JOVE Manual for UNIX Users

Jonathan Payne
(revised for 4.3BSD by Doug Kingston and Mark Seiden)

1. Introduction

JOVE* is an advanced, self-documenting, customizable real-time display editor. It (and this tutorial introduction) are based on the original EMACS editor and user manual written at M.I.T. by Richard Stallman+.

JOVE is considered a *display* editor because normally the text being edited is visible on the screen and is updated automatically as you type your commands.

It's considered a *real-time* editor because the display is updated very frequently, usually after each character or pair of characters you type. This minimizes the amount of information you must keep in your head as you edit.

JOVE is *advanced* because it provides facilities that go beyond simple insertion and deletion: filling of text; automatic indentations of programs; view more than one file at once; and dealing in terms of characters, words, lines, sentences and paragraphs. It is much easier to type one command meaning "go to the end of the paragraph" than to find the desired spot with repetition of simpler commands.

Self-documenting means that at almost any time you can easily find out what a command does, or to find all the commands that pertain to a topic.

Customizable means that you can change the definition of JOVE commands in little ways. For example, you can rearrange the command set; if you prefer to use arrow keys for the four basic cursor motion commands (up, down, left and right), you can. Another sort of customization is writing new commands by combining built in commands.

2. The Organization of the Screen

JOVE divides the screen up into several sections. The biggest of these sections is used to display the text you are editing. The terminal's cursor shows the position of *point*, the location at which editing takes place. While the cursor appears to point *at* a character, point should be thought of as between characters; it points *before* the character that the cursor appears on top of. Terminals have only one cursor, and when output is in progress it must appear where the typing is being done. This doesn't mean that point is moving; it is only that JOVE has no way of showing you the location of point except when the terminal is idle.

The lines of the screen are usually available for displaying text but sometimes are pre-empted by typeout from certain commands (such as a listing of all the editor commands). Most of the time, output from commands like these is only desired for a short period of time, usually just long enough to glance at it. When you have finished looking at the output, you can type Space to make your text reappear. (Usually a Space that you type inserts itself, but when there is typeout on the screen, it does nothing but get rid of that). Any other command executes normally, *after* redrawing your text.

2.1. The Message Line

The bottom line on the screen, called the *message line*, is reserved for printing messages and for accepting input from the user, such as filenames or search strings. When JOVE prompts for input, the cursor will temporarily appear on the bottom line, waiting for you to type a string. When you have finished typing your input, you can type a Return to send it to JOVE. If you change your mind about running the command that is waiting for input, you can type Control-G to abort, and you can continue with your editing.

When JOVE is prompting for a filename, all the usual editing facilities can be used to fix typos and such; in addition, JOVE has the following extra functions:

^N Insert the next filename from the argument list.

---

*JOVE stands for Jonathan's Own Version of Emacs.
+Although JOVE is meant to be compatible with EMACS, and indeed many of the basic commands are very similar, there are some major differences between the two editors, and you should not rely on their behaving identically.

ˆP Insert the previous filename from the argument list.

ˆR Insert the full pathname of the file in the current buffer.

Sometimes you will see **--more--** on the message line. This happens when typeout from a command is too long to fit in the screen. It means that if you type a Space the next screenful of typeout will be printed. If you are not interested, typing anything but a Space will cause the rest of the output to be discarded. Typing C-G will discard the output and print *Aborted* where the **--more--** was. Typing any other command will discard the rest of the output and also execute the command.

The message line and the list of filenames from the shell command that invoked JOVE are kept in a special buffer called *Minibuf* that can be edited like any other buffer.

2.2. The Mode Line

At the bottom of the screen, but above the message line, is the *mode line*. The mode line format looks like this:

JOVE (major minor) Buffer: bufr "file" *

major is the name of the current *major mode*. At any time, JOVE can be in only one major mode at a time. Currently there are only four major modes: *Fundamental*, *Text*, *Lisp* and *C*.

minor is a list of the minor modes that are turned on. **Abbrev** means that *Word Abbrev* mode is on; **AI** means that *Auto Indent* mode is on; **Fill** means that *Auto Fill* mode is on; **OvrWt** means that *Over Write* mode is on. **Def** means that you are in the process of defining a keyboard macro. This is not really a mode, but it's useful to be reminded about it. The meanings of these modes are described later in this document.

bufr is the name of the currently selected *buffer*. Each buffer has its own name and holds a file being edited; this is how JOVE can hold several files at once. But at any given time you are editing only one of them, the *selected* buffer. When we speak of what some command does to "the buffer", we are talking about the currently selected buffer. Multiple buffers makes it easy to switch around between several files, and then it is very useful that the mode line tells you which one you are editing at any time. (You will see later that it is possible to divide the screen into multiple *windows*, each showing a different buffer. If you do this, there is a mode line beneath each window.)

file is the name of the file that you are editing. This is the default filename for commands that expect a filename as input.

The asterisk at the end of the mode line means that there are changes in the buffer that have not been saved in the file. If the file has not been changed since it was read in or saved, there is no asterisk.

3. Command Input Conventions

3.1. Notational Conventions for ASCII Characters

In this manual, "Control" characters (that is, characters that are typed with the Control key and some other key at the same time) are represented by "C-" followed by another character. Thus, C-A is the character you get when you type A with the Control key (sometimes labeled CTRL) down. Most control characters when present in the JOVE buffer are displayed with a caret; thus, ˆA for C-A. Rubout (or DEL) is displayed as ˆ?, escape as ˆ[.

3.2. Command and Filename Completion

When you are typing the name of a JOVE command, you need type only enough letters to make the name unambiguous. At any point in the course of typing the name, you can type question mark (?) to see a list of all the commands whose names begin with the characters you've already typed; you can type Space to have JOVE supply as many characters as it can; or you can type Return to complete the command if there is only one possibility. For example, if you have typed the letters "*au*" and you then type a question mark, you will see the list

```
auto-execute-command
auto-execute-macro
auto-fill-mode
auto-indent-mode
```

If you type a Return at this point, JOVE will complain by ringing the bell, because the letters you've typed do not unambiguously specify a single command. But if you type Space, JOVE will supply the characters "*to-*" because all commands that begin "*au*" also begin "*auto-*". You could then type the letter "*f*" followed by either Space or Return, and JOVE would complete the entire command.

Whenever JOVE is prompting you for a filename, say in the *find-file* command, you also need only type enough of the name to make it unambiguous with respect to files that already exist. In this case, question mark and Space work just as they do in command completion, but Return always accepts the name just as you've typed it, because you might want to create a new file with a name similar to that of an existing file.

4. Commands and Variables

JOVE is composed of *commands* which have long names such as *next-line*. Then *keys* such as C-N are connected to commands through the *command dispatch table*. When we say that C-N moves the cursor down a line, we are glossing over a distinction which is unimportant for ordinary use, but essential for simple customization: it is the command *next-line* which knows how to move a down line, and C-N moves down a line because it is connected to that command. The name for this connection is a *binding*; we say that the key C-N *is bound to* the command *next-line*.

Not all commands are bound to keys. To invoke a command that isn't bound to a key, you can type the sequence ESC X, which is bound to the command *execute-named-command*. You will then be able to type the name of whatever command you want to execute on the message line.

Sometimes the description of a command will say "to change this, set the variable *mumble–foo*". A variable is a name used to remember a value. JOVE contains variables which are there so that you can change them if you want to customize. The variable's value is examined by some command, and changing that value makes the command behave differently. Until you are interesting in customizing JOVE, you can ignore this information.

4.1. Prefix Characters

Because there are more command names than keys, JOVE provides *prefix characters* to increase the number of commands that can be invoked quickly and easily. When you type a prefix character JOVE will wait for another character before deciding what to do. If you wait more than a second or so, JOVE will print the prefix character on the message line as a reminder and leave the cursor down there until you type your next character. There are two prefix characters built into JOVE: Escape and Control-X. How the next character is interpreted depends on which prefix character you typed. For example, if you type Escape followed by B you'll run *backward-word*, but if you type Control-X followed by B you'll run *select-buffer*. Elsewhere in this manual, the Escape key is indicated as "ESC", which is also what JOVE displays on the message line for Escape.

4.2. Help

To get a list of keys and their associated commands, you type ESC X *describe-bindings*. If you want to describe a single key, ESC X *describe-key* will work. A description of an individual command is available by using ESC X *describe-command*, and descriptions of variables by using ESC X *describe-variable*. If you can't remember the name of the thing you want to know about, ESC X *apropos* will tell you if a command or variable has a given string in its name. For example, ESC X *apropos describe* will list the names of the four describe commands mentioned briefly in this section.

5. Basic Editing Commands

5.1. Inserting Text

To insert printing characters into the text you are editing, just type them. All printing characters you type are inserted into the text at the cursor (that is, at *point*), and the cursor moves forward. Any characters after the cursor move forward too. If the text in the buffer is FOOBAR, with the cursor before the B, then if you type XX, you get FOOXXBAR, with the cursor still before the B.

To correct text you have just inserted, you can use Rubout. Rubout deletes the character *before* the cursor (not the one that the cursor is on top of or under; that is the character *after* the cursor). The cursor and all characters after it move backwards. Therefore, if you typing a printing character and then type Rubout, they cancel out.

To end a line and start typing a new one, type Return. Return operates by inserting a *line-separator*, so if you type Return in the middle of a line, you break the line in two. Because a line-separator is just a single character, you can type Rubout at the beginning of a line to delete the line-separator and join it with the preceding line.

As a special case, if you type Return at the end of a line and there are two or more empty lines just below it, JOVE does not insert a line-separator but instead merely moves to the next (empty) line. This behavior is convenient when you want to add several lines of text in the middle of a buffer. You can use the Control-O (*newline-and-backup*) command to "open" several empty lines at once; then you can insert the new text, filling up these empty lines. The advantage is that JOVE does not have to redraw the bottom part of the screen for each Return you type, as it would ordinarily. That "redisplay" can be both slow and distracting.

If you add too many characters to one line, without breaking it with Return, the line will grow too long to display on one screen line. When this happens, JOVE puts an "!" at the extreme right margin, and doesn't bother to display the rest of the line unless the cursor happens to be in it. The "!" is not part of your text; conversely, even though you can't see the rest of your line, it's still there, and if you break the line, the "!" will go away.

Direct insertion works for printing characters and space, but other characters act as editing commands and do not insert themselves. If you need to insert a control character, Escape, or Rubout, you must first *quote* it by typing the Control-Q command first.

5.2. Moving the Cursor

To do more than insert characters, you have to know how to move the cursor. Here are a few of the commands for doing that.

C-A	Move to the beginning of the line.
C-E	Move to the end of the line.
C-F	Move forward over one character.
C-B	Move backward over one character.
C-N	Move down one line, vertically. If you start in the middle of one line, you end in the middle of the next.
C-P	Move up one line, vertically.
ESC <	Move to the beginning of the entire buffer.
ESC >	Move to the end of the entire buffer.
ESC ,	Move to the beginning of the visible window.
ESC .	Move to the end of the visible window.

5.3. Erasing Text

Rubout	Delete the character before the cursor.
C-D	Delete the character after the cursor.
C-K	Kill to the end of the line.

You already know about the Rubout command which deletes the character before the cursor. Another command, Control-D, deletes the character after the cursor, causing the rest of the text on the line to shift left. If Control-D is typed at the end of a line, that line and the next line are joined together.

To erase a larger amount of text, use the Control-K command, which kills a line at a time. If Control-K is done at the beginning or middle of a line, it kills all the text up to the end of the line. If Control-K is done at the end of a line, it joins that line and the next line. If Control-K is done twice, it kills the rest of the line and the line separator also.

5.4. Files — Saving Your Work

The commands above are sufficient for creating text in the JOVE buffer. The more advanced JOVE commands just make things easier. But to keep any text permanently you must put it in a *file*. Files are the objects which UNIX†

† UNIX is a trademark of Bell Laboratories.

uses for storing data for a length of time. To tell JOVE to read text into a file, choose a filename, such as *foo.bar*, and type C-X C-R *foo.bar*<return>. This reads the file *foo.bar* so that its contents appear on the screen for editing. You can make changes, and then save the file by typing C-X C-S (save-file). This makes the changes permanent and actually changes the file *foo.bar*. Until then, the changes are only inside JOVE, and the file *foo.bar* is not really changed. If the file *foo.bar* doesn't exist, and you want to create it, read it as if it did exist. When you save your text with C-X C-S the file will be created.

5.5. Exiting and Pausing — Leaving JOVE

The command C-X C-C (*exit-jove*) will terminate the JOVE session and return to the shell. If there are modified but unsaved buffers, JOVE will ask you for confirmation, and you can abort the command, look at what buffers are modified but unsaved using C-X C-B (*list-buffers*), save the valuable ones, and then exit. If what you want to do, on the other hand, is *preserve* the editing session but return to the shell temporarily you can (under Berkeley UNIX only) issue the command ESC S (*pause-jove*), do your UNIX work within the c-shell, then return to JOVE using the *fg* command to resume editing at the point where you paused. For this sort of situation you might consider using an *interactive shell* (that is, a shell in a JOVE window) which lets you use editor commands to manipulate your UNIX commands (and their output) while never leaving the editor. (The interactive shell feature is described below.)

5.6. Giving Numeric Arguments to JOVE Commands

Any JOVE command can be given a *numeric argument*. Some commands interpret the argument as a repetition count. For example, giving an argument of ten to the C-F command (forward-character) moves forward ten characters. With these commands, no argument is equivalent to an argument of 1.

Some commands use the value of the argument, but do something peculiar (or nothing) when there is no argument. For example, ESC G (*goto-line*) with an argument **n** goes to the beginning of the **n**'th line. But ESC G with no argument doesn't do anything. Similarly, C-K with an argument kills that many lines, including their line separators. Without an argument, C-K when there is text on the line to the right of the cursor kills that text; when there is no text after the cursor, C-K deletes the line separator.

The fundamental way of specifying an argument is to use ESC followed by the digits of the argument, for example, ESC 123 ESC G to go to line 123. Negative arguments are allowed, although not all of the commands know what to do with one.

Typing C-U means do the next command four times. Two such C-U's multiply the next command by sixteen. Thus, C-U C-U C-F moves forward sixteen characters. This is a good way to move forward quickly, since it moves about 1/4 of a line on most terminals. Other useful combinations are: C-U C-U C-N (move down a good fraction of the screen), C-U C-U C-O (make "a lot" of blank lines), and C-U C-K (kill four lines — note that typing C-K four times would kill 2 lines).

There are other, terminal-dependent ways of specifying arguments. They have the same effect but may be easier to type. If your terminal has a numeric keypad which sends something recognizably different from the ordinary digits, it is possible to program JOVE to to allow use of the numeric keypad for specifying arguments.

5.7. The Mark and the Region

In general, a command that processes an arbitrary part of the buffer must know where to start and where to stop. In JOVE, such commands usually operate on the text between point and *the mark*. This body of text is called *the region*. To specify a region, you set point to one end of it and mark at the other. It doesn't matter which one comes earlier in the text.

C-@	Set the mark where point is.
C-X C-X	Interchange mark and point.

For example, if you wish to convert part of the buffer to all upper-case, you can use the C-X C-U command, which operates on the text in the region. You can first go to the beginning of the text to be capitalized, put the mark there, move to the end, and then type C-X C-U. Or, you can set the mark at the end of the text, move to the beginning, and then type C-X C-U. C-X C-U runs the command *case-region-upper*, whose name signifies that the region, or everything between point and mark, is to be capitalized.

The way to set the mark is with the C-@ command or (on some terminals) the C-Space command. They set the mark where point is. Then you can move point away, leaving mark behind. When the mark is set, "[Point pushed]" is printed on the message line.

Since terminals have only one cursor, there is no way for JOVE to show you where the mark is located. You have to remember. The usual solution to this problem is to set the mark and then use it soon, before you forget where it is. But you can see where the mark is with the command C-X C-X which puts the mark where point was and point where mark was. The extent of the region is unchanged, but the cursor and point are now at the previous location of the mark.

5.8. The Ring of Marks

Aside from delimiting the region, the mark is also useful for remembering a spot that you may want to go back to. To make this feature more useful, JOVE remembers 16 previous locations of the mark. Most commands that set the mark push the old mark onto this stack. To return to a marked location, use C-U C-@. This moves point to where the mark was, and restores the mark from the stack of former marks. So repeated use of this command moves point to all of the old marks on the stack, one by one. Since the stack is actually a ring, enough uses of C-U C-@ bring point back to where it was originally.

Some commands whose primary purpose is to move point a great distance take advantage of the stack of marks to give you a way to undo the command. The best example is ESC <, which moves to the beginning of the buffer. If there are more than 22 lines between the beginning of the buffer and point, ESC < sets the mark first, so that you can use C-U C-@ or C-X C-X to go back to where you were. You can change the number of lines from 22 since it is kept in the variable *mark-threshold*. By setting it to 0, you can make these commands always set the mark. By setting it to a very large number you can prevent these commands from ever setting the mark. If a command decides to set the mark, it prints the message *[Point pushed]*.

5.9. Killing and Moving Text

The most common way of moving or copying text with JOVE is to kill it, and get it back again in one or more places. This is very safe because the last several pieces of killed text are all remembered, and it is versatile, because the many commands for killing syntactic units can also be used for moving those units. There are also other ways of moving text for special purposes.

5.10. Deletion and Killing

Most commands which erase text from the buffer save it so that you can get it back if you change your mind, or move or copy it to other parts of the buffer. These commands are known as *kill* commands. The rest of the commands that erase text do not save it; they are known as *delete* commands. The delete commands include C-D and Rubout, which delete only one character at a time, and those commands that delete only spaces or line separators. Commands that can destroy significant amounts of nontrivial data generally kill. A command's name and description will use the words *kill* or *delete* to say which one it does.

C-D	Delete next character.
Rubout	Delete previous character.
ESC \	Delete spaces and tabs around point.
C-X C-O	Delete blank lines around the current line.
C-K	Kill rest of line or one or more lines.
C-W	Kill region (from point to the mark).
ESC D	Kill word.
ESC Rubout	Kill word backwards.
ESC K	Kill to end of sentence.
C-X Rubout	Kill to beginning of sentence.

5.11. Deletion

The most basic delete commands are C-D and Rubout. C-D deletes the character after the cursor, the one the cursor is "on top of" or "underneath". The cursor doesn't move. Rubout deletes the character before the cursor, and moves the cursor back. Line separators act like normal characters when deleted. Actually, C-D and Rubout aren't always *delete* commands; if you give an argument, they *kill* instead. This prevents you from losing a great deal of text by typing a large argument to a C-D or Rubout.

The other delete commands are those which delete only formatting characters: spaces, tabs, and line separators. ESC \ (*delete-white-space*) deletes all the spaces and tab characters before and after point. C-X C-O (*delete-blank-lines*) deletes all blank lines after the current line, and if the current line is blank deletes all the blank lines preceding the current line as well (leaving one blank line, the current line).

5.12. Killing by Lines

The simplest kill command is the C-K command. If issued at the beginning of a line, it kills all the text on the line, leaving it blank. If given on a line containing only white space (blanks and tabs) the line disappears. As a consequence, if you go to the front of a non-blank line and type two C-K's, the line disappears completely.

More generally, C-K kills from point up to the end of the line, unless it is at the end of a line. In that case, it kills the line separator following the line, thus merging the next line into the current one. Invisible spaces and tabs at the end of the line are ignored when deciding which case applies, so if point appears to be at the end of the line, you can be sure the line separator will be killed.

C-K with an argument of zero kills all the text before point on the current line.

5.13. Other Kill Commands

A kill command which is very general is C-W (*kill-region*), which kills everything between point and the mark.* With this command, you can kill and save contiguous characters, if you first set the mark at one end of them and go to the other end.

Other syntactic units can be killed, too; words, with ESC Rubout and ESC D; and, sentences, with ESC K and C-X Rubout.

5.14. Un-killing

Un-killing (yanking) is getting back text which was killed. The usual way to move or copy text is to kill it and then un-kill it one or more times.

C-Y	Yank (re-insert) last killed text.
ESC Y	Replace re-inserted killed text with the previously killed text.
ESC W	Save region as last killed text without killing.

Killed text is pushed onto a *ring buffer* called the *kill ring* that remembers the last 10 blocks of text that were killed. (Why it is called a ring buffer will be explained below). The command C-Y (*yank*) reinserts the text of the most recent kill. It leaves the cursor at the end of the text, and puts the mark at the beginning. Thus, a single C-Y undoes the C-W.

If you wish to copy a block of text, you might want to use ESC W (*copy-region*), which copies the region into the kill ring without removing it from the buffer. This is approximately equivalent to C-W followed by C-Y, except that ESC W does not mark the buffer as "changed" and does not cause the screen to be rewritten.

There is only one kill ring shared among all the buffers. After visiting a new file, whatever was last killed in the previous file is still on top of the kill ring. This is important for moving text between files.

5.15. Appending Kills

Normally, each kill command pushes a new block onto the kill ring. However, two or more kill commands immediately in a row (without any other intervening commands) combine their text into a single entry on the ring, so that a

*Often users switch this binding from C-W to C-X C-K because it is too easy to hit C-W accidentally.

single C-Y command gets it all back as it was before it was killed. This means that you don't have to kill all the text in one command; you can keep killing line after line, or word after word, until you have killed it all, and you can still get it all back at once.

Commands that kill forward from *point* add onto the end of the previous killed text. Commands that kill backward from *point* add onto the beginning. This way, any sequence of mixed forward and backward kill commands puts all the killed text into one entry without needing rearrangement.

5.16. Un-killing Earlier Kills

To recover killed text that is no longer the most recent kill, you need the ESC Y (*yank-pop*) command. The ESC Y command can be used only after a C-Y (yank) command or another ESC Y. It takes the un-killed text inserted by the C-Y and replaces it with the text from an earlier kill. So, to recover the text of the next-to-the-last kill, you first use C-Y to recover the last kill, and then discard it by use of ESC Y to move back to the previous kill.

You can think of all the last few kills as living on a ring. After a C-Y command, the text at the front of the ring is also present in the buffer. ESC Y "rotates" the ring bringing the previous string of text to the front and this text replaces the other text in the buffer as well. Enough ESC Y commands can rotate any part of the ring to the front, so you can get at any killed text so long as it is recent enough to be still in the ring. Eventually the ring rotates all the way around and the most recently killed text comes to the front (and into the buffer) again. ESC Y with a negative argument rotates the ring backwards.

When the text you are looking for is brought into the buffer, you can stop doing ESC Y's and the text will stay there. It's really just a copy of what's at the front of the ring, so editing it does not change what's in the ring. And the ring, once rotated, stays rotated, so that doing another C-Y gets another copy of what you rotated to the front with ESC Y.

If you change your mind about un-killing, C-W gets rid of the un-killed text, even after any number of ESC Y's.

6. Searching

The search commands are useful for finding and moving to arbitrary positions in the buffer in one swift motion. For example, if you just ran the spell program on a paper and you want to correct some word, you can use the search commands to move directly to that word. There are two flavors of search: *string search* and *incremental search*. The former is the default flavor—if you want to use incremental search you must rearrange the key bindings (see below).

6.1. Conventional Search

C-S	Search forward.
C-R	Search backward.

To search for the string "FOO" you type "C-S FOO<return>". If JOVE finds FOO it moves point to the end of it; otherwise JOVE prints an error message and leaves point unchanged. C-S searches forward from point so only occurrences of FOO after point are found. To search in the other direction use C-R. It is exactly the same as C-S except it searches in the opposite direction, and if it finds the string, it leaves point at the beginning of it, not at the end as in C-S.

While JOVE is searching it prints the search string on the message line. This is so you know what JOVE is doing. When the system is heavily loaded and editing in exceptionally large buffers, searches can take several (sometimes many) seconds.

JOVE remembers the last search string you used, so if you want to search for the same string you can type "C-S <return>". If you mistyped the last search string, you can type C-S followed by C-R. C-R, as usual, inserts the default search string into the minibuffer, and then you can fix it up.

6.2. Incremental Search

This search command is unusual in that is is *incremental*; it begins to search before you have typed the complete search string. As you type in the search string, JOVE shows you where it would be found. When you have typed enough characters to identify the place you want, you can stop. Depending on what you will do next, you may or may not need to terminate the search explicitly with a Return first.

The command to search is C-S (*i-search-forward*). C-S reads in characters and positions the cursor at the first occurrence of the characters that you have typed so far. If you type C-S and then F, the cursor moves in the text just after the next "F". Type an "O", and see the cursor move to after the next "FO". After another "O", the cursor is after the next "FOO". At the same time, the "FOO" has echoed on the message line.

If you type a mistaken character, you can rub it out. After the FOO, typing a Rubout makes the "O" disappear from the message line, leaving only "FO". The cursor moves back in the buffer to the "FO". Rubbing out the "O" and "F" moves the cursor back to where you started the search.

When you are satisfied with the place you have reached, you can type a Return, which stops searching, leaving the cursor where the search brought it. Also, any command not specially meaningful in searches stops the searching and is then executed. Thus, typing C-A would exit the search and then move to the beginning of the line. Return is necessary only if the next character you want to type is a printing character, Rubout, Return, or another search command, since those are the characters that have special meanings inside the search.

Sometimes you search for "FOO" and find it, but not the one you hoped to find. Perhaps there is a second FOO that you forgot about, after the one you just found. Then type another C-S and the cursor will find the next FOO. This can be done any number of times. If you overshoot, you can return to previous finds by rubbing out the C-S's.

After you exit a search, you can search for the same string again by typing just C-S C-S: one C-S command to start the search and then another C-S to mean "search again for the same string".

If your string is not found at all, the message line says "Failing I-search". The cursor is after the place where JOVE found as much of your string as it could. Thus, if you search for FOOT and there is no FOOT, you might see the cursor after the FOO in FOOL. At this point there are several things you can do. If your string was mistyped, you can rub some of it out and correct it. If you like the place you have found, you can type Return or some other JOVE command to "accept what the search offered". Or you can type C-G, which undoes the search altogether and positions you back where you started the search.

You can also type C-R at any time to start searching backwards. If a search fails because the place you started was too late in the file, you should do this. Repeated C-R's keep looking backward for more occurrences of the last search string. A C-S starts going forward again. C-R's can be rubbed out just like anything else.

6.3. Searching with Regular Expressions

In addition to the searching facilities described above, JOVE can search for patterns using regular expressions. The handling of regular expressions in JOVE is like that of *ed(1)* or *vi(1)*, but with some notable additions. The extra metacharacters understood by JOVE are \<, \>, \| and \{. The first two of these match the beginnings and endings of words; Thus the search pattern, "\<Exec" would match all words beginning with the letters "Exec".

An \| signals the beginning of an alternative — that is, the pattern "foo\|bar" would match either "foo" or "bar". The "curly brace" is a way of introducing several sub-alternatives into a pattern. It parallels the [] construct of regular expressions, except it specifies a list of alternative words instead of just alternative characters. So the pattern "foo\{bar,baz\}bie" matches "foobarbie" or "foobazbie".

JOVE only regards metacharacters as special if the variable *match-regular-expressions* is set to "on". The ability to have JOVE ignore these characters is useful if you're editing a document about patterns and regular expressions or when a novice is learning JOVE.

Another variable that affects searching is *case-ignore-search*. If this variable is set to "on" then upper case and lower case letters are considered equal.

7. Replacement Commands

Global search-and-replace operations are not needed as often in JOVE as they are in other editors, but they are available. In addition to the simple Replace operation which is like that found in most editors, there is a Query Replace operation which asks, for each occurrence of the pattern, whether to replace it.

7.1. Global replacement

To replace every occurrence of FOO after point with BAR, you can do, e.g., "ESC R FOO<return>BAR" as the *replace-string* command is bound to the ESC R. Replacement takes place only between point and the end of the buffer so if you want to cover the whole buffer you must go to the beginning first.

7.2. Query Replace

If you want to change only some of the occurrences of FOO, not all, then the global *replace-string* is inappropriate; Instead, use, e.g., "ESC Q FOO<return>BAR", to run the command *query-replace-string*. This displays each occurrence of FOO and waits for you to say whether to replace it with a BAR. The things you can type when you are shown an occurrence of FOO are:

Space	to replace the FOO.
Rubout	to skip to the next FOO without replacing this one.
Return	to stop without doing any more replacements.
Period	to replace this FOO and then stop.
! or P	to replace all remaining FOO’s without asking.
C-R or R	to enter a recursive editing level, in case the FOO needs to be edited rather than just replaced with a BAR. When you are done, exit the recursive editing level with C-X C-C and the next FOO will be displayed.
C-W	to delete the FOO, and then start editing the buffer. When you are finished editing whatever is to replace the FOO, exit the recursive editing level with C-X C-C and the next FOO will be displayed.
U	move to the last replacement and undo it.

Another alternative is using *replace-in-region* which is just like *replace-string* except it searches only within the region.

8. Commands for English Text

JOVE has many commands that work on the basic units of English text: words, sentences and paragraphs.

8.1. Word Commands

JOVE has commands for moving over or operating on words. By convention, they are all ESC commands.

ESC F	Move Forward over a word.
ESC B	Move Backward over a word.
ESC D	Kill forward to the end of a word.
ESC Rubout	Kill backward to the beginning of a word.

Notice how these commands form a group that parallels the character- based commands, C-F, C-B, C-D, and Rubout.

The commands ESC F and ESC B move forward and backward over words. They are thus analogous to Control-F and Control-B, which move over single characters. Like their Control- analogues, ESC F and ESC B move several words if given an argument. ESC F with a negative argument moves backward like ESC B, and ESC B with a negative argument moves forward. Forward motion stops right after the last letter of the word, while backward motion stops right before the first letter.

It is easy to kill a word at a time. ESC D kills the word after point. To be precise, it kills everything from point to the place ESC F would move to. Thus, if point is in the middle of a word, only the part after point is killed. If some punctuation comes after point, and before the next word, it is killed along with the word. If you wish to kill only the next word but not the punctuation, simply do ESC F to get to the end, and kill the word backwards with ESC Rubout. ESC D takes arguments just like ESC F.

ESC Rubout kills the word before point. It kills everything from point back to where ESC B would move to. If point is after the space in "FOO, BAR", then "FOO, " is killed. If you wish to kill just "FOO", then do a ESC B and a ESC D instead of a ESC Rubout.

8.2. Sentence Commands

The JOVE commands for manipulating sentences and paragraphs are mostly ESC commands, so as to resemble the word-handling commands.

ESC A	Move back to the beginning of the sentence.
ESC E	Move forward to the end of the sentence.
ESC K	Kill forward to the end of the sentence.
C-X Rubout	Kill back to the beginning of the sentence.

The commands ESC A and ESC E move to the beginning and end of the current sentence, respectively. They were chosen to resemble Control-A and Control-E, which move to the beginning and end of a line. Unlike them, ESC A and ESC E if repeated or given numeric arguments move over successive sentences. JOVE considers a sentence to end wherever there is a ".", "?", or "!" followed by the end of a line or by one or more spaces. Neither ESC A nor ESC E moves past the end of the line or spaces which delimit the sentence.

Just as C-A and C-E have a kill command, C-K, to go with them, so ESC A and ESC E have a corresponding kill command ESC K which kills from point to the end of the sentence. With minus one as an argument it kills back to the beginning of the sentence. Positive arguments serve as a repeat count.

There is a special command, C-X Rubout for killing back to the beginning of a sentence, because this is useful when you change your mind in the middle of composing text.

8.3. Paragraph Commands

The JOVE commands for handling paragraphs are

ESC [	Move back to previous paragraph beginning.
ESC]	Move forward to next paragraph end.

ESC [moves to the beginning of the current or previous paragraph, while ESC] moves to the end of the current or next paragraph. Paragraphs are delimited by lines of differing indent, or lines with text formatter commands, or blank lines. JOVE knows how to deal with most indented paragraphs correctly, although it can get confused by one- or two-line paragraphs delimited only by indentation.

8.4. Text Indentation Commands

Tab	Indent "appropriately" in a mode-dependent fashion.
LineFeed	Is the same as Return, except it copies the indent of the line you just left.
ESC M	Moves to the line’s first non-blank character.

The way to request indentation is with the Tab command. Its precise effect depends on the major mode. In *Text* mode, it indents to the next tab stop. In *C* mode, it indents to the "right" position for C programs.

To move over the indentation on a line, do ESC M (*first-non-blank*). This command, given anywhere on a line, positions the cursor at the first non-blank, non-tab character on the line.

8.5. Text Filling

Auto Fill mode causes text to be *filled* (broken up into lines that fit in a specified width) automatically as you type it in. If you alter existing text so that it is no longer properly filled, JOVE can fill it again if you ask.

Entering *Auto Fill* mode is done with ESC X *auto-fill-mode*. From then on, lines are broken automatically at spaces when they get longer than the desired width. To leave *Auto Fill* mode, once again execute ESC X *auto-fill-mode*. When *Auto Fill* mode is in effect, the word **Fill** appears in the mode line.

If you edit the middle of a paragraph, it may no longer correctly be filled. To refill a paragraph, use the command ESC J (*fill-paragraph*). It causes the paragraph that point is inside to be filled. All the line breaks are removed and new ones inserted where necessary.

The maximum line width for filling is in the variable *right-margin*. Both ESC J and auto-fill make sure that no line exceeds this width. The value of *right-margin* is initially 72.

Normally ESC J figures out the indent of the paragraph and uses that same indent when filling. If you want to change the indent of a paragraph you set *left-margin* to the new position and type C-U ESC J. *fill-paragraph*, when supplied a numeric argument, uses the value of *left-margin*.

If you know where you want to set the right margin but you don't know the actual value, move to where you want to set the value and use the *right-margin-here* command. *left-margin-here* does the same for the *left-margin* variable.

8.6. Case Conversion Commands

ESC L	Convert following word to lower case.
ESC U	Convert following word to upper case.
ESC C	Capitalize the following word.

The word conversion commands are most useful. ESC L converts the word after point to lower case, moving past it. Thus, successive ESC L's convert successive words. ESC U converts to all capitals instead, while ESC C puts the first letter of the word into upper case and the rest into lower case. All these commands convert several words at once if given an argument. They are especially convenient for converting a large amount of text from all upper case to mixed case, because you can move through the test using ESC L, ESC U or ESC C on each word as appropriate.

When given a negative argument, the word case conversion commands apply to the appropriate number of words before point, but do not move point. This is convenient when you have just typed a word in the wrong case. You can give the case conversion command and continue typing.

If a word case conversion command is given in the middle of a word, it applies only to the part of the word which follows the cursor, treating it as a whole word.

The other case conversion functions are *case-region-upper* and *case-region-lower*, which convert everything between point and mark to the specified case. Point and mark remain unchanged.

8.7. Commands for Fixing Typos

In this section we describe the commands that are especially useful for the times when you catch a mistake on your text after you have made it, or change your mind while composing text on line.

Rubout	Delete last character.
ESC Rubout	Kill last word.
C-X Rubout	Kill to beginning of sentence.
C-T	Transpose two characters.
C-X C-T	Transpose two lines.
ESC Minus ESC L	Convert last word to lower case.
ESC Minus ESC U	Convert last word to upper case.
ESC Minus ESC C	Convert last word to lower case with capital initial.

8.8. Killing Your Mistakes

The Rubout command is the most important correction command. When used among printing (self-inserting) characters, it can be thought of as canceling the last character typed.

When your mistake is longer than a couple of characters, it might be more convenient to use ESC Rubout or C-X Rubout. ESC Rubout kills back to the start of the last word, and C-X Rubout kills back to the start of the last sentence. C-X Rubout is particularly useful when you are thinking of what to write as you type it, in case you change your mind about phrasing. ESC Rubout and C-X Rubout save the killed text for C-Y and ESC Y to retrieve.

ESC Rubout is often useful even when you have typed only a few characters wrong, if you know you are confused in your typing and aren't sure what you typed. At such a time, you cannot correct with Rubout except by looking at the screen to see what you did. It requires less thought to kill the whole word and start over again, especially if the system is heavily loaded.

If you were typing a command or command parameters, C-G will abort the command with no further processing.

8.9. Transposition

The common error of transposing two characters can be fixed with the C-T (*transpose-characters*) command. Normally, C-T transposes the two characters on either side of the cursor and moves the cursor forward one character. Repeating the command several times "drags" a character to the right. (Remember that *point* is considered to be between two characters, even though the visible cursor in your terminal is on only one of them.) When given at the end of a line, rather than switching the last character of the line with the line separator, which would be useless, C-T transposes the last two characters on the line. So, if you catch your transposition error right away, you can fix it with just a C-T. If you don't catch it so fast, you must move the cursor back to between the two characters.

To transpose two lines, use the C-X C-T (*transpose-lines*) command. The line containing the cursor is exchanged with the line above it; the cursor is left at the beginning of the line following its original position.

8.10. Checking and Correcting Spelling

When you write a paper, you should correct its spelling at some point close to finishing it. To correct the entire buffer, do ESC X *spell-buffer*. This invokes the UNIX *spell* program, which prints a list of all the misspelled words. JOVE catches the list and places it in a JOVE buffer called **Spell**. You are given an opportunity to delete from that buffer any words that aren't really errors; then JOVE looks up each misspelled word and remembers where it is in the buffer being corrected. Then you can go forward to each misspelled word with C-X C-N (*next-error*) and backward with C-X C-P (*previous-error*). See the section entitled *Error Message Parsing*.

9. File Handling

The basic unit of stored data is the file. Each program, each paper, lives usually in its own file. To edit a program or paper, the editor must be told the name of the file that contains it. This is called *visiting* a file. To make your changes to the file permanent on disk, you must *save* the file.

9.1. Visiting Files

C-X C-V	Visit a file.
C-X C-R	Same as C-X C-V.
C-X C-S	Save the visited file.
ESC ~	Tell JOVE to forget that the buffer has been changed.

Visiting a file means copying its contents into JOVE where you can edit them. JOVE remembers the name of the file you visited. Unless you use the multiple buffer feature of JOVE, you can only be visiting one file at a time. The name of the current selected buffer is visible in the mode line.

The changes you make with JOVE are made in a copy inside JOVE. The file itself is not changed. The changed text is not permanent until you *save* it in a file. The first time you change the text, an asterisk appears at the end of the mode line; this indicates that the text contains fresh changes which will be lost unless you save them.

To visit a file, use the command C-X C-V. Follow the command with the name of the file you wish to visit, terminated by a Return. You can abort the command by typing C-G, or edit the filename with many of the standard JOVE commands (e.g., C-A, C-E, C-F, ESC F, ESC Rubout). If the filename you wish to visit is similar to the filename in the mode line (the default filename), you can type C-R to insert the default and then edit it. If you do type a Return to finish the command, the new file's text appears on the screen, and its name appears in the mode line. In addition, its name becomes the new default filename.

If you wish to save the file and make your changes permanent, type C-X C-S. After the save is finished, C-X C-S prints the filename and the number of characters and lines that it wrote to the file. If there are no changes to save (no asterisk at the end of the mode line), the file is not saved; otherwise the changes saved and the asterisk at the end of the mode line will disappear.

What if you want to create a file? Just visit it. JOVE prints *(New file)* but aside from that behaves as if you had visited an existing empty file. If you make any changes and save them, the file is created. If you visit a nonexistent file unintentionally (because you typed the wrong filename), go ahead and visit the file you meant. If you don't save the unwanted file, it is not created.

If you alter one file and then visit another in the same buffer, JOVE offers to save the old one. If you answer YES, the old file is saved; if you answer NO, all the changes you have made to it since the last save are lost. You should not type ahead after a file visiting command, because your type-ahead might answer an unexpected question in a way that you would regret.

Sometimes you will change a buffer by accident. Even if you undo the effect of the change by editing, JOVE still knows that "the buffer has been changed". You can tell JOVE to pretend that there have been no changes with the ESC ˜ command (*make-buffer-unmodified*). This command simply clears the "modified" flag which says that the buffer contains changes which need to be saved. Even if the buffer really *is* changed JOVE will still act as if it were not.

If JOVE is about to save a file and sees that the date of the version on disk does not match what JOVE last read or wrote, JOVE notifies you of this fact, and asks what to do, because this probably means that something is wrong. For example, somebody else may have been editing the same file. If this is so, there is a good chance that your work or his work will be lost if you don't take the proper steps. You should first find out exactly what is going on. If you determine that somebody else has modified the file, save your file under a different filename and then DIFF the two files to merge the two sets of changes. (The "patch" command is useful for applying the results of context diffs directly). Also get in touch with the other person so that the files don't diverge any further.

9.2. How to Undo Drastic Changes to a File

If you have made several extensive changes to a file and then change your mind about them, and you haven't yet saved them, you can get rid of them by reading in the previous version of the file. You can do this with the C-X C-V command, to visit the unsaved version of the file.

9.3. Recovering from system/editor crashes

JOVE does not have *Auto Save* mode, but it does provide a way to recover your work in the event of a system or editor crash. JOVE saves information about the files you're editing every so many changes to a buffer to make recovery possible. Since a relatively small amount of information is involved it's hardly even noticeable when JOVE does this. The variable "sync-frequency" says how often to save the necessary information, and the default is every 50 changes. 50 is a very reasonable number: if you are writing a paper you will not lose more than the last 50 characters you typed, which is less than the average length of a line.

9.4. Miscellaneous File Operations

ESC X *write-file* <file><return> writes the contents of the buffer into the file <file>, and then visits that file. It can be thought of as a way of "changing the name" of the file you are visiting. Unlike C-X C-S, *write-file* saves even if the buffer has not been changed. C-X C-W is another way of getting this command.

ESC X *insert-file* <file><return> inserts the contents of <file> into the buffer at point, leaving point unchanged before the contents. You can also use C-X C-I to get this command.

ESC X *write-region* <file><return> writes the region (the text between point and mark) to the specified file. It does not set the visited filename. The buffer is not changed.

ESC X *append-region* <file><return> appends the region to <file>. The text is added to the end of <file>.

10. Using Multiple Buffers

When we speak of "the buffer", which contains the text you are editing, we have given the impression that there is only one. In fact, there may be many of them, each with its own body of text. At any time only one buffer can be *selected* and available for editing, but it isn't hard to switch to a different one. Each buffer individually remembers which file it is visiting, what modes are in effect, and whether there are any changes that need saving.

C-X B	Select or create a buffer.
C-X C-F	Visit a file in its own buffer.
C-X C-B	List the existing buffers.

C-X K Kill a buffer.

Each buffer in JOVE has a single name, which normally doesn't change. A buffer's name can be any length. The name of the currently selected buffer and the name of the file visited in it are visible in the mode line when you are at top level. A newly started JOVE has only one buffer, named **Main**, unless you specified files to edit in the shell command that started JOVE.

10.1. Creating and Selecting Buffers

To create a new buffer, you need only think of a name for it (say, FOO) and then do C-X B FOO<return>, which is the command C-X B (*select-buffer*) followed by the name. This makes a new, empty buffer (if one by that name didn't previously exist) and selects it for editing. The new buffer is not visiting any file, so if you try to save it you will be asked for the filename to use. Each buffer has its own major mode; the new buffer's major mode is *Text* mode by default.

To return to buffer FOO later after having switched to another, the same command C-X B FOO<return> is used, since C-X B can tell whether a buffer named FOO exists already or not. C-X B Main<return> reselects the buffer Main that JOVE started out with. Just C-X B<return> reselects the previous buffer. Repeated C-X B<return>'s alternate between the last two buffers selected.

You can also read a file into its own newly created buffer, all with one command: C-X C-F (*find-file*), followed by the filename. The name of the buffer is the last element of the file's pathname. C-F stands for "Find", because if the specified file already resides in a buffer in your JOVE, that buffer is reselected. So you need not remember whether you have brought the file in already or not. A buffer created by C-X C-F can be reselected later with C-X B or C-X C-F, whichever you find more convenient. Nonexistent files can be created with C-X C-F just as they can with C-X C-V.

10.2. Using Existing Buffers

To get a list of all the buffers that exist, do C-X C-B (*list-buffers*). Each buffer's type, name, and visited filename is printed. An asterisk before the buffer name indicates a buffer which contains changes that have not been saved. The number that appears at the beginning of a line in a C-X C-B listing is that buffer's *buffer number*. You can select a buffer by typing its number in place of its name. If a buffer with that number doesn't already exist, a new buffer is created with that number as its name.

If several buffers have modified text in them, you should save some of them with C-X C-M (*write-modified-files*). This finds all the buffers that need saving and then saves them. Saving the buffers this way is much easier and more efficient (but more dangerous) than selecting each one and typing C-X C-S. If you give C-X C-M an argument, JOVE will ask for confirmation before saving each buffer.

ESC X *rename-buffer* <new name><return> changes the name of the currently selected buffer.

ESC X *erase-buffer* <buffer name><return> erases the contents of the <buffer name> without deleting the buffer entirely.

10.3. Killing Buffers

After you use a JOVE for a while, it may fill up with buffers which you no longer need. Eventually you can reach a point where trying to create any more results in an "out of memory" or "out of lines" error. When this happens you will want to kill some buffers with the C-X K (*delete-buffer*) command. You can kill the buffer FOO by doing C-X K FOO<return>. If you type C-X K <return> JOVE will kill the previously selected buffer. If you try to kill a buffer that needs saving JOVE will ask you to confirm it.

If you need to kill several buffers, use the command *kill-some-buffers*. This prompts you with the name of each buffer and asks for confirmation before killing that buffer.

11. Controlling the Display

Since only part of a large file will fit on the screen, JOVE tries to show the part that is likely to be interesting. The display control commands allow you to see a different part of the file.

C-L Reposition point at a specified vertical position, OR clear and redraw the screen with point in the same place.

C-V	Scroll forwards (a screen or a few lines).
ESC V	Scroll backwards.
C-Z	Scroll forward some lines.
ESC Z	Scroll backwards some lines.

The terminal screen is rarely large enough to display all of your file. If the whole buffer doesn't fit on the screen, JOVE shows a contiguous portion of it, containing *point.* It continues to show approximately the same portion until point moves outside of what is displayed; then JOVE chooses a new portion centered around the new *point.* This is JOVE's guess as to what you are most interested in seeing, but if the guess is wrong, you can use the display control commands to see a different portion. The available screen area through which you can see part of the buffer is called *the window*, and the choice of where in the buffer to start displaying is also called *the window.* (When there is only one window, it plus the mode line and the input line take up the whole screen).

First we describe how JOVE chooses a new window position on its own. The goal is usually to place *point* half way down the window. This is controlled by the variable *scroll-step*, whose value is the number of lines above the bottom or below the top of the window that the line containing point is placed. A value of 0 (the initial value) means center *point* in the window.

The basic display control command is C-L (*redraw-display*). In its simplest form, with no argument, it tells JOVE to choose a new window position, centering point half way from the top as usual.

C-L with a positive argument chooses a new window so as to put point that many lines from the top. An argument of zero puts point on the very top line. Point does not move with respect to the text; rather, the text and point move rigidly on the screen.

If point stays on the same line, the window is first cleared and then redrawn. Thus, two C-L's in a row are guaranteed to clear the current window. ESC C-L will clear and redraw the entire screen.

The *scrolling* commands C-V, ESC V, C-Z, and ESC Z, let you move the whole display up or down a few lines. C-V (*next-page*) with an argument shows you that many more lines at the bottom of the screen, moving the text and point up together as C-L might. C-V with a negative argument shows you more lines at the top of the screen, as does ESC V (*previous-page*) with a positive argument.

To read the buffer a window at a time, use the C-V command with no argument. It takes the last line at the bottom of the window and puts it at the top, followed by nearly a whole window of lines not visible before. Point is put at the top of the window. Thus, each C-V shows the "next page of text", except for one line of overlap to provide context. To move backward, use ESC V without an argument, which moves a whole window backwards (again with a line of overlap).

C-Z and ESC Z scroll one line forward and one line backward, respectively. These are convenient for moving in units of lines without having to type a numeric argument.

11.1. Multiple Windows

JOVE allows you to split the screen into two or more *windows* and use them to display parts of different files, or different parts of the same file.

C-X 2	Divide the current window into two smaller ones.
C-X 1	Delete all windows but the current one.
C-X D	Delete current window.
C-X N	Switch to the next window.
C-X P	Switch to the previous window.
C-X O	Same as C-X P.
C-X ^	Make this window bigger.
ESC C-V	Scroll the other window.

When using *multiple window* mode, the text portion of the screen is divided into separate parts called *windows*, which can display different pieces of text. Each window can display different files, or parts of the same file. Only one of the windows is *active;* that is the window which the cursor is in. Editing normally takes place in that window alone. To edit in another window, you would give a command to move the cursor to the other window, and then edit there.

Each window displays a mode line for the buffer it's displaying. This is useful to keep track of which window corresponds with which file. In addition, the mode line serves as a separator between windows. By setting the variable *mode-line-should-standout* to "on" you can have JOVE display the mode-line in reverse video (assuming your particular terminal has the reverse video capability).

The command C-X 2 (*split-current-window*) enters multiple window mode. A new mode line appears across the middle of the screen, dividing the text display area into two halves. Both windows contain the same buffer and display the same position in it, namely where point was at the time you issued the command. The cursor moves to the second window.

To return to viewing only one window, use the command C-X 1 (*delete-other-windows*). The current window expands to fill the whole screen, and the other windows disappear until the next C-X 2. (The buffers and their contents are unaffected by any of the window operations).

While there is more than one window, you can use C-X N (*next-window*) to switch to the next window, and C-X P (*previous-window*) to switch to the previous one. If you are in the bottom window and you type C-X N, you will be placed in the top window, and the same kind of thing happens when you type C-X P in the top window, namely you will be placed in the bottom window. C-X O is the same as C-X P. It stands for "other window" because when there are only two windows, repeated use of this command will switch between the two windows.

Often you will be editing one window while using the other just for reference. Then, the command ESC C-V (*page-next-window*) is very useful. It scrolls the next window, as if you switched to the next window, typed C-V, and switched back, without your having to do all that. With a negative argument, ESC C-V will do an ESC V in the next window.

When a window splits, both halves are approximately the same size. You can redistribute the screen space between the windows with the C-X ˆ (*grow-window*) command. It makes the currently selected window grow one line bigger, or as many lines as is specified with a numeric argument. Use ESC X *shrink-window* to make the current window smaller.

11.2. Multiple Windows and Multiple Buffers

Buffers can be selected independently in each window. The C-X B command selects a new buffer in whichever window contains the cursor. Other windows' buffers do not change.

You can view the same buffer in more than one window. Although the same buffer appears in both windows, they have different values of point, so you can move around in one window while the other window continues to show the same text. Then, having found one place you wish to refer to, you can go back into the other window with C-X O or C-X P to make your changes.

If you have the same buffer in both windows, you must beware of trying to visit a different file in one of the windows with C-X C-V, because if you bring a new file into this buffer, it will replaced the old file in *both* windows. To view different files in different windows, you must switch buffers in one of the windows first (with C-X B or C-X C-F, perhaps).

A convenient "combination" command for viewing something in another window is C-X 4 (*window-find*). With this command you can ask to see any specified buffer, file or tag in the other window. Follow the C-X 4 with either B and a buffer name, F and a filename, or T and a tag name. This switches to the other window and finds there what you specified. If you were previously in one-window mode, multiple-window mode is entered. C-X 4 B is similar to C-X 2 C-X B. C-X 4 F is similar to C-X 2 C-X C-F. C-X 4 T is similar to C-X 2 C-X T. The difference is one of efficiency, and also that C-X 4 works equally well if you are already using two windows.

12. Processes Under JOVE

Another feature in JOVE is its ability to interact with UNIX in a useful way. You can run other UNIX commands from JOVE and catch their output in JOVE buffers. In this chapter we will discuss the different ways to run and interact with UNIX commands.

12.1. Non-interactive UNIX commands

To run a UNIX command from JOVE just type "C-X !" followed by the name of the command terminated with Return. For example, to get a list of all the users on the system, you do:

C-X ! who<return>

Then JOVE picks a reasonable buffer in which the output from the command will be placed. E.g., "who" uses a buffer called **who**; "ps alx" uses **ps**; and "fgrep -n foo *.c" uses **fgrep**. If JOVE wants to use a buffer that already exists it first erases the old contents. If the buffer it selects holds a file, not output from a previous shell command, you must first delete that buffer with C-X K.

Once JOVE has picked a buffer it puts that buffer in a window so you can see the command's output as it is running. If there is only one window JOVE will automatically make another one. Otherwise, JOVE tries to pick the most convenient window which isn't the current one.

It's not a good idea to type anything while the command is running. There are two reasons for this:

(i) JOVE won't see the characters (thus won't execute them) until the command finishes, so you may forget what you've typed.

(ii) Although JOVE won't know what you've typed, it *will* know that you've typed something, and then it will try to be "smart" and not update the display until it's interpreted what you've typed. But, of course, JOVE won't interpret what you type until the UNIX command completes, so you're left with the uneasy feeling you get when you don't know what the hell the computer is doing*.

If you want to interrupt the command for some reason (perhaps you mistyped it, or you changed your mind) you can type C-]. Typing this inside JOVE while a process is running is the same as typing C-C when you are outside JOVE, namely the process stops in a hurry.

When the command finishes, JOVE puts you back in the window in which you started. Then it prints a message indicating whether or not the command completed successfully in its (the command's) opinion. That is, if the command had what it considers an error (or you interrupt it with C-]) JOVE will print an appropriate message.

12.2. Limitations of Non-Interactive Processes

The reason these are called non-interactive processes is that you can't type any input to them; you can't interact with them; they can't ask you questions because there is no way for you to answer. For example, you can't run a command interpreter (a shell), or *mail* or *crypt* with C-X ! because there is no way to provide it with input. Remember that JOVE (not the process in the window) is listening to your keyboard, and JOVE waits until the process dies before it looks at what you type.

C-X ! is useful for running commands that do some output and then exit. For example, it's very useful to use with the C compiler to catch compilation error messages (see Compiling C Programs), or with the *grep* commands.

12.3. Interactive Processes — Run a Shell in a Window

Some versions of JOVE† have the capability of running interactive processes. This is more useful than non-interactive processes for certain types of jobs:

(i) You can go off and do some editing while the command is running. This is useful for commands that do sporadic output and run for fairly long periods of time.

(ii) Unlike non-interactive processes, you can type input to these. In addition, you can edit what you type with the power of all the JOVE commands *before* you send the input to the process. This is a really important feature, and is especially useful for running a shell in a window.

(iii) Because you can continue with normal editing while one of the processes is running, you can create a bunch of contexts and manage them (select them, delete them, or temporarily put them aside) with JOVE's window and buffer mechanisms.

---

*This is a bug and should be fixed, but probably won't be for a while.
† For example, the version provided with 4.3BSD.

Although we may have given an image of processes being attached to *windows,* in fact they are attached to *buffers.* Therefore, once an *i-process* is running you can select another buffer into that window, or if you wish you can delete the window altogether. If you reselect that buffer later it will be up to date. That is, even though the buffer wasn't visible it was still receiving output from the process. You don't have to worry about missing anything when the buffer isn't visible.

12.4. Advantages of Running Processes in JOVE Windows.

There are several advantages to running a shell in a window. What you type isn't seen immediately by the process; instead JOVE waits until you type an entire line before passing it on to the process to read. This means that before you type <return> all of JOVE's editing capabilities are available for fixing errors on your input line. If you discover an error at the beginning of the line, rather than erasing the whole line and starting over, you can simply move to the error, correct it, move back and continue typing.

Another feature is that you have the entire history of your session in a JOVE buffer. You don't have to worry about output from a command moving past the top of the screen. If you missed some output you can move back through it with ESC V and other commands. In addition, you can save yourself retyping a command (or a similar one) by sending edited versions of previous commands, or edit the output of one command to become a list of commands to be executed ("immediate shell scripts").

12.5. Differences between Normal and I-process Buffers

JOVE behaves differently in several ways when you are in an *i-process* buffer. Most obviously, <return> does different things depending on both your position in the buffer and on the state of the process. In the normal case, when point is at the end of the buffer, Return does what you'd expect: it inserts a line-separator and then sends the line to the process. If you are somewhere else in the buffer, possibly positioned at a previous command that you want to edit, Return will place a copy of that line (with the prompt discarded if there is one) at the end of the buffer and move you there. Then you can edit the line and type Return as in the normal case. If the process has died for some reason, Return does nothing. It doesn't even insert itself. If that happens unexpectedly, you should type ESC X *list-processes*<return> to get a list of each process and its state. If your process died abnormally, *list-processes* may help you figure out why.

12.6. How to Run a Shell in a Window

Type ESC X *i-shell*<return> to start up a shell. As with C-X !, JOVE will create a buffer, called **shell–1**, and select a window for this new buffer. But unlike C-X ! you will be left in the new window. Now, the shell process is said to be attached to **shell–1**, and it is considered an *i-process* buffer.

13. Directory Handling

To save having to use absolute pathnames when you want to edit a nearby file JOVE allows you to move around the UNIX filesystem just as the c-shell does. These commands are:

cd dir	Change to the specified directory.
pushd [dir]	Like *cd*, but save the old directory on the directory stack. With no directory argument, simply exchange the top two directories on the stack and *cd* to the new top.
popd	Take the current directory off the stack and *cd* to the directory now at the top.
dirs	Display the contents of the directory stack.

The names and behavior of these commands were chosen to mimic those in the c-shell.

14. Editing C Programs

This section details the support provided by JOVE for working on C programs.

14.1. Indentation Commands

To save having to lay out C programs "by hand", JOVE has an idea of the correct indentation of a line, based on the surrounding context. When you are in C Mode, JOVE treats tabs specially — typing a tab at the beginning of a new line means "indent to the right place". Closing braces are also handled specially, and are indented to match the corresponding open brace.

14.2. Parenthesis and Brace Matching

To check that parentheses and braces match the way you think they do, turn on *Show Match* mode (ESC X show-match-mode). Then, whenever you type a close brace or parenthesis, the cursor moves momentarily to the matching opener, if it's currently visible. If it's not visible, JOVE displays the line containing the matching opener on the message line.

14.3. C Tags

Often when you are editing a C program, especially someone else's code, you see a function call and wonder what that function does. You then search for the function within the current file and if you're lucky find the definition, finally returning to the original spot when you are done. However, if are unlucky, the function turns out to be external (defined in another file) and you have to suspend the edit, *grep* for the function name in every .c that might contain it, and finally visit the appropriate file.

To avoid this diversion or the need to remember which function is defined in which file, Berkeley UNIX has a program called *ctags(1)*, which takes a set of source files and looks for function definitions, producing a file called *tags* as its output.

JOVE has a command called C-X T (*find-tag*) that prompts you for the name of a function (a *tag*), looks up the tag reference in the previously constructed tags file, then visits the file containing that tag in a new buffer, with point positioned at the definition of the function. There is another version of this command, namely *find-tag-at-point*, that uses the identifier at *point.*

So, when you've added new functions to a module, or moved some old ones around, run the *ctags* program to regenerate the *tags* file. JOVE looks in the file specified in the *tag-file* variable. The default is "./tags", that is, the tag file in the current directory. If you wish to use an alternate tag file, you use C-U C-X T, and JOVE will prompt for a file name. If you find yourself specifying the same file again and again, you can set *tag-file* to that file, and run *find-tag* with no numeric argument.

To begin an editing session looking for a particular tag, use the *−t tag* command line option to JOVE. For example, say you wanted to look at the file containing the tag *SkipChar*, you would invoke JOVE as:

% jove −t SkipChar

14.4. Compiling Your Program

You've typed in a program or altered an existing one and now you want to run it through the compiler to check for errors. To save having to suspend the edit, run the compiler, scribble down error messages, and then resume the edit, JOVE allows you to compile your code while in the editor. This is done with the C-X C-E (*compile-it*) command. If you run *compile-it* with no argument it runs the UNIX *make* program into a buffer; If you need a special command or want to pass arguments to *make*, run *compile-it* with any argument (C-U is good enough) and you will be prompted for the command to execute.

If any error messages are produced, they are treated specially by JOVE. That treatment is the subject of the next section.

14.5. Error Message Parsing and Spelling Checking

JOVE knows how to interpret the error messages from many UNIX commands; In particular, the messages from *cc*, *grep* and *lint* can be understood. After running the *compile-it* command, the *parse-errors* command is automatically executed, and any errors found are displayed in a new buffer. The files whose names are found in parsing the error messages are each brought into JOVE buffers and the point is positioned at the first error in the first file. The commands *current-error*, C-X C-N (*next-error*), and C-X C-P (*previous-error*) can be used to traverse the list of errors.

If you already have a file called *errs* containing, say, c compiler messages then you can get JOVE to interpret the messages by invoking it as:

% jove −p errs

JOVE has a special mechanism for checking the the spelling of a document; It runs the UNIX spell program into a buffer. You then delete from this buffer all those words that are not spelling errors and then JOVE runs the *parse-spelling-errors* command to yield a list of errors just as in the last section.

15. Simple Customization

15.1. Major Modes

To help with editing particular types of file, say a paper or a C program, JOVE has several *major modes*. These are as follows:

15.1.1. Text mode

This is the default major mode. Nothing special is done.

15.1.2. C mode

This mode affects the behavior of the tab and parentheses characters. Instead of just inserting the tab, JOVE determines where the text "ought" to line up for the C language and tabs to that position instead. The same thing happens with the close brace and close parenthesis; they are tabbed to the "right" place and then inserted. Using the *auto-execute-command* command, you can make JOVE enter *C Mode* whenever you edit a file whose name ends in *.c*.

15.1.3. Lisp mode

This mode is analogous to *C Mode*, but performs the indentation needed to lay out Lisp programs properly. Note also the *grind-s-expr* command that prettyprints an *s-expression* and the *kill-mode-expression* command.

15.2. Minor Modes

In addition to the major modes, JOVE has a set of minor modes. These are as follows:

15.2.1. Auto Indent

In this mode, JOVE indents each line the same way as that above it. That is, the Return key in this mode acts as the Linefeed key ordinarily does.

15.2.2. Show Match

Move the cursor momentarily to the matching opening parenthesis when a closing parenthesis is typed.

15.2.3. Auto Fill

In *Auto Fill* mode, a newline is automatically inserted when the line length exceeds the right margin. This way, you can type a whole paper without having to use the Return key.

15.2.4. Over Write

In this mode, any text typed in will replace the previous contents. (The default is for new text to be inserted and "push" the old along.) This is useful for editing an already-formatted diagram in which you want to change some things without moving other things around on the screen.

15.2.5. Word Abbrev

In this mode, every word you type is compared to a list of word abbreviations; whenever you type an abbreviation, it is replaced by the text that it abbreviates. This can save typing if a particular word or phrase must be entered many times. The abbreviations and their expansions are held in a file that looks like:

```
abbrev:phrase
```

This file can be set up in your *~/.joverc* with the *read-word-abbrev-file* command. Then, whenever you are editing a buffer in *Word Abbrev* mode, JOVE checks for the abbreviations you've given. See also the commands *read-word-abbrev-file*, *write-word-abbrev-file*, *edit-word-abbrevs*, *define-global-word-abbrev*, *define-mode-word-abbrev*, and *bind-macro-to-word-abbrev*, and the variable *auto-case-abbrev*.

15.3. Variables

JOVE can be tailored to suit your needs by changing the values of variables. A JOVE variable can be given a value with the *set* command, and its value displayed with the *print* command.

The variables JOVE understands are listed along with the commands in the alphabetical list at the end of this document.

15.4. Key Re-binding

Many of the commands built into JOVE are not bound to specific keys. The command handler in JOVE is used to invoke these commands and is activated by the *execute-extended-command* command (ESC X). When the name of a command typed in is unambiguous, that command will be executed. Since it is very slow to have to type in the name of each command every time it is needed, JOVE makes it possible to *bind* commands to keys. When a command is *bound* to a key any future hits on that key will invoke that command. All the printing characters are initially bound to the command *self-insert*. Thus, typing any printing character causes it to be inserted into the text. Any of the existing commands can be bound to any key. (A *key* may actually be a *control character* or an *escape sequence* as explained previously under *Command Input Conventions*).

Since there are more commands than there are keys, two keys are treated as *prefix* commands. When a key bound to one of the prefix commands is typed, the next character typed is interpreted on the basis that it was preceded by one of the prefix keys. Initially ^X and ESC are the prefix keys and many of the built in commands are initially bound to these "two stroke" keys. (For historical reasons, the Escape key is often referred to as "Meta").

15.5. Keyboard Macros

Although JOVE has many powerful commands, you often find that you have a task that no individual command can do. JOVE allows you to define your own commands from sequences of existing ones "by example"; Such a sequence is termed a *macro*. The procedure is as follows: First you type the *start-remembering* command, usually bound to C-X (. Next you "perform" the commands which as they are being executed are also remembered, which will constitute the body of the macro. Then you give the *stop-remembering* command, usually bound to C-X). You now have a *keyboard macro*. To run this command sequence again, use the command *execute-keyboard-macro*, usually bound to C-X E. You may find this bothersome to type and re-type, so there is a way to bind the macro to a key. First, you must give the keyboard macro a name using the *name-keyboard-macro* command. Then the binding is made with the *bind-macro-to-key* command. We're still not finished because all this hard work will be lost if you leave JOVE. What you do is to save your macros into a file with the *write-macros-to-file* command. There is a corresponding *read-macros-from-file* command to retrieve your macros in the next editing session.

15.6. Initialization Files

Users will likely want to modify the default key bindings to their liking. Since it would be quite annoying to have to set up the bindings each time JOVE is started up, JOVE has the ability to read in a "startup" file. Whenever JOVE is started, it reads commands from the file *.joverc* in the user's home directory. These commands are read as if they were typed to the command handler (ESC X) during an edit. There can be only one command per line in the startup file. If there is a file */usr/lib/jove/joverc*, then this file will be read before the user's *.joverc* file. This can be used to set up a system-wide default startup mode for JOVE that is tailored to the needs of that system.

The *source* command can be used to read commands from a specified file at any time during an editing session, even from inside the *.joverc* file. This means that a macro can be used to change the key bindings, e.g., to enter a mode, by reading from a specified file which contains all the necessary bindings.

16. Alphabetical List of Commands and Variables

16.1. Prefix-1 (Escape)

This reads the next character and runs a command based on the character typed. If you wait for more than a second or so before typing the next character, the message "ESC" will be printed on the message line to remind you that JOVE is waiting for another character.

16.2. Prefix-2 (C-X)

This reads the next character and runs a command based on the character typed. If you wait for more than a second or so before typing another character, the message "C-X" will be printed on the message line to remind you that JOVE is waiting for another character.

16.3. Prefix-3 (Not Bound)

This reads the next character and runs a command based on the character typed. If you wait for more than a second or so before typing the next character, the character that invoked Prefix-3 will be printed on the message line to remind you that JOVE is waiting for another one.

16.4. allow-ˆS-and-ˆQ (variable)

This variable, when set, tells JOVE that your terminal does not need to use the characters C-S and C-Q for flow control, and that it is okay to bind things to them. This variable should be set depending upon what kind of terminal you have.

16.5. allow-bad-filenames (variable)

If set, this variable permits filenames to contain "bad" characters such as those from the set *&%!"‘[]{}. These files are harder to deal with, because the characters mean something to the shell. The default value is "off".

16.6. append-region (Not Bound)

This appends the region to a specified file. If the file does not already exist it is created.

16.7. apropos (Not Bound)

This types out all the commands, variables and macros with the specific keyword in their names. For each command and macro that contains the string, the key sequence that can be used to execute the command or macro is printed; with variables, the current value is printed. So, to find all the commands that are related to windows, you type

    ESC X apropos window<Return>

16.8. auto-case-abbrev (variable)

When this variable is on (the default), word abbreviations are adjusted for case automatically. For example, if "jove" were the abbreviation for "jonathan’s own version of emacs", then typing "jove" would give you "jonathan’s own version of emacs", typing "Jove" would give you "Jonathan’s own version of emacs", and typing "JOVE" would give you "Jonathan’s Own Version of Emacs". When this variable is "off", upper and lower case are distinguished when looking for the abbreviation, i.e., in the example above, "JOVE" and "Jove" would not be expanded unless they were defined separately.

16.9. auto-execute-command (Not Bound)

This tells JOVE to execute a command automatically when a file whose name matches a specified pattern is visited. The first argument is the command you want executed and the second is a regular expression pattern that specifies the files that apply. For example, if you want to be in show-match-mode when you edit C source files (that is, files that end with ".c" or ".h") you can type

    ESC X auto-execute-command show-match-mode .*.[ch]$

16.10. auto-execute-macro (Not Bound)

This is like *auto-execute-command* except you use it to execute macros automatically instead of built-in commands.

16.11. auto-fill-mode (Not Bound)

This turns on Auto Fill mode (or off if it's currently on) in the selected buffer. When JOVE is in Auto Fill mode it automatically breaks lines for you when you reach the right margin so you don't have to remember to hit Return. JOVE uses 78 as the right margin but you can change that by setting the variable *right-margin* to another value. See the *set* command to learn how to do this.

16.12. auto-indent-mode (Not Bound)

This turns on Auto Indent mode (or off if it's currently on) in the selected buffer. When JOVE is in Auto Indent mode, Return indents the new line to the same position as the line you were just on. This is useful for lining up C code (or any other language (but what else is there besides C?)). This is out of date because of the new command called *newline-and-indent* but it remains because of several "requests" on the part of, uh, enthusiastic and excitable users, that it be left as it is.

16.13. backward-character (C-B)

This moves point backward over a single character. If point is at the beginning of the line it moves to the end of the previous line.

16.14. backward-paragraph (ESC [)

This moves point backward to the beginning of the current or previous paragraph. Paragraphs are bounded by lines that begin with a Period or Tab, or by blank lines; a change in indentation may also signal a break between paragraphs, except that JOVE allows the first line of a paragraph to be indented differently from the other lines.

16.15. backward-s-expression (ESC C-B)

This moves point backward over a s-expression. It is just like *forward-s-expression* with a negative argument.

16.16. backward-sentence (ESC A)

This moves point backward to the beginning of the current or previous sentence. JOVE considers the end of a sentence to be the characters ".", "!" or "?" followed by a Return or by one or more spaces.

16.17. backward-word (ESC B)

This moves point backward to the beginning of the current or previous word.

16.18. bad-filename-extensions (variable)

This contains a list of words separated by spaces which are to be considered bad filename extensions, and so will not be counted in filename completion. The default is ".o" so if you have jove.c and jove.o in the same directory, the filename completion will *not* complain of an ambiguity because it will ignore jove.o.

16.19. beginning-of-file (ESC <)

This moves point backward to the beginning of the buffer. This sometimes prints the "Point Pushed" message. If the top of the buffer isn't on the screen JOVE will set the mark so you can go back to where you were if you want.

16.20. beginning-of-line (C-A)

This moves point to the beginning of the current line.

16.21. beginning-of-window (ESC ,)

This moves point to the beginning of the current window. The sequence "ESC ," is the same as "ESC <" (beginning of file) except without the shift key on the "<", and can thus can easily be remembered.

16.22. bind-to-key (Not Bound)

This attaches a key to an internal JOVE command so that future hits on that key invoke that command. For example, to make "C-W" erase the previous word, you type "ESC X bind-to-key kill-previous-word C-W".

16.23. bind-macro-to-key (Not Bound)

This is like *bind-to-key* except you use it to attach keys to named macros.

16.24. bind-macro-to-word-abbrev (Not Bound)

This command allows you to bind a macro to a previously defined word abbreviation. Whenever you type the abbreviation, it will first be expanded as an abbreviation, and then the macro will be executed. Note that if the macro moves around, you should set the mark first (C-@) and then exchange the point and mark last (C-X C-X).

16.25. buffer-position (Not Bound)

This displays the current file name, current line number, total number of lines, percentage of the way through the file, and the position of the cursor in the current line.

16.26. c-mode (Not Bound)

This turns on C mode in the currently selected buffer. This is one of currently four possible major modes: Fundamental, Text, C, Lisp. When in C or Lisp mode, Tab, "}", and ")" behave a little differently from usual: They are indented to the "right" place for C (or Lisp) programs. In JOVE, the "right" place is simply the way the author likes it (but I've got good taste).

16.27. case-character-capitalize (Not Bound)

This capitalizes the character after point, i.e., the character undo the cursor. If a negative argument is supplied that many characters *before* point are upper cased.

16.28. case-ignore-search (variable)

This variable, when set, tells JOVE to treat upper and lower case as the same when searching. Thus "jove" and "JOVE" would match, and "JoVe" would match either. The default value of this variable is "off".

16.29. case-region-lower (Not Bound)

This changes all the upper case letters in the region to their lower case equivalent.

16.30. case-region-upper (Not Bound)

This changes all the lower case letters in the region to their upper case equivalent.

16.31. case-word-capitalize (ESC C)

This capitalizes the current word by making the current letter upper case and making the rest of the word lower case. Point is moved to the end of the word. If point is not positioned on a word it is first moved forward to the beginning of the next word. If a negative argument is supplied that many words *before* point are capitalized. This is useful for correcting the word just typed without having to move point to the beginning of the word yourself.

16.32. case-word-lower (ESC L)

This lower-cases the current word and leaves point at the end of it. If point is in the middle of a word the rest of the word is converted. If point is not in a word it is first moved forward to the beginning of the next word. If a negative argument is supplied that many words *before* point are converted to lower case. This is useful for correcting the word just typed without having to move point to the beginning of the word yourself.

16.33. case-word-upper (ESC U)

This upper-cases the current word and leaves point at the end of it. If point is in the middle of a word the rest of the word is converted. If point is not in a word it is first moved forward to the beginning of the next word. If a negative argument is supplied that many words *before* point are converted to upper case. This is useful for correcting the word just typed without having to move point to the beginning of the word yourself.

16.34. character-to-octal-insert (Not Bound)

This inserts a Back-slash followed by the ascii value of the next character typed. For example, "C-G" inserts the string "\007".

16.35. cd (Not Bound)

This changes the current directory.

16.36. clear-and-redraw (ESC C-L)

This clears the entire screen and redraws all the windows. Use this when JOVE gets confused about what's on the screen, or when the screen gets filled with garbage characters or output from another program.

16.37. comment-format (variable)

This variable tells JOVE how to format your comments when you run the command *fill-comment.* Its format is this:

<open pattern>%!<line header>%c<line trailer>%!<close pattern>

The %!, %c, and %! must appear in the format; everything else is optional. A newline (represented by %n) may appear in the open or close patterns. %% is the representation for %. The default comment format is for C comments. See *fill-comment* for more.

16.38. compile-it (C-X C-E)

This compiles your program by running the UNIX command "make" into a buffer, and automatically parsing the error messages that are created (if any). See the *parse-errors* and *parse-special-errors* commands. To compile a C program without "make", use "C-U C-X C-E" and JOVE will prompt for a command to run instead of make. (And then the command you type will become the default command.) You can use this to parse the output from the C compiler or the "grep" or "lint" programs.

16.39. continue-process (Not Bound)

This sends SIGCONT to the current interactive process, *if* the process is currently stopped.

16.40. copy-region (ESC W)

This takes all the text in the region and copies it onto the kill ring buffer. This is just like running *kill-region* followed by the *yank* command. See the *kill-region* and *yank* commands.

16.41. current-error (Not Bound)

This moves to the current error in the list of parsed errors. See the *next-error* and *previous-error* commands for more detailed information.

16.42. date (Not Bound)

This prints the date on the message line.

16.43. define-mode-word-abbrev (Not Bound)

This defines a mode-specific abbreviation.

16.44. define-global-word-abbrev (Not Bound)

This defines a global abbreviation.

16.45. delete-blank-lines (C-X C-O)

This deletes all the blank lines around point. This is useful when you previously opened many lines with "C-O" and now wish to delete the unused ones.

16.46. delete-buffer (C-X K)

This deletes a buffer and frees up all the memory associated with it. Be careful! Once a buffer has been deleted it is gone forever. JOVE will ask you to confirm if you try to delete a buffer that needs saving. This command is useful for when JOVE runs out of space to store new buffers.

16.47. delete-macro (Not Bound)

This deletes a macro from the list of named macros. It is an error to delete the keyboard-macro. Once the macro is deleted it is gone forever. If you are about to save macros to a file and decide you don't want to save a particular one, delete it.

16.48. delete-next-character (C-D)

This deletes the character that's just after point (that is, the character under the cursor). If point is at the end of a line, the line separator is deleted and the next line is joined with the current one.

16.49. delete-other-windows (C-X 1)

This deletes all the other windows except the current one. This can be thought of as going back into One Window mode.

16.50. delete-previous-character (Rubout)

This deletes the character that's just before point (that is, the character before the cursor). If point is at the beginning of the line, the line separator is deleted and that line is joined with the previous one.

16.51. delete-white-space (ESC \)

This deletes all the Tabs and Spaces around point.

16.52. delete-current-window (C-X D)

This deletes the current window and moves point into one of the remaining ones. It is an error to try to delete the only remaining window.

16.53. describe-bindings (Not Bound)

This types out a list containing each bound key and the command that gets invoked every time that key is typed. To make a wall chart of JOVE commands, set *send-typeout-to-buffer* to "on" and JOVE will store the key bindings in a buffer which you can save to a file and then print.

16.54. describe-command (Not Bound)

This prints some info on a specified command.

16.55. describe-key (Not Bound)

This waits for you to type a key and then tells the name of the command that gets invoked every time that key is hit. Once you have the name of the command you can use the *describe-command* command to find out exactly what it does.

16.56. describe-variable (Not Bound)

This prints some info on a specified variable.

16.57. digit (ESC [0-9])

This reads a numeric argument. When you type "ESC" followed by a number, "digit" keeps reading numbers until you type some other command. Then that command is executes with the numeric argument you specified.

16.58. digit-1 (Not Bound)

This pretends you typed "ESC 1". This is useful for terminals that have keypads that send special sequences for numbers typed on the keypad as opposed to numbers typed from the keyboard. This can save having type "ESC" when you want to specify an argument.

16.59. digit-2 (Not Bound)

This pretends you typed "ESC 2". This is useful for terminals that have keypads that send special sequences for numbers typed on the keypad as opposed to numbers typed from the keyboard. This can save having type "ESC" when you want to specify an argument.

16.60. digit-3 (Not Bound)

This pretends you typed "ESC 3". This is useful for terminals that have keypads that send special sequences for numbers typed on the keypad as opposed to numbers typed from the keyboard. This can save having type "ESC" when you want to specify an argument.

16.61. digit-4 (Not Bound)

This pretends you typed "ESC 4". This is useful for terminals that have keypads that send special sequences for numbers typed on the keypad as opposed to numbers typed from the keyboard. This can save having type "ESC" when you want to specify an argument.

16.62. digit-5 (Not Bound)

This pretends you typed "ESC 5". This is useful for terminals that have keypads that send special sequences for numbers typed on the keypad as opposed to numbers typed from the keyboard. This can save having type "ESC" when you want to specify an argument.

16.63. digit-6 (Not Bound)

This pretends you typed "ESC 6". This is useful for terminals that have keypads that send special sequences for numbers typed on the keypad as opposed to numbers typed from the keyboard. This can save having type "ESC" when you want to specify an argument.

16.64. digit-7 (Not Bound)

This pretends you typed "ESC 7". This is useful for terminals that have keypads that send special sequences for numbers typed on the keypad as opposed to numbers typed from the keyboard. This can save having type "ESC" when you want to specify an argument.

16.65. digit-8 (Not Bound)

This pretends you typed "ESC 8". This is useful for terminals that have keypads that send special sequences for numbers typed on the keypad as opposed to numbers typed from the keyboard. This can save having type "ESC" when you want to specify an argument.

16.66. digit-9 (Not Bound)

This pretends you typed "ESC 9". This is useful for terminals that have keypads that send special sequences for numbers typed on the keypad as opposed to numbers typed from the keyboard. This can save having type "ESC" when you want to specify an argument.

16.67. digit-0 (Not Bound)

This pretends you typed "ESC 0". This is useful for terminals that have keypads that send special sequences for numbers typed on the keypad as opposed to numbers typed from the keyboard. This can save having type "ESC" when you want to specify an argument.

16.68. dirs (Not Bound)

This prints out the directory stack. See the "cd", "pushd", "popd" commands for more info.

16.69. disable-biff (variable)

When this is set, JOVE disables biff when you're editing and enables it again when you get out of JOVE, or when you pause to the parent shell or push to a new shell. (This means arrival of new mail will not be immediately apparent but will not cause indiscriminate writing on the display). The default is "off".

16.70. dstop-process (Not Bound)

Send the "dsusp" character to the current process. This is the character that suspends a process on the next read from the terminal. Most people have it set to C-Y. This only works if you have the interactive process feature, and if you are in a buffer bound to a process.

16.71. edit-word-abbrevs (Not Bound)

This creates a buffer with a list of each abbreviation and the phrase it expands into, and enters a recursive edit to let you change the abbreviations or add some more. The format of this list is "abbreviation:phrase" so if you add some more you should follow that format. It's probably simplest just to copy some already existing abbreviations and edit them. When you are done you type "C-X C-C" to exit the recursive edit.

16.72. end-of-file (ESC >)

This moves point forward to the end of the buffer. This sometimes prints the "Point Pushed" message. If the end of the buffer isn't on the screen JOVE will set the mark so you can go back to where you were if you want.

16.73. end-of-line (C-E)

This moves point to the end of the current line. If the line is too long to fit on the screen JOVE will scroll the line to the left to make the end of the line visible. The line will slide back to its normal position when you move backward past the leftmost visible character or when you move off the line altogether.

16.74. end-of-window (ESC .)

This moves point to the last character in the window.

16.75. eof-process (Not Bound)

Sends EOF to the current interactive process. This only works on versions of JOVE which run under 4.2-3 BSD VAX UNIX. You can't send EOF to processes on the 2.9 BSD PDP-11 UNIX.

16.76. erase-buffer (Not Bound)

This erases the contents of the specified buffer. This is like *delete-buffer* except it only erases the contents of the buffer, not the buffer itself. If you try to erase a buffer that needs saving you will be asked to confirm it.

16.77. error-window-size (variable)

This is the percentage of the screen to use for the error-window on the screen. When you execute *compile-it, error-window-size* percent of the screen will go to the error window. If the window already exists and is a different size, it is made to be this size. The default value is 20%.

16.78. exchange-point-and-mark (C-X C-X)

This moves point to mark and makes mark the old point. This is for quickly moving from one end of the region to another.

16.79. execute-named-command (ESC X)

This is the way to execute a command that isn't bound to any key. When you are prompted with ": " you can type the name of the command. You don't have to type the entire name. Once the command is unambiguous you can type Space and JOVE will fill in the rest for you. If you are not sure of the name of the command, type "?" and JOVE will print a list of all the commands that you could possibly match given what you've already typed. If you don't have any idea what the command's name is but you know it has something to do with windows (for example), you can do "ESC X apropos window" and JOVE will print a list of all the commands that are related to windows. If you find yourself constantly executing the same commands this way you probably want to bind them to keys so that you can execute them more quickly. See the *bind-to-key* command.

16.80. execute-keyboard-macro (C-X E)

This executes the keyboard macro. If you supply a numeric argument the macro is executed that many times.

16.81. execute-macro (Not Bound)

This executes a specified macro. If you supply a numeric argument the macro is executed that many times.

16.82. exit-jove (C-X C-C)

This exits JOVE. If any buffers need saving JOVE will print a warning message and ask for confirmation. If you leave without saving your buffers all your work will be lost. If you made a mistake and really do want to exit then you can. If you are in a recursive editing level *exit-jove* will return you from that.

16.83. file-creation-mode (variable)

This variable has an octal value. It contains the mode (see *chmod(1)*) with which files should be created. This mode gets modified by your current umask setting (see *umask(1)*). The default value is usually *0666* or *0644.*

16.84. files-should-end-with-newline (variable)

This variable indicates that all files should always have a newline at the end. This is often necessary for line printers and the like. When set, if JOVE is writing a file whose last character is not a newline, it will add one automatically.

16.85. fill-comment (Not Bound)

This command fills in your C comments to make them pretty and readable. This filling is done according the variable *comment-format.*

```
/*
 * the default format makes comments like this.
 */
```

This can be changed by changing the format variable. Other languages may be supported by changing the format variable appropriately. The formatter looks backwards from dot for an open comment symbol. If found, all indentation is done relative the position of the first character of the open symbol. If there is a matching close symbol, the entire comment is formatted. If not, the region between dot and the open symbol is reformatted.

16.86. fill-paragraph (ESC J)

This rearranges words between lines so that all the lines in the current paragraph extend as close to the right margin as possible, ensuring that none of the lines will be greater than the right margin. The default value for *right-margin* is 78, but can be changed with the *set* and *right-margin-here* commands. JOVE has a complicated algorithm for determining the beginning and end of the paragraph. In the normal case JOVE will give all the lines the same indent as they currently have, but if you wish to force a new indent you can supply a numeric argument to *fill-paragraph* (e.g., by typing C-U ESC J) and JOVE will indent each line to the column specified by the *left-margin* variable. See also the *left-margin* variable and *left-margin-here* command.

16.87. fill-region (Not Bound)

This is like *fill-paragraph,* except it operates on a region instead of just a paragraph.

16.88. filter-region (Not Bound)

This sends the text in the region to a UNIX command, and replaces the region with the output from that command. For example, if you are lazy and don't like to take the time to write properly indented C code, you can put the region around your C file and *filter-region* it through *cb,* the UNIX C beautifier. If you have a file that contains a bunch of lines that need to be sorted you can do that from inside JOVE too, by filtering the region through the *sort* UNIX command. Before output from the command replaces the region JOVE stores the old text in the kill ring, so if you are unhappy with the results you can easily get back the old text with "C-Y".

16.89. find-file (C-X C-F)

This visits a file into its own buffer and then selects that buffer. If you've already visited this file in another buffer, that buffer is selected. If the file doesn't yet exist, JOVE will print "(New file)" so that you know.

16.90. find-tag (C-X T)

This finds the file that contains the specified tag. JOVE looks up tags by default in the "tags" file in the current directory. You can change the default tag name by setting the *tag-file* variable to another name. If you specify a numeric argument to this command, you will be prompted for a tag file. This is a good way to specify another tag file without changing the default. If the tag cannot be found the error is reported and point stays where it is.

16.91. find-tag-at-point (Not Bound)

This finds the file that contains the tag that point is currently on. See *find-tag.*

16.92. first-non-blank (ESC M)

This moves point back to the indent of the current line.

16.93. forward-character (C-F)

This moves forward over a single character. If point is at the end of the line it moves to the beginning of the next one.

16.94. forward-paragraph (ESC])

This moves point forward to the end of the current or next paragraph. Paragraphs are bounded by lines that begin with a Period or Tab, or by blank lines; a change in indentation may also signal a break between paragraphs, except that JOVE allows the first line of a paragraph to be indented differently from the other lines.

16.95. forward-s-expression (ESC C-F)

This moves point forward over a s-expression. If the first significant character after point is "(", this moves past the matching ")". If the character begins an identifier, this moves just past it. This is mode dependent, so this will move over atoms in LISP mode and C identifiers in C mode. JOVE also matches "{".

16.96. forward-sentence (ESC E)

This moves point forward to the end of the current or next sentence. JOVE considers the end of a sentence to be the characters ".", "!" or "?" followed by a Return, or one or more spaces.

16.97. forward-word (ESC F)

This moves point forward to the end of the current or next word.

16.98. fundamental-mode (Not Bound)

This sets the major mode to Fundamental. This affects what JOVE considers as characters that make up words. For instance, Single-quote is not part of a word in Fundamental mode, but is in Text mode.

16.99. goto-line (ESC G)

If a numeric argument is supplied point moves to the beginning of that line. If no argument is supplied, point remains where it is. This is so you don't lose your place unintentionally, by accidentally hitting the "G" instead of "F".

16.100. grind-s-expr (Not Bound)

When point is positioned on a "(", this re-indents that LISP expression.

16.101. grow-window (C-X ^)

This makes the current window one line bigger. This only works when there is more than one window and provided there is room to change the size.

16.102. paren-flash () }])

This handles the C mode curly brace indentation, the Lisp mode paren indentation, and the Show Match mode paren/curly brace/square bracket flashing.

16.103. handle-tab (Tab)

This handles indenting to the "right" place in C and Lisp mode, and just inserts itself in Text mode.

16.104. i-search-forward (Not Bound)

Incremental search. Like search-forward except that instead of prompting for a string and searching for that string all at once, it accepts the string one character at a time. After each character you type as part of the search string, it searches for the entire string so far. When you like what it found, type the Return key to finish the search. You can take back a character with Rubout and the search will back up to the position before that character was typed. C-G aborts the search.

16.105. i-search-reverse (Not Bound)

Incremental search. Like search-reverse except that instead of prompting for a string and searching for that string all at once, it accepts the string one character at a time. After each character you type as part of the search string, it searches for the entire string so far. When you like what it found, type the Return key to finish the search. You can take back a character with Rubout and the search will back up to the position before that character was typed. C-G aborts the search.

16.106. insert-file (C-X C-I)

This inserts a specified file into the current buffer at point. Point is positioned at the beginning of the inserted file.

16.107. internal-tabstop (variable)

The number of spaces JOVE should print when it displays a tab character. The default value is 8.

16.108. interrupt-process (Not Bound)

This sends the interrupt character (usually C-C) to the interactive process in the current buffer. This is only for versions of JOVE that have the interactive processes feature. This only works when you are inside a buffer that's attached to a process.

16.109. i-shell (Not Bound)

This starts up an interactive shell in a window. JOVE uses "shell-1" as the name of the buffer in which the interacting takes place. See the manual for information on how to use interactive processes.

16.110. i-shell-command (Not Bound)

This is like *shell-command* except it lets you continue with your editing while the command is running. This is really useful for long running commands with sporadic output. See the manual for information on how to use interactive processes.

16.111. kill-next-word (ESC D)

This kills the text from point to the end of the current or next word.

16.112. kill-previous-word (ESC Rubout)

This kills the text from point to the beginning of the current or previous word.

16.113. kill-process (Not Bound)

This command prompts for a buffer name or buffer number (just as select-buffer does) and then sends the process in that buffer a kill signal (9).

16.114. kill-region (C-W)

This deletes the text in the region and saves it on the kill ring. Commands that delete text but save it on the kill ring all have the word "kill" in their names. Type "C-Y" to yank back the most recent kill.

16.115. kill-s-expression (ESC C-K)

This kills the text from point to the end of the current or next s-expression.

16.116. kill-some-buffers (Not Bound)

This goes through all the existing buffers and asks whether or not to kill them. If you decide to kill a buffer, and it turns out that the buffer is modified, JOVE will offer to save it first. This is useful for when JOVE runs out of memory to store lines (this only happens on PDP-11's) and you have lots of buffers that you are no longer using.

16.117. kill-to-beginning-of-sentence (C-X Rubout)

This kills from point to the beginning of the current or previous sentence.

16.118. kill-to-end-of-line (C-K)

This kills from point to the end of the current line. When point is at the end of the line the line separator is deleted and the next line is joined with current one. If a numeric argument is supplied that many lines are killed; if the argument is negative that many lines *before* point are killed; if the argument is zero the text from point to the beginning of the line is killed.

16.119. kill-to-end-of-sentence (ESC K)

This kills from point to the end of the current or next sentence. If a negative numeric argument is supplied it kills from point to the beginning of the current or previous sentence.

16.120. left-margin (variable)

This is how far lines should be indented when auto-indent mode is on, or when the *newline-and-indent* command is run (usually by typing LineFeed). It is also used by fill-paragraph and auto-fill mode. If the value is zero (the default) then the left margin is determined from the surrounding lines.

16.121. left-margin-here (Not Bound)

This sets the *left-margin* variable to the current position of point. This is an easy way to say, "Make the left margin begin here," without having to count the number of spaces over it actually is.

16.122. lisp-mode (Not Bound)

This turns on Lisp mode. Lisp mode is one of four mutually exclusive major modes: Fundamental, Text, C, and Lisp. In Lisp mode, the characters Tab and) are treated specially, similar to the way they are treated in C mode. Also, Auto Indent mode is affected, and handled specially.

16.123. list-buffers (C-X C-B)

This types out a list containing various information about each buffer. Right now that list looks like this:

```
(* means the buffer needs saving)
NO   Lines  Type        Name            File
--   -----  ----        ----            ----
1    1      File        Main            [No file]
2    1      Scratch   * Minibuf         [No file]
3    519    File      * commands.doc    commands.doc
```

The first column lists the buffer's number. When JOVE prompts for a buffer name you can either type in the full name, or you can simply type the buffer's number. The second column is the number of lines in the buffer. The third says what type of buffer. There are four types: "File", "Scratch", "Process", "I-Process". "File" is simply a buffer that holds a file; "Scratch" is for buffers that JOVE uses internally; "Process" is one that holds the output from a UNIX command; "I-Process" is one that has an interactive process attached to it. The next column contains the name of the buffer. And the last column is the name of the file that's attached to the buffer. In this case, both Minibuf and commands.doc have been changed but not yet saved. In fact Minibuf won't be saved since it's an internal JOVE buffer that I don't even care about.

16.124. list-processes (Not Bound)

This makes a list somewhat like "list-buffers" does, except its list consists of the current interactive processes. Right now the list looks like this:

```
Buffer          Status          Command name
------          ------          ------- ----
shell-1         Running         i-shell
fgrep           Done            fgrep -n Buffer *.c
```

The first column has the name of the buffer to which the process is attached. The second has the status of the process; if a process has exited normally the status is "Done" as in fgrep; if the process exited with an error the status is "Exit N" where N is the value of the exit code; if the process was killed by some signal the status is the name of the signal that was used; otherwise the process is running. The last column is the name of the command that is being run.

16.125. mailbox (variable)

Set this to the full pathname of your mailbox. JOVE will look here to decide whether or not you have any unread mail. This defaults to /usr/spool/mail/$USER, where $USER is set to your login name.

16.126. mail-check-frequency (variable)

This is how often (in seconds) JOVE should check your mailbox for incoming mail. See also the *mailbox* and *disable-biff* variables.

16.127. make-backup-files (variable)

If this variable is set, then whenever JOVE writes out a file, it will move the previous version of the file (if there was one) to "#filename". This is often convenient if you save a file by accident. The default value of this variable is "off". *Note:* this is an optional part of JOVE, and your guru may not have it enabled, so it may not work.

16.128. make-buffer-unmodified (ESC ~)

This makes JOVE think the selected buffer hasn't been changed even if it has. Use this when you accidentally change the buffer but don't want it considered changed. Watch the mode line to see the * disappear when you use this command.

16.129. make-macro-interactive (Not Bound)

This command is meaningful only while you are defining a keyboard macro. Ordinarily, when a command in a macro definition requires a trailing text argument (file name, search string, etc.), the argument you supply becomes part of the macro definition. If you want to be able to supply a different argument each time the macro is used, then while you are defining it, you should give the make-macro-interactive command just before typing the argument which will be used during the definition process. Note: you must bind this command to a key in order to use it; you can't say ESC X make-macro-interactive.

16.130. mark-threshold (variable)

This variable contains the number of lines point may move by before the mark is set. If, in a search or something, point moves by more than this many lines, the mark is set so that you may return easily. The default value of this variable is 22 (one screenful, on most terminals).

16.131. marks-should-float (variable)

When this variable is "off", the position of a mark is remembered as a line number within the buffer and a character number within the line. If you add or delete text before the mark, it will no longer point to the text you marked originally because that text is no longer at the same line and character number. When this variable is "on", the position of a mark is adjusted to compensate for each insertion and deletion. This makes marks much more sensible to use, at the cost of slowing down insertion and deletion somewhat. The default value is "on".

16.132. match-regular-expressions (variable)

When set, JOVE will match regular expressions in search patterns. This makes special the characters ., *, [,], ^, and $, and the two-character sequences \<, \>, \{, \} and \|. See the *ed(1)* manual page, the tutorial "Advanced Editing in UNIX ", and the section above "Searching with Regular Expressions" for more information.

16.133. meta-key (variable)

You should set this variable to "on" if your terminal has a real Meta key. If your terminal has such a key, then a key sequence like ESC Y can be entered by holding down Meta and typing Y.

16.134. mode-line (variable)

The format of the mode line can be determined by setting this variable. The items in the line are specified using a printf(3) format, with the special things being marked as "%x". Digits may be used between the 'x' may be:

C	check for new mail, and displays "[New mail]" if there is any (see also the mail-check-interval and disable-biff variables)
F	the current file name, with leading path stripped
M	the current list of major and minor modes
b	the current buffer name
c	the fill character (-)
d	the current directory
e	end of string--this must be the last item in the string
f	the current file name
l	the current load average (updated automatically)
m	the buffer-modified symbol (*)
n	the current buffer number
s	space, but only if previous character is not a space
t	the current time (updated automatically)
[ ]	the square brackets printed when in a recursive edit
( )	items enclosed in %(... %) will only be printed on the bottom mode line, rather than copied when the window is split

In addition, any other character is simply copied into the mode line. Characters may be escaped with a backslash. To get a feel for all this, try typing "ESC X print mode-line" and compare the result with your current mode line.

16.135. mode-line-should-standout (variable)

If set, the mode line will be printed in reverse video, if your terminal supports it. The default for this variable is "off".

16.136. name-keyboard-macro (Not Bound)

This copies the keyboard macro and gives it a name freeing up the keyboard macro so you can define some more. Keyboard macros with their own names can be bound to keys just like built in commands can. See the *read-macros-file-file* and *write-macros-to-file* commands.

16.137. newline (Return)

This divides the current line at point moving all the text to the right of point down onto the newly created line. Point moves down to the beginning of the new line.

16.138. newline-and-backup (C-O)

This divides the current line at point moving all the text to the right of point down onto the newly created line. The difference between this and "newline" is that point does not move down to the beginning of the new line.

16.139. newline-and-indent (LineFeed)

This behaves the same was as Return does when in Auto Indent mode. This makes Auto Indent mode obsolete but it remains in the name of backward compatibility.

16.140. next-error (C-X C-N)

This moves to the next error in the list of errors that were parsed with *parse-errors* or *parse-special-errors.* In one window the list of errors is shown with the current one always at the top. In another window is the file that contains the error. Point is positioned in this window on the line where the error occurred.

16.141. next-line (C-N)

This moves down to the next line.

16.142. next-page (C-V)

This displays the next page of the buffer by taking the bottom line of the window and redrawing the window with it at the top. If there isn't another page in the buffer JOVE rings the bell. If a numeric argument is supplied the screen is scrolled up that many lines; if the argument is negative the screen is scrolled down.

16.143. next-window (C-X N)

This moves into the next window. Windows live in a circular list so when you're in the bottom window and you try to move to the next one you are moved to the top window. It is an error to use this command with only one window.

16.144. number-lines-in-window (Not Bound)

This displays the line numbers for each line in the buffer being displayed. The number isn't actually part of the text; it's just printed before the actual buffer line is. To turn this off you run the command again; it toggles.

16.145. over-write-mode (Not Bound)

This turns Over Write mode on (or off if it's currently on) in the selected buffer. When on, this mode changes the way the self-inserting characters work. Instead of inserting themselves and pushing the rest of the line over to the right, they replace or over-write the existing character. Also, Rubout replaces the character before point with a space instead of deleting it. When Over Write mode is on "OvrWt" is displayed on the mode line.

16.146. page-next-window (ESC C-V)

This displays the next page in the next window. This is exactly the same as "C-X N C-V C-X P".

16.147. paren-flash-delay (variable)

How long, in tenths of seconds, JOVE should pause on a matching parenthesis in *Show* mode. The default is 5.

16.148. parse-errors (Not Bound)

This takes the list of C compilation errors (or output from another program in the same format) in the current buffer and parses them for use with the *next-error* and *previous-error* and *current-error* commands. This is a very useful tool and helps with compiling C programs and when used in conjunction with the "grep" UNIX command very helpful in making changes to a bunch of files. This command understands errors produced by cc, cpp, and lint; plus any other program with the same format (e.g., "grep -n"). JOVE visits each file that has an error and remembers each line that contains an error. It doesn't matter if later you insert or delete some lines in the buffers containing errors; JOVE remembers where they are regardless. *next-error* is automatically executed after one of the parse commands, so you end up at the first error.

16.149. parse-special-errors (Not Bound)

This parses errors in an unknown format. Error parsing works with regular expression search strings with \('s around the the file name and the line number. So, you can use *parse-special-errors* to parse lines that are in a slightly different format by typing in your own search string. If you don't know how to use regular expressions you can't use this command.

16.150. parse-spelling-errors-in-buffer (Not Bound)

This parses a list of words in the current buffer and looks them up in another buffer that you specify. This will probably go away soon.

16.151. pause-jove (ESC S)

This stops JOVE and returns control to the parent shell. This only works for users using the C-shell, and on systems that have the job control facility. To return to JOVE you type "fg" to the C-shell.

16.152. physical-tabstop (variable)

How many spaces your terminal prints when it prints a tab character.

16.153. pop-mark (Not Bound)

This gets executed when you run *set-mark* with a numeric argument. JOVE remembers the last 16 marks and you use *pop-mark* to go backward through the ring of marks. If you execute " *pop-mark* enough times you will eventually get back to where you started.

16.154. popd (Not Bound)

This pops one entry off the directory stack. Entries are pushed with the *pushd* command. The names were stolen from the C-shell and the behavior is the same.

16.155. previous-error (C-X C-P)

This is the same as *next-error* except it goes to the previous error. See *next-error* for documentation.

16.156. previous-line (C-P)

This moves up to the previous line.

16.157. previous-page (ESC V)

This displays the previous page of the current buffer by taking the top line and redrawing the window with it at the bottom. If a numeric argument is supplied the screen is scrolled down that many lines; if the argument is negative the screen is scrolled up.

16.158. previous-window (C-X P and C-X O)

This moves into the next window. Windows live in a circular list so when you're in the top window and you try to move to the previous one you are moved to the bottom window. It is an error to use this command with only one window.

16.159. print (Not Bound)

This prints the value of a JOVE variable.

16.160. print-message (Not Bound)

This command prompts for a message, and then prints it on the bottom line where JOVE messages are printed.

16.161. process-bind-to-key (Not Bound)

This command is identical to bind-to-key, except that it only affects your bindings when you are in a buffer attached to a process. When you enter the process buffer, any keys bound with this command will automatically take their new values. When you switch to a non-process buffer, the old bindings for those keys will be restored. For example, you might want to execute

```
process-bind-to-key stop-process ˆZ
process-bind-to-key interrupt-process ˆC
```

Then, when you start up an interactive process and switch into that buffer, C-Z will execute stop-process and C-C will execute interrupt- process. When you switch back to a non-process buffer, C-Z will go back to executing scroll-up (or whatever you have it bound to).

16.162. process-newline (Return)

This this only gets executed when in a buffer that is attached to an interactive-process. JOVE does two different things depending on where you are when you hit Return. When you're at the end of the I-Process buffer this does what Return normally does, except it also makes the line available to the process. When point is positioned at some other position that line is copied to the end of the buffer (with the prompt stripped) and point is moved there with it, so you can then edit that line before sending it to the process. This command *must* be bound to the key you usually use to enter shell commands (Return), or else you won't be able to enter any.

16.163. process-prompt (variable)

What a prompt looks like from the i-shell and i-shell-command processes. The default is "% ", the default C-shell prompt. This is actually a regular expression search string. So you can set it to be more than one thing at once using the \| operator. For instance, for LISP hackers, the prompt can be

"% -> <[0-9]>: ".

16.164. push-shell (Not Bound)

This spawns a child shell and relinquishes control to it. This works on any version of UNIX, but this isn't as good as *pause-jove* because it takes time to start up the new shell and you get a brand new environment every time. To return to JOVE you type "C-D".

16.165. pushd (Not Bound)

This pushes a directory onto the directory stack and cd's into it. It asks for the directory name but if you don't specify one it switches the top two entries no the stack. It purposely behaves the same as C-shell's *pushd.*

16.166. pwd (Not Bound)

This prints the working directory.

16.167. quadruple-numeric-argument (C-U)

This multiplies the numeric argument by 4. So, "C-U C-F" means forward 4 characters and "C-U C-U C-N" means down 16 lines.

16.168. query-replace-string (ESC Q)

This replaces the occurrences of a specified string with a specified replacement string. When an occurrence is found point is moved to it and then JOVE asks what to do. The options are:

Space	to replace this occurrence and go on to the next one.
Period	to replace this occurrence and then stop.
Rubout	to skip this occurrence and go on to the next one.
C-R	to enter a recursive edit. This lets you temporarily suspend the replace, do some editing, and then return to continue where you left off. To continue with the Query Replace type "C-X C-C" as if you were trying to exit JOVE. Normally you would but when you are in a recursive edit all it does is exit that recursive editing level.
C-W	to delete the matched string and then enter a recursive edit.
U	to undo the last replacement.
P or !	to go ahead and replace the remaining occurrences without asking.
Return	to stop the Query Replace.

The search for occurrences starts at point and goes to the end of the buffer, so to replace in the entire buffer you must first go to the beginning.

16.169. quit-process (Not Bound)

This is the same as typing "C-\" (the Quit character) to a normal UNIX process, except it sends it to the current process in JOVE. This is only for versions of JOVE that have the interactive processes feature. This only works when you are inside a buffer that's attached to a process.

16.170. quoted-insert (C-Q)

This lets you insert characters that normally would be executed as other JOVE commands. For example, to insert "C-F" you type "C-Q C-F".

16.171. read-word-abbrev-file (Not Bound)

This reads a specified file that contains a bunch of abbreviation definitions, and makes those abbreviations available. If the selected buffer is not already in Word Abbrev mode this command puts it in that mode.

16.172. read-macros-from-file (Not Bound)

This reads the specified file that contains a bunch of macro definitions, and defines all the macros that were currently defined when the file was created. See *write-macros-to-file* to see how to save macros.

16.173. redraw-display (C-L)

This centers the line containing point in the window. If that line is already in the middle the window is first cleared and then redrawn. If a numeric argument is supplied, the line is positioned at that offset from the top of the window. For example, "ESC 0 C-L" positions the line containing point at the top of the window.

16.174. recursive-edit (Not Bound)

This enters a recursive editing level. This isn't really very useful. I don't know why it's available for public use. I think I'll delete it some day.

16.175. rename-buffer (Not Bound)

This lets you rename the current buffer.

16.176. replace-in-region (Not Bound)

This is the same as *replace-string* except that it is restricted to occurrences between Point and Mark.

16.177. replace-string (ESC R)

This replaces all occurrences of a specified string with a specified replacement string. This is just like *query-replace-string* except it replaces without asking.

16.178. right-margin (variable)

Where the right margin is for *Auto Fill* mode and the *justify-paragraph* and *justify-region* commands. The default is 78.

16.179. right-margin-here (Not Bound)

This sets the *right-margin* variable to the current position of point. This is an easy way to say, "Make the right margin begin here," without having to count the number of spaces over it actually is.

16.180. save-file (C-X C-S)

This saves the current buffer to the associated file. This makes your changes permanent so you should be sure you really want to. If the buffer has not been modified *save-file* refuses to do the save. If you really do want to write the file you can use "C-X C-W" which executes *write-file.*

16.181. scroll-down (ESC Z)

This scrolls the screen one line down. If the line containing point moves past the bottom of the window point is moved up to the center of the window. If a numeric argument is supplied that many lines are scrolled; if the argument is negative the screen is scrolled up instead.

16.182. scroll-step (variable)

How many lines should be scrolled if the *previous-line* or *next-line* commands move you off the top or bottom of the screen. You may wish to decrease this variable if you are on a slow terminal.

16.183. scroll-up (C-Z)

This scrolls the screen one line up. If the line containing point moves past the top of the window point is moved down to the center of the window. If a numeric argument is supplied that many lines are scrolled; if the argument is negative the screen is scrolled down instead.

16.184. search-exit-char (variable)

Set this to the character you want to use to exit incremental search. The default is Newline, which makes i-search compatible with normal string search.

16.185. search-forward (C-S)

This searches forward for a specified search string and positions point at the end of the string if it's found. If the string is not found point remains unchanged. This searches from point to the end of the buffer, so any matches before point will be missed.

16.186. search-reverse (C-R)

This searches backward for a specified search string and positions point at the beginning if the string if it's found. If the string is not found point remains unchanged. This searches from point to the beginning of the buffer, so any matches after point will be missed.

16.187. select-buffer (C-X B)

This selects a new or already existing buffer making it the current one. You can type either the buffer name or number. If you type in the name you need only type the name until it is unambiguous, at which point typing Escape or Space will complete it for you. If you want to create a new buffer you can type Return instead of Space, and a new empty buffer will be created.

16.188. self-insert (Most Printing Characters)

This inserts the character that invoked it into the buffer at point. Initially all but a few of the printing characters are bound to *self-insert.*

16.189. send-typeout-to-buffer (variable)

When this is set JOVE will send output that normally overwrites the screen (temporarily) to a buffer instead. This affects commands like *list-buffers, list-processes,* and other commands that use command completion. The default value is "off".

16.190. set (Not Bound)

This gives a specified variable a new value. Occasionally you'll see lines like "set this variable to that value to do this". Well, you use the *set* command to do that.

16.191. set-mark (C-@)

This sets the mark at the current position in the buffer. It prints the message "Point pushed" on the message line. It says that instead of "Mark set" because when you set the mark the previous mark is still remembered on a ring of 16 marks. So "Point pushed" means point is pushed onto the ring of marks and becomes the value of "the mark". To go through the ring of marks you type "C-U C-@", or execute the *pop-mark* command. If you type this enough times you will get back to where you started.

16.192. shell (variable)

The shell to be used with all the shell commands command. If your SHELL environment variable is set, it is used as the value of *shell;* otherwise "/bin/csh" is the default.

16.193. shell-command (C-X !)

This runs a UNIX command and places the output from that command in a buffer. JOVE creates a buffer that matches the name of the command you specify and then attaches that buffer to a window. So, when you have only one window running this command will cause JOVE to split the window and attach the new buffer to that window. Otherwise, JOVE finds the most convenient of the available windows and uses that one instead. If the buffer already exists it is first emptied, except that if it's holding a file, not some output from a previous command, JOVE prints an error message and refuses to execute the command. If you really want to execute the command you should delete that buffer (saving it first, if you like) or use *shell-command-to-buffer,* and try again.

16.194. shell-command-to-buffer (Not Bound)

This is just like *shell-command* except it lets you specify the buffer to use instead of JOVE.

16.195. shell-flags (variable)

This defines the flags that are passed to shell commands. The default is "-c". See the *shell* variable to change the default shell.

16.196. show-match-mode (Not Bound)

This turns on Show Match mode (or off if it's currently on) in the selected buffer. This changes "}" and ")" so that when they are typed the are inserted as usual, and then the cursor flashes back to the matching "{" or "(" (depending on what was typed) for about half a second, and then goes back to just after the "}" or ")" that invoked the command. This is useful for typing in complicated expressions in a program. You can change how long the cursor sits on the matching paren by setting the "paren-flash-delay" variable in tenths of a second. If the matching "{" or "(" isn't visible nothing happens.

16.197. shrink-window (Not Bound)

This makes the current window one line shorter, if possible. Windows must be at least 2 lines high, one for the text and the other for the mode line.

16.198. source (Not Bound)

This reads a bunch of JOVE commands from a file. The format of the file is the same as that in your initialization file (your ".joverc") in your main directory. There should be one command per line and it should be as though you typed "ESC X" while in JOVE. For example, here's part of my initialization file:

```
bind-to-key i-search-reverse ^R
bind-to-key i-search-forward ^S
bind-to-key pause-jove ^[S
```

What they do is make "C-R" call the *i-search-reverse* command and "C-S" call *i-search-forward* and "ESC S" call *pause-jove.*

16.199. spell-buffer (Not Bound)

This runs the current buffer through the UNIX *spell* program and places the output in buffer "Spell". Then JOVE lets you edit the list of words, expecting you to delete the ones that you don't care about, i.e., the ones you know are spelled correctly. Then the *parse-spelling-errors-in-buffer* command comes along and finds all the misspelled words and sets things up so the error commands work.

16.200. split-current-window (C-X 2)

This splits the current window into two equal parts (providing the resulting windows would be big enough) and displays the selected buffer in both windows. Use "C-X 1" to go back to 1 window mode.

16.201. start-remembering (C-X ()

This starts remembering your key strokes in the Keyboard macro. To stop remembering you type "C-X)". Because of a bug in JOVE you can't stop remembering by typing "ESC X stop-remembering"; *stop-remembering* must be bound to "C-X)" in order to make things work correctly. To execute the remembered key strokes you type "C-X E" which runs the *execute-keyboard-macro* command. Sometimes you may want a macro to accept different input each time it runs. To see how to do this, see the *make-macro-interactive* command.

16.202. stop-process (Not Bound)

This sends a stop signal (C-Z, for most people) to the current process. It only works if you have the interactive process feature, and you are in a buffer attached to a process.

16.203. stop-remembering (C-X))

This stop the definition of the keyboard macro. Because of a bug in JOVE, this must be bound to "C-X)". Anything else will not work properly.

16.204. string-length (Not Bound)

This prints the number of characters in the string that point sits in. Strings are surrounded by double quotes. JOVE knows that "\007" is considered a single character, namely "C-G", and also knows about other common ones, like "\r" (Return) and "\n" (LineFeed). This is mostly useful only for C programmers.

16.205. suspend-jove (ESC S)

This is a synonym for *pause-jove*.

16.206. sync-frequency (variable)

The temporary files used by JOVE are forced out to disk every *sync-frequency* modifications. The default is 50, which really makes good sense. Unless your system is very unstable, you probably shouldn't fool with this.

16.207. tag-file (variable)

This the name of the file in which JOVE should look up tag definitions. The default value is "./tags".

16.208. text-mode (Not Bound)

This sets the major mode to Text. Currently the other modes are Fundamental, C and Lisp mode.

16.209. transpose-characters (C-T)

This switches the character before point with the one after point, and then moves forward one. This doesn't work at the beginning of the line, and at the end of the line it switches the two characters before point. Since point is moved forward, so that the character that was before point is still before point, you can use "C-T" to drag a character down the length of a line. This command pretty quickly becomes very useful.

16.210. transpose-lines (C-X C-T)

This switches the current line with the one above it, and then moves down one so that the line that was above point is still above point. This, like *transpose-characters*, can be used to drag a line down a page.

16.211. unbind-key (Not Bound)

Use this to unbind *any* key sequence. You can use this to unbind even a prefix command, since this command does not use "key-map completion". For example, "ESC X unbind-key ESC [" unbinds the sequence "ESC [". This is useful for "turning off" something set in the system-wide ".joverc" file.

16.212. update-time-frequency (variable)

How often the mode line is updated (and thus the time and load average, if you display them). The default is 30 seconds.

16.213. use-i/d-char (variable)

If your terminal has insert/delete character capability you can tell JOVE not to use it by setting this to "off". In my opinion it is only worth using insert/delete character at low baud rates. WARNING: if you set this to "on" when your terminal doesn't have insert/delete character capability, you will get weird (perhaps fatal) results.

16.214. version (Not Bound)

Displays the version number of this JOVE.

16.215. visible-bell (variable)

Use the terminal's visible bell instead of beeping. This is set automatically if your terminal has the capability.

16.216. visible-spaces-in-window (Not Bound)

This displays an underscore character instead of each space in the window and displays a greater-than followed by spaces for each tab in the window. The actual text in the buffer is not changed; only the screen display is affected. To turn this off you run the command again; it toggles.

16.217. visit-file (C-X C-V)

This reads a specified file into the current buffer replacing the old text. If the buffer needs saving JOVE will offer to save it for you. Sometimes you use this to start over, say if you make lots of changes and then change your mind. If that's the case you don't want JOVE to save your buffer and you answer "NO" to the question.

16.218. window-find (C-X 4)

This lets you select another buffer in another window three different ways. This waits for another character which can be one of the following:

T	Finds a tag in the other window.
F	Finds a file in the other window.
B	Selects a buffer in the other window.

This is just a convenient short hand for "C-X 2" (or "C-X O" if there are already two windows) followed by the appropriate sequence for invoking each command. With this, though, there isn't the extra overhead of having to redisplay. In addition, you don't have to decide whether to type "C-X 2" or "C-X O" since "C-X 4" does the right thing.

16.219. word-abbrev-mode (Not Bound)

This turns on Word Abbrev mode (or off if it's currently on) in the selected buffer. Word Abbrev mode lets you specify a word (an abbreviation) and a phrase with which JOVE should substitute the abbreviation. You can use this to define words to expand into long phrases, e.g., "jove" can expand into "Jonathan's Own Version of Emacs"; another common use is defining words that you often misspell in the same way, e.g., "thier" => "their" or "teh" => "the". See the information on the *auto-case-abbrev* variable.

There are two kinds of abbreviations: mode specific and global. If you define a Mode specific abbreviation in C mode, it will expand only in buffers that are in C mode. This is so you can have the same abbreviation expand to different things depending on your context. Global abbreviations expand regardless of the major mode of the buffer. The way it works is this: JOVE looks first in the mode specific table, and then in the global table. Whichever it finds it in first is the one that's used in the expansion. If it doesn't find the word it is left untouched. JOVE tries to expand words as they are typed, when you type a punctuation character or Space or Return. If you are in Auto Fill mode the expansion will be filled as if you typed it yourself.

16.220. wrap-search (variable)

If set, searches will "wrap around" the ends of the buffer instead of stopping at the bottom or top. The default is "off".

16.221. write-files-on-make (variable)

When set, all modified files will be written out before calling make when the *compile-it* command is executed. The default is "on".

16.222. write-word-abbrev-file (Not Bound)

This writes the currently defined abbreviations to a specified file. They can be read back in and automatically defined with *read-word-abbrev-file.*

16.223. write-file (C-X C-W)

This saves the current buffer to a specified file, and then makes that file the default file name for this buffer. If you specify a file that already exists you are asked to confirm over-writing it.

16.224. write-macros-to-file (Not Bound)

This writes the currently defined macros to a specified file. The macros can be read back in with *read-macros-from-file* so you can define macros and still use them in other instantiations of JOVE.

16.225. write-modified-files (C-X C-M)

This saves all the buffers that need saving. If you supply a numeric argument it asks for each buffer whether you really want to save it.

16.226. write-region (Not Bound)

This writes the text in the region to a specified file. If the file already exists you are asked to confirm over-writing it.

16.227. yank (C-Y)

This undoes the last kill command. That is, it inserts the killed text at point. When you do multiple kill commands in a row, they are merged so that yanking them back with "C–Y" yanks back all of them.

16.228. yank-pop (ESC Y)

This yanks back previous killed text. JOVE has a kill ring on which the last 10 kills are stored. *Yank* yanks a copy of the text at the front of the ring. If you want one of the last ten kills you use "ESC Y" which rotates the ring so another different entry is now at the front. You can use "ESC Y" only immediately following a "C-Y" or another "ESC Y". If you supply a negative numeric argument the ring is rotated the other way. If you use this command enough times in a row you will eventually get back to where you started. Experiment with this. It’s extremely useful.