Name
joe – Joe's Own Editor

Syntax
joe [global-options] [ [local-options] filename ]...
jstar [global-options] [ [local-options] filename ]...
jmacs [global-options] [ [local-options] filename ]...
rjoe [global-options] [ [local-options] filename ]...
jpico [global-options] [ [local-options] filename ]...
jupp [global-options] [ [local-options] filename ]...

Description
JOE is a powerful ASCII-text screen editor. It has a "mode-less" user interface which is similar to many user-friendly PC editors. Users of Micro-Pro's WordStar or Borland's "Turbo" languages will feel at home. JOE is a full featured UNIX screen-editor though, and has many features for editing programs and text.

JOE also emulates several other editors. JSTAR is a close imitation of WordStar with many "JOE" extensions. JPICO is a close imitation of the Pine mailing system’s PICO editor, but with many extensions and improvements. JMACS is a GNU-EMACS imitation. RJOE is a restricted version of JOE, which allows you to edit only the files specified on the command line.

Although JOE is actually six different editors, it still requires only one executable, but one with six different names. The name of the editor with an "rc" appended gives the name of JOE's initialisation file, which determines the personality of the editor.

JUPP is free software; you can distribute it and/or modify it under the terms of the GNU General Public License, Version 1, as published by the Free Software Foundation. (main.c contains more detailed exceptions.) I have no plans for turning JOE into a commercial or share-ware product. See the source code for exact authorship and licencing information. JOE is available over the Internet from http://joe-editor.sf.net/. JUPP is available at http://mirbsd.de/jupp.

Usage
To start the editor, type joe followed by zero or more names of files you want to edit. Each file name may be preceded by a local option setting (see the local options table which follows). Other global options, which apply to the editor as a whole, may also be placed on the command line (see the global options table which follows). If you are editing a new file, you can either give the name of the new file when you invoke the editor, or in the editor when you save the new file. A modified syntax for file names is provided to allow you to edit program output, standard input/output, or sections of files or devices. See the section Filenames below for details.

On cygwin32 systems, the special option -CYGhack is replaced by anything that comes past it (and separating whitespace) on the command line as one option (to work around a Cygwin bug as it cannot correctly be passed a UNC pathname with spaces as one argument from Explorer at all).

Once you are in the editor, you can type in text and use special control-character sequences to perform other editing tasks. To find out what the control-character sequences are, read the rest of this man page or type ^K H for help in the editor.

Now for some obscure computer-lore:
The ^ means that you hold down the Control key while pressing the following key (the same way the Shift key works for uppercase letters). A number of control-key sequences are duplicated on other keys, so that you don’t need to press the control key: ESC will work in place of ^], Del will work in place of ^?, Backspace will work in place of ^H, Tab will work in place of ^I, Return or Enter will work in place of ^M and Linefeed will work in place of ^J. Some keyboards may give you trouble with some control keys. ^_ and ^@ can usually be entered without pressing shift (i.e., try ^_, ^6 and ^2). Other keyboards may realign these to other keys. Try: ^., ^, and ^/. ^SPACE can usually be used in place of ^@. ^\ and ^] are interpreted by many communication programs, including telnet and kermit. Usually you just hit the key twice to get it to pass through the communication program.

Once you have typed ^KH, the first help window appears at the top of the screen. You can continue to enter and edit text while the help window is on. To page through other topics, hit ^[ and ^[. (that is, ESC , and ESC .) Use ^KH to dismiss the help window.

You can customise the keyboard layout, the help screens and a number of behavior defaults by copying JOE’s initialisation file (usually /etc/jupp/joerc) to .joerc in your home directory and then by modifying it. See the section joerc below. The filename is actually .nameerc where name is the argv[0] the editor is called with.

Custom syntax files are loaded from .jupp/syntax/name.jsf in your home directory and .jupp/charmaps/name holds custom charmaps (name here is the name of the syntax or charmap).

To have JOE used as your default editor for e-mail and News, you need to set the EDITOR and VISUAL environment variables in your shell initialisation file (.cshrc or .profile) to refer to JOE (the joe binary usually resides as /usr/local/bin/joe).

There are a number of other obscure invocation parameters which may have to be set, particularly if your terminal screen is not updating as you think it should. See the section Environment variables below.

Command Line Options

The following global options may be specified on the command line:

-asis Characters with codes above 127 will be sent to the terminal as-is, instead of as inverse of the corresponding character below 128. If this does not work, check your terminal server.

-backpath path If this option is given, backup files will be stored in the specified directory instead of in each file’s original directory.

-baud nnn Set the baud rate for the purposes of terminal screen optimisation. Joe inserts delays for baud rates below 19200, which bypasses tty buffering so that typeahead will interrupt the screen output. Scrolling commands will not be used for 38400 baud. This is useful for X-terms and other console ttys which really aren’t going over a serial line.

-beep Joe will beep on command errors and when the cursor goes past extremes.

-columns nnn Sets the number of screen columns.
-csmode
Continued search mode: a search immediately following a search will repeat the previous search instead of prompting for new string. This is useful for the the `^[S and `^[R commands and for when joe is trying to be emacs.

-dopadding
Joe usually assumes that there is some kind of flow control between it and the tty. If there isn’t, this option will make joe output extra `^[@s to the tty as specified by the termcap entry. The extra `^[@s allow the terminal to catch up after long terminal commands.

-exask
This option makes `^[KX verify the file name that it’s about to write.

-force
This option makes sure that the last line of the file has a line-feed which it’s saved.

-help
The editor will start with the help screen on if this option is given.

-keepup
Normally the column number and control-key prefix fields of the status lines are on a one second delay to reduce CPU consumption, but with this option they are updated after each keystroke.

-lightoff
The block highlighting will go away after any block command if this option is given.

-lines nnn
Sets the number of screen lines.

-marking
Text between `^[KB and the cursor is highlighted (use with -lightoff and a modified joerc file to have drop-anchor style block selection).

-mid
If this option is set and the cursor moves off the window, the window will be scrolled so that the cursor is in the center. This option is forced on slow terminals which don’t have scrolling commands.

-nobackups
This option prevents backup files.

-nonotice
This option prevent the copyright notice from being displayed when the editor starts.

-nosta
This option eliminates the top-most status line. It’s nice for when you only want to see your text on the screen or if you’re using a vt52.

-noxon
Attempt to turn off `^[S/^[Q processing. This is useful for when joe is trying to be WordStar or EMACS.
-orphan
   When this option is active, extra files on the command line will be placed in orphaned buffers instead of in extra windows. This is useful for when joe is trying to be emacs.

-pg nnn  This specifies the number of lines to keep after PgUp/PgDn (‘U/’V). If -1 is given, half the window is kept.

-skiptop nnn
   Don’t use the top nnn lines of the screen. Useful for when joe is used as a BBS editor.

Each of these options may be specified in the joerc file as well. In addition, the NOXON, BAUD, LINES, COLUMNS and DOPADDING options may be specified with environment variables. See the section Environment variables below.

The following options may be specified before each filename on the command line:

+nnn  The cursor starts on the specified line.

-crlf  Joe uses CR-LF as the end of line sequence instead of just LF. This is for editing MS-DOS or VMS files.

-hex  Sets the buffer to hex edit mode.

-wordwrap
   Joe wraps the previous word when you type past the right margin.

-autoindent
   When you hit Return on an indented line, the indentation is duplicated onto the new line.

-overwrite
   Typing overwrites existing characters instead of inserting before them.

-lmargin nnn
   Sets the left margin.

-rmargin nnn
   Sets the right margin.

-tab nnn
   Sets the tab width.

-indentc nnn
   Sets the indentation character for ’K, and ’K. (32 for SPACE, 9 for TAB).

-istep nnn
   Sets the indentation step for ’K, and ’K.
-linums Line numbers are displayed before each line.
-rdonly The file is read only.

-keymap name
Use an alternate section of the joerc file for the key sequence bindings. For example, joe, jstar, rjoe and jupp support -keymap cua to make \Z, \X, \C and \V do the same thing as in contemporary GUI editors.

These options can also be specified in the joerc file. They can be set depending on the file-name extension. Programs (.c, .h or .p extension) usually have autoindent enabled. Wordwrap is enabled on other files, but rc files have it disabled.

Editing Tasks
Basic Editing

When you type characters into the editor, they are normally inserted into the file being edited (or appended to the file if the cursor is at the end of the file). This is the normal operating mode of the editor. If you want to replace some existing text, you have to delete the old text before or after you type in the replacement text. The Backspace key can be used for deleting text: move the cursor to right after the text you want to delete and hit Backspace a number of times.

Hit the Enter or Return key to insert a line-break. For example, if the cursor was in the middle of a line and you hit Return, the line would be split into two lines with the cursor appearing at the beginning of the second line. Hit Backspace at the beginning of a line to eliminate a line-break.

Use the arrow keys to move around the file. If your keyboard doesn’t have arrow keys (or if they don’t work for some reason), use \F to move forwards (right), \B to move backwards (left), \P to move to the previous line (up), and \N to move to the next line (down). The right and left arrow keys simply move forwards or backwards one character at a time through the text: if you’re at the beginning of a line and you press left-arrow, you will end up at the end of the previous line. The up and down arrow keys move forwards and backwards by enough characters so that the cursor appears in the same column that it was in on the original line.

If you want to indent the text you enter, you can use the TAB key. This inserts a special control character which makes the characters which follow it begin at the next TAB STOP. TAB STOPS normally occur every 8 columns, but this can be changed with the \T D command. Python programmers often set TAB STOPS on every 4 columns.

If for some reason your terminal screen gets messed up (for example, if you receive a mail notice from biff), you can have the editor refresh the screen by hitting \R.

There are many other keys for deleting text and moving around the file. For example, hit \D to delete the character the cursor is on instead of deleting backwards like Backspace. \D will also delete a line-break if the cursor is at the end of a line. Type \Y to delete the entire line the cursor is on or \J to delete just from the cursor to the end of the line.

Hit \A to move the cursor to the beginning of the line it’s on. Hit \E to move the cursor to the end of the line. Hit \U or \V for scrolling the cursor up or down 1/2 a screen’s worth. "Scrolling" means that the text on the screen moves, but the cursor stays at the same place relative to the screen. Hit \K U or \K V to move the cursor to the beginning or the end of the file. Look at the help screens in the editor to find even more delete and movement commands.
If you make a mistake, you can hit ^_ to "undo" it. On most keyboards you hit just ^- to get ^_, but on some you might have to hold both the Shift and Control keys down at the same time to get it. If you "undo" too much, you can "redo" the changes back into existence by hitting ^^ (type this with just ^6 on most keyboards).

If you were editing in one place within the file, and you then temporarily had to look or edit some other place within the file, you can get back to the original place by hitting ^K -. This command actually returns you to the last place you made a change in the file. You can step through a history of places with ^K - and ^K =, in the same way you can step through the history of changes with the "undo" and "redo" commands.

When you are done editing the file, hit ^K X to exit the editor. You will be prompted for a file name if you hadn't already named the file you were editing.

When you edit a file, you actually edit only a copy of the file. So if you decide that you don't want the changes you made to a file during a particular edit session, you can hit ^C to exit the editor without saving them.

If you edit a file and save the changes, a "backup" copy of that file is created in the current directory, with a ~ appended to the name, which contains the original version of the file.

**Word wrap and formatting**

If you type past the right edge of the screen in a C language or PASCAL file, the screen will scroll to the right to follow the cursor. If you type past the right edge of the screen in a normal file (one whose name doesn't end in .c, .h or .p), JOE will automatically wrap the last word onto the next line so that you don't have to hit Return. This is called word-wrap mode. Word-wrap can be turned on or off with the ^TW command. JOE's initialisation file is usually set up so that this mode is automatically turned on for all non-program files. See the section below on the joerc file to change this and other defaults.

Aside for Word-wrap mode, JOE does not automatically keep paragraphs formatted like some word-processors. Instead, if you need a paragraph to be reformatted, hit ^K J. This command "fills in" the paragraph that the cursor is in, fitting as many words in a line as is possible. A paragraph, in this case, is a block of text separated above and below by a blank line.

The margins which JOE uses for paragraph formatting and word-wrap can be set with the ^TL and ^TR commands. If the left margin is set to a value other than 1, then when you start typing at the beginning of a line, the cursor will immediately jump to the left margin.

If you want to center a line within the margins, use the ^KA command.

**Over-type mode**

Sometimes it's tiresome to have to delete old text before or after you insert new text. This happens, for example, when you are changing a table and you want to maintain the column position of the right side of the table. When this occurs, you can put the editor in over-type mode with ^TT. When the editor is in this mode, the characters you type in replace existing characters, in the way an idealised typewriter would. Also, Backspace simply moves left instead of deleting the character to the left, when it's not at the end or beginning of a line. Over-type mode is not the natural way of dealing with text electronically, so you should go back to insert-mode as soon as possible by typing ^TT again.

If you need to insert while you're in over-type mode, hit ^@. This inserts a single SPACE into the text.
Control and Meta characters

Each character is represented by a number. For example, the number for 'A' is 65 and the number for '1' is 49. All of the characters which you normally see have numbers in the range of 32 - 126 (this particular arbitrary assignment between characters and numbers is called the ASCII character set). The numbers outside of this range, from 0 to 255, aren't usually displayed, but sometimes have other special meanings. The number 10, for example, is used for the line-breaks. You can enter these special, non-displayed control characters by first hitting ' and then hitting a character in the range @ A B C ... X Y Z [ ] ` _ to get the number 0 - 31, and ? to get 127. For example, if you hit 'J', you'll insert a line-break character, or if you hit 'I', you'll insert a TAB character (which does the same thing the TAB key does). A useful control character to enter is 12 ('L), which causes most printers to advance to the top of the page. You'll notice that JOE displays this character as an underlined L. You can enter the characters above 127, the meta characters, by first hitting \. This adds 128 to the next (possibly control) character entered. JOE displays characters above 128 in inverse-video. Some foreign languages, which have more letters than English, use the meta characters for the rest of their alphabet. You have to put the editor in ASIS mode (described later) to have these passed untranslated to the terminal.

Prompts

If you hit TAB at any file name prompt, joe will attempt to complete the name you entered as much as possible. If it couldn't complete the entire name, because there are more than one possible completions, joe beeps. If you hit TAB again, joe list the completions. You can use the arrow keys to move around this directory menu and press RETURN or SPACE to select an item. If you press the first letter of one of the directory entries, it will be selected, or if more than one entry has the same first letter, the cursor will jump between those entries. If you select a subdirectory or .., the directory name is appended to the prompt and the new directory is loaded into the menu. You can hit Backspace to go back to the previous directory.

Most prompts record a history of the responses you give them. You can hit up and down arrow to step through these histories.

Prompts are actually single line windows with no status line, so you can use any editing command that you normally use on text within the prompts. The prompt history is actually just other lines of the same "prompt file". Thus you can can search backwards though the prompt history with the normal \K F command if you want.

Since prompts are windows, you can also switch out of them with \K P and \K N.

Where am I?

Hit \K SPACE to have JOE report the line number, column number, and byte number on the last line of the screen. The number associated with the character the cursor is on (its ASCII code) is also shown. You can have the line number and/or column number always displayed on the status line by setting placing the appropriate escape sequences in the status line setup strings. Edit the joerc file for details.

File operations

You can hit \K D to save the current file (possibly under a different name from what the file was called originally). After the file is saved, you can hit \K E to edit a different file.

If you want to save only a selected section of the file, see the section on Blocks below.

If you want to include another file in the file you’re editing, use \K R to insert it.
Temporarily suspending the editor

If you need to temporarily stop the editor and go back to the shell, hit `^K Z`. You might want to do this to stop whatever you're editing and answer an e-mail message or read this man page, for example. You have to type `fg` or `exit` (you'll be told which when you hit `^K Z`) to return to the editor.

Searching for text

Hit `^K F` to have the editor search forwards or backwards for a text fragment (string) for you. You will be prompted for the text to search for. After you hit `Return`, you are prompted to enter options. You can just hit `Return` again to have the editor immediately search forwards for the text, or you can enter one or more of these options:

- **b** Search backwards instead of forwards.
- **i** Treat uppercase and lowercase letters as the same when searching. Normally uppercase and lowercase letters are considered to be different.
- **nnn** (where **nnn** is a number) If you enter a number, JOE searches for the Nth occurrence of the text. This is useful for going to specific places in files structured in some regular manner.
- **r** Replace text. If you enter the **r** option, then you will be further prompted for replacement text. Each time the editor finds the search text, you will be prompted as to whether you want to replace the found search text with the replacement text. You hit: y to replace the text and then find the next occurrence, n to not replace this text, but to then find the next occurrence, l to replace the text and then stop searching, r to replace all of the remaining occurrences of the search text in the remainder of the file without asking for confirmation (subject to the **nnn** option above), or `^C` to stop searching and replacing.

You can hit `^L` to repeat the previous search.

Regular Expressions

A number of special character sequences may be entered as search text:

- **\*** This finds zero or more characters. For example, if you give A\*B as the search text, JOE will try to find an A followed by any number of characters and then a B.

- **\?** This finds exactly one character. For example, if you give A\?B as the search text, JOE will find AxB, but not AB or AXXB.

- **\^ \$** These match the beginning and end of a line. For example, if you give \^test\$, then JOE with find test on a line by itself.

- **\< \>** These match the beginning and end of a word. For example, if you give \<\is\>\>, then joe will find whole words which have the sub-string is within them.

- **\[\]** This matches any single character which appears within the brackets. For example, if \[Tt\]his is entered as the search string, then JOE finds both This and this. Ranges of characters can be
entered within the brackets. For example, \[A-Z\] finds any uppercase letter. If the first character given in the brackets is ^, then JOE tries to find any character not given in the the brackets.

\c This works like \*, but matches a balanced C-language expression. For example, if you search for malloc(c), then JOE will find all function calls to malloc, even if there was a ) within the parenthesis.

\+ This finds zero or more of the character which immediately follows the \+. For example, if you give \[ ]\+/\[ ], where the characters within the brackets are both SPACE and TAB, then JOE will find whitespace.

\% Matches a single \.

\n This finds the special end-of-line or line-break character.

A number of special character sequences may also be given in the replacement string:

\& This gets replaced by the text which matched the search string. For example, if the search string was \<\*\>, which matches words, and you give "\&", then joe will put quote marks around words.

\0 - \9 These get replaced with the text which matched the Nth \*, ?, +, c, +, or [\...] in the search string.

\\ Use this if you need to put a \ in the replacement string.

\n Use this if you need to put a line-break in the replacement string.

Some examples:

Suppose you have a list of addresses, each on a separate line, which starts with "Address:" and has each element separated by commas. Like so:

Address: S. Holmes, 221b Baker St., London, England

If you wanted to rearrange the list, to get the country first, then the city, then the person's name, and then the address, you could do this:

Type ^K F to start the search, and type:

Address:\*,\*,\*,\*,\*$

to match "Address:", the four comma-separated elements, and then the end of the line. When asked for options, you would type r to replace the string, and then type:

Address:\3,\2,\0,\1

To shuffle the information the way you want it. After hitting return, the search would begin, and the sample line would be changed to:
Blocks

If you want to move, copy, save or delete a specific section of text, you can do it with highlighted blocks. First, move the cursor to the start of the section of text you want to work on, and press \texttt{ˆK B}. Then move the cursor to the character just after the end of the text you want to affect and press \texttt{ˆK K}. The text between the \texttt{ˆK B} and \texttt{ˆK K} should become highlighted. Now you can move your cursor to someplace else in your document and press \texttt{ˆK M} to move the highlighted text there. You can press \texttt{ˆK C} to make a copy of the highlighted text and insert it to where the cursor is positioned. \texttt{ˆK Y} to deletes the highlighted text. \texttt{ˆK W}, writes the highlighted text to a file.

A very useful command is \texttt{ˆK /}, which filters a block of text through a unix command. For example, if you select a list of words with \texttt{ˆK B} and \texttt{ˆK K}, and then type \texttt{ˆK / sort}, the list of words will be sorted. Another useful unix command for \texttt{ˆK /}, is \texttt{tr}. If you type \texttt{ˆK / tr a-z A-Z}, then all of the letters in the highlighted block will be converted to uppercase.

After you are finished with some block operations, you can just leave the highlighting on if you don’t mind it (of course, if you accidentally hit \texttt{ˆK Y} without noticing...). If it really bothers you, however, just hit \texttt{ˆK B ˆK K}, to turn the highlighting off.

Indenting program blocks

Auto-indent mode toggled with the \texttt{ˆT I} command. The \texttt{joerc} is normally set up so that files with names ending with .p, .c or .h have auto-indent mode enabled. When auto-indent mode is enabled and you hit \texttt{Return}, the cursor will be placed in the same column that the first non-SPACE/TAB character was in on the original line.

You can use the \texttt{ˆK ,} and \texttt{ˆK .} commands to shift a block of text to the left or right. If no highlighting is set when you give these commands, the program block the cursor is located in will be selected, and will be moved by subsequent \texttt{ˆK ,} and \texttt{ˆK .} commands. The number of columns these commands shift by can be set through a \texttt{ˆT} option.

Windows

You can edit more than one file at the same time or edit two or more different places of the same file. To do this, hit \texttt{ˆK O}, to split the screen into two windows. Use \texttt{ˆK P} or \texttt{ˆK N} to move the cursor into the top window or the lower window. Use \texttt{ˆK E} to edit a new file in one of the windows. A window will go away when you save the file with \texttt{ˆK X} or abort the file with \texttt{ˆC}. If you abort a file which exists in two windows, one of the window goes away, not the file.

You can hit \texttt{ˆK O} within a window to create even more windows. If you have too many windows on the screen, but you don’t want to eliminate them, you can hit \texttt{ˆK I}. This will show only the window the cursor is in, or if there was only one window on the screen to begin with, try to fit all hidden windows on the screen. If there are more windows than can fit on the screen, you can hit \texttt{ˆK N} on the bottom-most window or \texttt{ˆK P} on the top-most window to get to them.

If you gave more than one file name to JOE on the command line, each file will be placed in a different window.

You can change the height of the windows with the \texttt{ˆK G} and \texttt{ˆK T} commands.
Keyboard macros

Macros allow you to record a series of keystrokes and replay them with the press of two keys. This is useful to automate repetitive tasks. To start a macro recording, hit ^K followed by a number from 0 to 9. The status line will display (Macro n recording...). Now, type in the series of keystrokes that you want to be able to repeat. The commands you type will have their usual effect. Hit ^K to stop recording the macro. Hit ^K followed by the number you recorded the macro in to execute one iteration of the keystrokes.

For example, if you want to put "**" in front of a number of lines, you can type:

^K ['^A ** <down arrow> ^K ]

Which starts the macro recording, moves the cursor to the beginning of the line, inserts "**", moves the cursor down one line, and then ends the recording. Since we included the key-strokes needed to position the cursor on the next line, we can repeatedly use this macro without having to move the cursor ourselves, something you should always keep in mind when recording a macro.

If you find that the macro you are recording itself has a repeated set of key-strokes in it, you can record a macro within the macro, as long as you use a different macro number. Also you can execute previously recorded macros from within new macros.

Repeat

You can use the repeat command, ^K \, to repeat a macro, or any other edit command or even a normal character, a specified number of times. Hit ^K \, type in the number of times you want the command repeated and press Return. The next edit command you now give will be repeated that many times.

For example, to delete the next 20 lines of text, type:

^K 20<return>^Y

Rectangle mode

Type ^T X to have ^K B and ^K K select rectangular blocks instead of stream-of-text blocks. This mode is useful for moving, copying, deleting or saving columns of text. You can also filter columns of text with the ^K / command- if you want to sort a column, for example. The insert file command, ^K R is also effected.

When rectangle mode is selected, over-type mode is also useful (^T T). When over-type mode is selected, rectangles will replace existing text instead of getting inserted before it. Also the delete block command (^K Y) will clear the selected rectangle with SPACES and TABs instead of deleting it. Over-type mode is especially useful for the filter block command (^K /), since it will maintain the original width of the selected column.

Tag search

If you are editing a large C program with many source files, you can use the ctags program to generate a tags file. This file contains a list of program symbols and the files and positions where the symbols are defined. The ^K ; command can be used to lookup a symbol (functions, defined constants, etc.), load the file where the symbol is defined into the current window and position the cursor to where the symbol is defined. ^K ; prompts you for the symbol you want, but uses the symbol the cursor was on as a default. Since ^K ; loads the definition file into the current window, you probably want to split the window first with ^K O, to have both the original file and the definition file loaded.
Shell windows
Hit `K V' to run a command shell in one of JOE’s windows. When the cursor is at the end of a shell window (use `K V if it’s not), whatever you type is passed to the shell instead of the window. Any output from the shell or from commands executed in the shell is appended to the shell window (the cursor will follow this output if it’s at the end of the shell window). This command is useful for recording the results of shell commands—e.g., the output of `make`, the result of `grep` a set of files for a string, or directory listings from FTP sessions. Besides typeable characters, the keys `C, Backspace, DEL, Return and `D are passed to the shell. Type the shell `exit command to stop recording shell output. If you press `C in a shell window, when the cursor is not at the end of the window, the shell is killed.

Environment variables
For JOE to operate correctly, a number of other environment settings must be correct. The throughput (baud rate) of the connection between the computer and your terminal must be set correctly for JOE to update the screen smoothly and allow typeahead to defer the screen update. Use the `stty nnn command to set this. You want to set it as close as possible to actual throughput of the connection. For example, if you are connected via a 1200 baud modem, you want to use this value for `stty. If you are connected via 14.4k modem, but the terminal server you are connected to connects to the computer a 9600 baud, you want to set your speed as 9600 baud. The special baud rate of 38400 or `extb is used to indicate that you have a very-high speed connection, such as a memory mapped console or an X-window terminal emulator. If you can’t use `stty to set the actual throughput (perhaps because of a modem communicating with the computer at a different rate than it’s communicating over the phone line), you can put a numeric value in the `BAUD environment variable instead (use `setenv BAUD 9600 for csh or `BAUD=9600; export BAUD for sh).

The `SHELL or `EXECSHELL environment variable must be set to the full pathname of a shell executable that accepts the `-i (interactive) and `-c (run a command) arguments of the Korn Shell; otherwise, `/bin/sh is used.

The `TERM environment variable must be set to the type of terminal you’re using. If the size (number of lines/columns) of your terminal is different from what is reported in the TERMcap or TERMINFO entry, you can set this with the `stty rows nn cols nn command, or by setting the `LINES and `COLUMNS environment variables.

The `xterm−xfree86 terminal allows automatic entering and leaving of the bracketed paste mode.

The `JOETERM environment variable may be set to override the regular `TERM environment variable for specifying your terminal type.

JOE uses two character maps for its operation: the terminal I/O character map, which determines how characters are sent to the terminal and whether the %a/%A message specifiers use UCS, and the file encoding, which can be specified per file using the `−encoding option and changed with the `T E command, and which defaults to the terminal I/O character map, which, in turn, is determined from the current locale, if the system supports such, otherwise the `LC_ALL, `LC_CTYPE and `LANG environment variables (if they contain a period, only the part after it and before an optional "at sign" is used); on cygwin32 before 1.7.2, the codepage is used instead if the POSIX locale environment variables are empty; the environment variable `JOECHARMAP can be used to manually force one overriding all methods described above, and can be used together with `−encoding to specify a different default file character map.

JOE normally expects that flow control between the computer and your terminal to use `S/Q handshaking (i.e., if the computer is sending characters too fast for your terminal, your terminal sends `S to stop the output and `Q to restart it). If the flow control uses out-of-band or hardware handshaking or if your terminal is fast enough to always keep up with the computer output and you wish to map `S/Q to edit commands, you can set the environment variable `NOXON to have JOE attempt to turn off `S/Q
handshaking. If the connection between the computer and your terminal uses no handshaking and your terminal is not fast enough to keep up with the output of the computer, you can set the environment variable `DOPADDING` to have JOE slow down the output by interspersing PAD characters between the terminal screen update sequences.

**Filenames**

Wherever JOE expects you to enter a file name, whether on the command line or in prompts within the editor, you may also type:

!command

Read or write data to or from a shell command. For example, use `joe 'ls'` to get a copy of your directory listing to edit or from within the editor use `''K D !mail jhallen@world.std.com` to send the file being edited to me.

>>filename

Use this to have JOE append the edited text to the end of the file "filename."

filename,START,SIZE

Use this to access a fixed section of a file or device. `START` and `SIZE` may be entered in decimal (ex.: 123) octal (ex.: 0777) or hexadecimal (ex.: 0xFF). For example, use `joe /dev/fd0,508,2` to edit bytes 508 and 509 of the first floppy drive in Linux.

- Use this to get input from the standard input or to write output to the standard output. For example, you can put joe in a pipe of commands: `quota -v | joe - | mail root`, if you want to complain about your low quota.

**The joerc file**

`''T` options, the help screens and the key-sequence to editor command bindings are all defined in JOE's initialisation file. If you make a copy of this file (which normally resides in `/etc/jupp/joerc`) to `$HOME/.joerc`, you can customise these settings to your liking. The syntax of the initialisation file should be fairly obvious, and there are further instruction in it.

**Acknowledgments**

JOE was written by Joseph H. Allen. If you have bug reports or questions, e-mail them to jhallen@world.std.com. Larry Foard (entropy@world.std.com) and Gary Gray (ggray@world.std.com) also helped with the creation of JOE. Thorsten "mirabilos" Glaser (tg@mirbsd.org) created JUPP, and the 16-bit MS-DOS version of JUPP 2.8 was compiled by Andreas Totlis (atotlis@t-online.de).

**Bugs**

This manual page describes only the JOE flavour; documentation for JUPP is especially missing.