edit change was made. If it is then decided that the original edit change was needed, the Redo Operator makes the same edit change again. Professional Bedit has a configurable (potentially unlimited) number of levels of Undo and Redo while other versions of Bedit have 2 levels of Undo and Redo.

Example:
If the text "wokkinh" was changed to "working", the Undo Operator changes this back to "wokkinh". The Redo Operator changes it again to "working".

See Also: Undo

Operator: Repeat Change

Config Name: REPEATCHANGE

vi Emulation: .

Button for button bar: None

Description:
The Repeat Change Operator repeats the last change made by an operator or repeats the last macro. For example, if the last change was to insert "text" at the current caret position, then the Repeat Change Operator will insert "text" at a new caret position. When a macro is executed, the Repeat Change Operator will execute that macro again. Thus, this Operator cannot be used in a macro. A change is delimited by many things. A motion or mouse selection delimits a change. If auto indent is turned on, the Enter key delimits a change. Furthermore, only the first change after executing an operator is repeatable. In other words, if you execute the Insert Operator and enter a line of text, that line is repeatable. But if you then select another location while still in text entry mode and enter another line, this second line is not repeatable. NOTE: The Repeat Change Operator cannot be assigned to a button (on the button bar). See Configuration File more information about button bar assignments.

Operator: **Repeat Find Char Left**

Config Name: **REPFINDCHARLEFT**

vi Emulation: **,**

Button for button bar: 

Description:
The Repeat Find Char Left Operator attempts to find a character in the current line to the left of the caret. The character to find is the same as the last find char operation.

See Also: **Find Char Back**, **Find Char**, **Repeat Find Char Right**

Operator: Repeat Find Char Right

Config Name: REPFINDCHARRIGHT

vi Emulation: ;

Button for button bar: 

Description:
The Repeat Find Char Right Operator attempts to find a character in the current line to the right of the caret. The character to find is the same as the last find char operation.

See Also: Find Char Back, Find Char, Repeat Find Char Left

Operator: Search and Replace

Config Name: REPLACE

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Search and Replace Operator brings up a dialog box to specify a string to search for and a string with which to replace occurrences of the search string. Both strings are limited to 127 characters. The search string is a regular expression and is interpreted the same as for the Find String Operator.

The Search and Replace dialog box has buttons to select the type of action to take. The first button (Find Next) searches for the next occurrence of the search string without changing the file. The second (Replace) replaces the current selection and searches for the next occurrence of the search string. The third button (Replace All) searches from the current caret position to the end of the file, replacing each occurrence of the search string. If Replace All is selected, once the operation has begun, it can be canceled by clicking the left mouse button anywhere on the screen. (Warm and fuzzy.)

See Also: Find String

Operator: **Replace Line**

Config Name: **REPLACELINE**

vi Emulation: **C**

Button for button bar: 

Description:
The Replace Line Operator replaces the text from the caret position to the end of the line. If a count is entered before the Replace Line Operator, the count specifies how many lines are replaced after, and including the current one.

See Also: **String Replace**

Operator: Reposition

Config Name: REPOSITION

vi Emulation: Z

Button for button bar: 

Description:
The Reposition Operator positions the displayed text at the caret according to the next character typed as follows:

1) - (minus) places the current line at the bottom of the display
2) . (dot) places the current line in the middle of the display
3) any other character places the current line at the top of the display

Operator: ReRead

Config Name: REREAD

vi Emulation: **:e! % or :e %**

Button for button bar: 

Description:
The ReRead Operator reads the current file from disk using the same Bedit window. Any changes that have been made after last saving the file can be discarded. The ReRead Operator is most convenient when Bedit is used in conjunction with another program that modifies the same file. For example, while Bedit is editing a file, you can Save the file, execute a spell checker that modifies the same file (from another window), and then ReRead the file into Bedit to continue editing the file.

The Command Line Operator can also be used to reread the file by using the "edit" command.

See Also: **Command Line Operator**

Operator: Right

Config Name: RIGHT

vi Emulation: l

Button for button bar: 

Description:
The Right Operator moves the caret one position to the right. If a count is entered before the Right Operator, the caret is moved the specified number of positions to the right.

See Also: Up, Down, Left

Operator: Save

Config Name: SAVE

vi Emulation: :W

Button for button bar: 

Description:
The Save Operator unconditionally updates the current disk file with the memory copy of the text. This saves the edit changes that have been made to the text. If the file was Imported (or Exported to this file) a UNIX file is saved. If not, a DOS file is saved. Autosave works just like the Save Operator.

The Command Line Operator can also be used to save the file by using the "write" command.

See Also: Save All, Open, SaveAs, Import, Export, ExportAs, Command Line Operator

Operator: Save All

Config Name: SAVEALL

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Save All Operator unconditionally updates all disk files being edited with the memory copies of the text for the those files. This is like the Save Operator for every open file.

See Also: Save, Open, SaveAs, Import, Export, ExportAs

Operator: Save As

Config Name: SAVEAS

vi Emulation: :w filename

Button for button bar: 

Description:
The Save As Operator unconditionally saves the text for the current editor window to a file whose name is to be selected. A dialog box is brought up for the purpose of specifying the new file name. This file is assumed to be a DOS file and all subsequent saves to it will use the DOS end-of-line sequences (saving as a DOS file).

See Also: Save All, Open, Save, Import, Export, ExportAs

Operator: Scroll Down

Config Name: SCROLLDN

vi Emulation: CNTL+d

Button for button bar: 

Description:
The Scroll Down Operator scrolls the display down half a page.

See Also: Scroll Up

Operator: Scroll Up

Config Name: SCROLLUP

vi Emulation: CNTL+u

Button for button bar: 

Description:
The Scroll Up Operator scrolls the display up half a page.

See Also: Scroll Down

Operator: Select

Config Name: SELECT

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Select Operator selects text in the same ways as the Copy Operator, but does not copy the selected text anywhere.

See Also: Copy Operator

Operator: Select All

Config Name: SELECTALL

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Select All Operator selects all text for the current Bedit editor window.

See Also: Copy, Cut, String Replace

Operator: Set Find String

Config Name: SETFINDSTRING

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Set Find String Operator changes the string that will be used for the next Find Next or Find Previous, Operator. The string must not include an end-of-line sequence, since the string can only be found within a single line. The string length is not otherwise limited. If text is selected before executing the Set Find String Operator, that text is the new string used for later Find Next and Find Previous Operators. If no text is selected, a string must be entered through the keyboard. The string is shown on the status line (at the bottom of the Bedit window) as it is entered. The only editing available for this string as it is being entered is that backspace erases the previous character. The string is ended by pressing the Enter key. Since no dialog box is used, this method of changing the "find string" is simpler than the Find String Operator, but less powerful. This operator can be used in macros since it does not use a dialog box. Furthermore, this Operator allows more than one string to be searched for by a single macro.

The Find String Operator is recommended for normal user interactions since it has more options and flexibility. See Macros for suggestions on using the Set Find String Operator in macros.See Also: Find Next, Find Previous, Find String, Macros

Operator: Set Font

Config Name: SETFONT

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Set Font Operator brings up a dialog box to select and set the font for the current file in this edit session. When the new font has been set for this file, the user is given the option of using this font for the default font when other files are opened for editing. In the Professional Bedit editor the font selection is saved/restored for each file between edit sessions.

Operator: Set Shift Width

Config Name: SETSW

vi Emulation: :set sw=4

Button for button bar: 

Description:

The Set Shift Width Operator brings up a dialog box to set the current shift width (also known as the tab size). This can also be set using the Command Line Operator

See Also: Shift Right, Shift Left, Autoindent

Operator: Shift Left

Config Name: SHIFTL

vi Emulation: <

Button for button bar: 

Description:
The Shift Left Operator shifts the lines left by Shift Width columns. The rules for selecting the lines to shift are the same as for the Copy Operator or the Cut Operator. That is, if text is already selected, those lines are shifted left. Executing the Shift Left Operator twice in a row shifts the current line (or lines if a count was specified). Another alternative is to follow the Shift Operator with a motion to indicate the line range to shift.

See Also: Set Shift Width, Shift Right

Operator: Shift Right

Config Name: SHIFTR

vi Emulation: >

Button for button bar: 

Description:
The Shift Right Operator shifts the selected lines right by Shift Width columns. The rules for selecting the lines to shift are the same as for the Copy Operator or the Cut Operator. That is, if text is already selected, those lines are shifted left. Executing the Shift Right Operator twice in a row shifts the current line (or lines if a count was specified). Another alternative is to follow the Shift Operator with a motion to indicate the line range to shift.

See Also: Set Shift Width, Shift Left

Operator: String Replace

Config Name: SREPLACE

vi Emulation: c

Button for button bar: 

Description:
The String Replace Operator clears selected text and places the Bedit editor in Text Entry Mode. If text was already selected, that text is cleared. If the String Replace Operator is executed twice in succession, the current line is cleared. A count entered prior to the String Replace Operator causes that number of lines to be cleared. Alternatively, if the caret position is changed (e.g. by a motion Operator) after executing the String Replace Operator, the text is cleared between the two caret positions.

See Also: Replace Line

Operator: Start Record

Config Name: STARTRECORD

vi Emulation: Similar to :map

Button for button bar: 

Description:
The Start Record Operator begins recording a new macro. After this Operator is executed, all keyboard input is recorded in the new macro (except use of the "Alt" key) until the Stop Record Operator is executed. Care must be taken in using the mouse while recording a macro, since the action of the mouse will **NOT** be recorded. For more information about Bedit macros see Macros.

The Command Line Operator command "map" can also be used to begin recording a macro.

See Also: Macro Assign, Macro Maintenance, Stop Record, Command Line Operator

Operator: Stop Record

Config Name: STOPRECORD

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Stop Record Operator stops recording a new macro begun with the Start Record Operator, and then brings up a dialog box to allow naming and a potential key assignment for the new macro. The maximum number of macros for Professional Bedit is 127 while it is 2 for other versions of Bedit. A new macro must be given a unique name, but it need not be assigned to a key combination when the Stop Record Operator is executed. If a macro is assigned to a key combination, whenever that key combination is entered, the macro is played. A key combination can be used in the same ways whether it has a Bedit macro or a Bedit Operator assigned to it. That is, a Bedit macro can be played (by entering the key combination assigned to it) directly from the keyboard or by entering that key combination in another macro. In this way macros can be built on other macros. However, care must be taken not to build too many "levels" of macros. If a macro is built on another macro which is built on another built on another, etc, these levels can cause a "stack overflow", resulting in an editor crash. This would usually require many more levels than would prove useful, so you should not be too concerned about using multilevel macros. However, we do suggest that macros not be built in more than about 5 levels: not for reliability, but because multilevel macros are slower. The valid key combinations are listed under Operator Assignable Key Combinations in Configuration File.

When a macro is assigned to a key combination, the box can be checked. This indicates that whenever the macro is played a dialog box will be brought up to enter parameters for iteratively playing the macro. The First and Last Lines restrict where the macro can be active. The Max Times is the maximum number of times the macro will be allowed to play within the specified line range. For more information about Bedit macros see Macros.

See Also: Macro Assign, Macro Maintenance, Start Record

Operator: Substitute

Config Name: SUB

vi Emulation: S

Button for button bar: 

Description:
The Substitute Operator replaces one or more characters in the current line with new text. A count preceding the operator specifies the number of characters to replace, with default of one character replaced.

Operator: Tile

Config Name: TILE

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Tile Operator reformats the Bedit display to show all file windows as non-overlapping (tiled) windows.

Operator: Toggle Case

Config Name: TOGGLECASE

vi Emulation: ~

Button for button bar: 

Description:
The Toggle Case Operator toggles the case of the next character if it is a letter and the caret is moved to the right of the toggled character. If text is selected, all selected letters are toggled. That is, each lower case letter is changed to upper case and vice versa. If a count precedes the Toggle Case Operator, it has the same effect as selecting the specified number of characters.

Operator: Undo

Config Name: UNDO

vi Emulation: u

Button for button bar: 

Description:
The Undo Operator undoes changes made to the text. The maximum number of sequential undo operations (levels of undo and redo) is configurable (potentially unlimited) for Professional Bedit and is 2 for other versions of Bedit. See the Redo Operator for more information about the Bedit Undo and Redo Operators.

See Also: Redo

Operator: Unselect

Config Name: UNSELECT

vi Emulation: Not A vi Command

Button for button bar: 

Description:
The Unselect Operator unselects any currently selected text, leaving thecaret at the beginning of the previously selected text. See Mouse Usage, the Select Operator and the Copy Operator for more information about text selection within the Bedit editor.

See Also: Mouse Usage, Select Operator, Copy Operator

Operator: Up

Config Name: UP

vi Emulation: k

Button for button bar: 

Description:
The Up Operator moves the caret one line up. If a count is entered before the Up Operator, the caret is moved the specified number of lines up.

See Also: Right, Down, Left

Operator: Word

Config Name: WORD

vi Emulation: W

Button for button bar: 

Description:
The Word Operator moves the caret to the beginning of the next word, where words are considered to be composed of alphanumeric characters (including the underscore).

See Also: Back, Backbig, Wordbig

Operator: Word Big

Config Name: WORDBIG

vi Emulation: W

Button for button bar: 

Description:
The Word Big Operator moves the caret to the beginning of the next "big" word, where big words are considered to be separated by white space.

See Also: Back, Backbig, Word

Operator: Word Wrap

Config Name: WORDWRAP

vi Emulation: Similar to :set wm=

Button for button bar: 

Description:
The Word Wrap Operator toggles word wrap. When word wrap is checked in the menu (word wrap on) text entered in Text Entry Mode will be broken at white space before the word wrap column specified in the configuration file or setusing the Command Line Operator "set wc=" command. See Configuration File for more information about initialization parameters in the Bedit configuration file.

See Also: Command Line Operator

Windows Keys

The keyboard topics below describe the standard Windows keyboard usage. The default configuration files maintain consistency with these descriptions. Confusion may result if you change these assignments in the configuration file,so it is recommended that you keep the following assignments in bedit.cfg:

DELETE = CDELETERANY

BACKSPC = LEFT

Ctrl+INS = COPY

Shift+INS = PASTE

Shift+DELETE = CUT

Choose from the following list to review the standard keys used in Windows:

Windows Cursor Movement Keys

Key(s)	Function
DIRECTION key	Moves the cursor left, right, up, or down in a field.
End or Ctrl+Right Arrow	Moves to the end of a field.
Home or Ctrl+Left Arrow	Moves to the beginning of a field.
PAGE UP or PAGE DOWN	Moves up or down in a field, one screen at a time.

Windows Dialog Box Keys

Key(s)	Function
TAB	Moves from field to field (left to right and top to bottom).
Shift+TAB	Moves from field to field in reverse order.
ALT+letter	Moves to the option or group whose underlined letter matches the one you type.
DIRECTION key	Moves from option to option within a group of options.
ENTER	Executes a command button. Or, chooses the selected item in a list box and executes the command.
ESC	Closes a dialog box without completing the command. (Same as Cancel)
ALT+DOWN ARROW	Opens a drop-down list box.
ALT+UP or DOWN ARROW	Selects item in a drop-down list box.
SPACEBAR	Cancels a selection in a list box. Selects or clears a check box.
Ctrl+SLASH	Selects all the items in a list box.
Ctrl+BACKSLASH	Cancels all selections except the current selection.
Shift+ DIRECTION key	Extends selection in a text box.
Shift+ HOME	Extends selection to first character in a text box.
Shift+ END	Extends selection to last character in a text box

Windows Editing Keys

Key(s)	Function
Backspace	Deletes the character to the left of the cursor. Or, deletes selected text.
Delete	Deletes the character to the right of the cursor. Or, deletes selected text.

Windows Help Keys

Key(s)	Function
F1	Gets Help and displays the Help Index for the application. If the Help window is already open, pressing F1 displays the "Using Windows Help" topics. In some Windows applications, pressing F1 displays a Help topic on the selected command, dialog box option, or system message.

Windows Menu Keys

Key(s)	Function
Alt	Selects the first menu on the menu bar.
Letter key	Chooses the menu, or menu item, whose underlined letter matches the one you type.
Alt+letter key	Pulls down the menu whose underlined letter matches the one you type.
LEFT or RIGHT ARROW	Moves among menus.
UP or DOWN ARROW	Moves among menu items.
Enter	Chooses the selected menu item.

Windows System Keys

The following keys can be used from any window, regardless of the application you are using.

Key(s)	Function
Ctrl+Esc	Switches to the Task List.
Alt+Esc	Switches to the next application window or minimized icon, including full-screen programs.
Alt+TAB	Switches to the next application window, restoring applications that are running as icons.
Alt+PrtSc	Copies the entire screen to Clipboard.
Ctrl+F4	Closes the active Bedit window.
Ctrl+F6	Makes the next Bedit window active.
F1	Gets Help and displays the Help Index for the application. (See Help Keys)

Windows Text Selection Keys

Within the Bedit editor, text can be selected using the keyboard, the mouse or Bedit Operators. The following Windows standard keyboard methods are available in Bedit for those who prefer them.

Key(s)	Function
Shift+LEFT or RIGHT ARROW	Selects text one character at a time to the left or right.
Shift+DOWN or UP	Selects one line of text up or down.
Shift+END	Selects text to the end of the line.
Shift+HOME	Selects text to the beginning of the line.
Shift+PAGE DOWN	Selects text down one window.
Shift+PAGE UP	Selects text up one window.
Ctrl+Shift+LEFT or RIGHT ARROW	Selects text to the next or previous word.
Ctrl+Shift+END	Selects text to the end of the document.
Ctrl+Shift+HOME	Selects text to the beginning of the document.

In addition to the above Windows standards, the Bedit editor also supports the following text selection methods.

Ctrl+Shift+PAGE UP or PAGE DOWN	Selects text to the top (PAGE UP) or bottom (PAGE DOWN) of the page.

Windows Menu/Window Keys

Key(s)	Function
ALT+SPACEBAR	Opens the Control menu for an application window.
Alt+Hyphen	Opens the Control menu for a document window.
Alt+F4	Closes a window.
Alt+Esc	Switches to the next application window or minimized icon, including full-screen programs.
Alt+TAB	Switches to the next application window, restoring applications that are running as icons.
Alt+ENTER	Switches a non-Windows application between running in a window and running full screen.
DIRECTION key	Moves a window when you have chosen Move from the Control menu. Or, changes the size of a window when you have chosen Size from the Control menu.