```
EEEEEEEEEE      MM        MM       AAAAAA         CCCCCCCC       SSSSSSSS
EEEEEEEEEE      MM        MM       AAAAAA         CCCCCCCC       SSSSSSSS
EE              MMMM    MMMM      AA    AA       CC             SS
EE              MMMM    MMMM      AA    AA       CC             SS
EE              MM  MM  MM        AA    AA       CC             SS
EE              MM  MM  MM        AA    AA       CC             SS
EEEEEEEE        MM      MM        AA    AA       CC               SSSSSS
EEEEEEEE        MM      MM        AA    AA       CC               SSSSSS
EE              MM      MM        AAAAAAAAAA     CC                     SS
EE              MM      MM        AAAAAAAAAA     CC                     SS
EE              MM      MM        AA    AA       CC                     SS
EE              MM      MM        AA    AA       CC                     SS
EEEEEEEEEE      MM      MM        AA    AA         CCCCCCCC     SSSSSSSS
EEEEEEEEEE      MM      MM        AA    AA         CCCCCCCC     SSSSSSSS


  GGGGGGGG      UU      UU        IIIIII         DDDDDDDD       EEEEEEEEEE
  GGGGGGGG      UU      UU        IIIIII         DDDDDDDD       EEEEEEEEEE
GG              UU      UU          II           DD    DD       EE
GG              UU      UU          II           DD    DD       EE
GG              UU      UU          II           DD    DD       EE
GG              UU      UU          II           DD    DD       EE
GG              UU      UU          II           DD    DD       EEEEEEEE
GG              UU      UU          II           DD    DD       EEEEEEEE
GG  GGGGGG      UU      UU          II           DD    DD       EE
GG  GGGGGG      UU      UU          II           DD    DD       EE
GG      GG      UU      UU          II           DD    DD       EE
GG      GG      UU      UU          II           DD    DD       EE
  GGGGGG        UUUUUUUUUU        IIIIII         DDDDDDDD       EEEEEEEEEE
  GGGGGG        UUUUUUUUUU        IIIIII         DDDDDDDD       EEEEEEEEEE


 *START* User ALLEN  Job  EMACS Seq. 4593 Date 17-May-79 12:30:38
Monitor 1031 Great Pumpkin, TOPS-20 Monitor *START*
File: PS:<EMACS>EMACS.GUIDE.131 Created:  2-Jan-79 15:19:50
        Printed: 17-May-79 12:53:05
QUEUE Switches: /COPIES:10
       /SPACING:0 /LIMIT:2380 /FORMS:NARROW
```

CONTENTS

I INTRODUCTION

This document was prepared from the internal documentation contained within the INFO subpackage of EMACS. All of this information is available on-line within EMACS. It is assumed that you have already used the EMACS self-tutorial by giving the TEACH-EMACS command. If you have not done so, it is recommended that you do.

II What You See on the Screen

EMACS divides the screen into several areas, which contain different sorts of information. The biggest area, of course, is the one in which you usually see the text you are editing. The terminal's blinker usually appears in the middle of the text, showing the position of "point", the location at which requested changes will be made. While the cursor appears to point AT a character, point should be thought of as BETWEEN two characters; it points BEFORE the character that the blinker appears on top of. Terminals have only one blinker, and when type out is in progress it must appear where the typing is being done. At those times, unfortunately, there is no way to tell where point is except by memory or dead reckoning.

The top lines of the screen are usually available for text but are sometimes pre-preempted by an "error message", which says that some command you gave was illegal or used improperly, or by printed output produced by a command (such as, the description of a command printed by Meta-?). The error message or printed output will go away and the text behind it will be revealed again as soon as you type a command. If you just wish to make the error message go away, you can type a space.

A few lines at the bottom of the screen compose what is called "the echo area". It is the place where INFO types out the commands that you give it. EMACS does not normally echo the commands it is given, but if you pause for more than a second in the middle of a multi-character command then the whole command (including what you have typed so far) will be echoed. This behavior is designed to give confident users optimum response, while giving nervous users information on what they are doing.

EMACS also uses the echo area for reading and displaying the arguments for some commands, such as searches, and for printing certain information in response to certain commands such as M-=.

The line above the echo area is known as the "More line" or the "Mode line". It is the line that usually starts with "EMACS (something)". Its purpose is to tell the user at all times what is going on in the EMACS - primarily, to state any reasons why commands will not be interpreted in the standard way. The Mode line is very important, and if you are surprised by how EMACS reacts to your commands you should look there for enlightenment.

III How to Interpret the Information in the Mode Line

The line above the echo area is known as the "More line" or the "Mode line". It is the line that usually starts with "EMACS (something)". Its purpose is to tell the user at all times what is going on in the EMACS - primarily, to state any reasons why commands will not be interpreted in the standard way. The Mode line is very important, and if you are surprised by how EMACS reacts to your commands you should look there for enlightenment.

The normal situation is that characters typed in will be interpreted as EMACS commands. When this is so, you are at "top level", and the Mode line will look like

EMACS type (major minor) bfr: file --pos--

"type" will usually not really be there. When it is there, it indicates that the EMACS job you are using is not an ordinary one, in that it is acting as the servant of some other job. A type of "LEDIT" indicates an EMACS serving one or more LISPs, while a type of "MAILT" indicates an EMACS serving as a :MAIL "edit escape".

"major" will always be the name of the "major mode" you are in. At any time, EMACS is in one and only one of its possible major modes. The major modes available include Text Mode, Fundamental Mode (which EMACS starts out in), LISP Mode, and others. Sometimes the name of the major mode will be followed immediately with another name inside square-brackets ("[-]"). This name is called the "submode". The submode indicates that you are "inside" of a command which causes your editing commands to be changed temporarily, but does not change WHAT text you are editing. You can get out of any submode with C-C C-C.

"minor" is a list of some of the minor modes which are turned on at the moment. "Fill" means that Auto Fill mode is on. "Save" means that Auto-saving is on. "Save(off)" means that Auto-saving is on in general but momentarily turned off (it was overridden the last time a file was selected). "Atom" means that Atom Word mode is on.

"bfr" is the name of the currently selected buffer - see the section "Buffers". Normally, this will be the buffer "Main", which is the one buffer that exists when EMACS is started. Note that when we speak of "the buffer", we mean the one that is selected - but unless you explicitly ask to make new ones the initial buffer "Main" will be the only buffer there is.

"file" is the name of the file that you are editing. It is the last file that was selected in the buffer you are in. If "(RO)" (for "read only") appears after the filename, it means that changes you make will not be stored permanently on disk unless you say so explicitly (with a C-X C-S or C-X C-W command.

"Pos" tells you whether there is additional text above the top of the screen, or below the bottom. If your file is small and it is all on the screen, --pos-- is omitted. Otherwise, it is --TOP--, --BOT--, or --nn%-- where nn is the percentage of the file above the top of the screen. Sometimes you will see --MORE-- instead. This implies that you can see more of what was being displayed by typing a space. Inside some "type-out" commands, many of which have "List" or "View" in their names, typing anything other than a space when --MORE-- is visible will "flush" the additional display. This is what to do if you don't want to see the rest. "FLUSHED" will be printed after the --MORE--, and the character you typed will be obeyed (unless it was a <Delete>. A <Delete> will just be ignored).

So much for what the Mode line says at top level. When the mode line doesn't start with "EMACS (", and doesn't look anything like the breakdown given above, then EMACS is not at top level, and your typing will not be understood in the usual way. You must not naively assume that what will accomplish a certain thing at top level will still work.

For example, many commands read in arguments. While the argument to a command is being read, typed in characters will be interpreted as part of the argument instead of as commands. At such times, the mode line may contain text describing the command that is reading the arguments. If you invoked the command deliberately, you should have had some idea of what to do next. If you don't know what to do, you can type "?", which MOST commands will respond to with a description of what alternatives are available. Alternatively, you can "get out" of most commands and back to top level by typing Control-G. There are a few times when C-G won't work - they are when the Mode line is enclosed in parentheses or in brackets. These special cases are now described.

When the Mode line says something like "[command name - ^R]", It means that some command, invoked form EMACS, has in turn invoked EMACS (recursively), because as part of its processing it wants you to edit some text. That text might be unrelated to the file you are editing (which will be obvious from its appearance) and of special relevance to the particular command; then the mode line serves to inform you that you should edit the text so as to control the command as desired. In some commands, the text being edited might be your file itself. In such cases the mode line is to remind you that, although things otherwise look like top level, you are really still inside the command; it hasn't "finished" its work yet, and you should eventually return to it. In any case, if the mode line says "[...^R]" You can get out of the inner EMACS invocation with the command Control-<ESC> or C-M-C (C-C C-C). It

will get you back either into the command, or out of the command and back to top level, depending on the particular command.

When the text in the Mode line is surrounded by parentheses, it means that you are inside a "Minibuffer". A minibuffer is a special case of the recursive EMACS - it means that some command is requesting input. However, it gives extra information, because all minibuffers are handled alike in certain ways. You can get out of a minibuffer with a Control-G if it is empty, or with two Control-G's if it is not (the first Control-G clears out the text in it, and the second gets out). Get in the habit of typing C-G when EMACS doesn't seem to be reacting normally or when you are not sure what you have typed.

IV Basic Editing Commands

To insert printing characters into the text you are editing, just type them. When EMACS is in its normal state, they will be stuck into the buffer at the cursor, which will move forward. Any characters after the cursor will move forward too. If the cursor is in between a FOO and a BAR, typing XX will give you FOOXXBAR.

The most important way to make a correction is <Delete>. <Delete> 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. You can rub out a line boundary by typing <delete> when the cursor is at the front of a line. If you have just inserted a character, <Delete> will remove that very character, to undo the insertion.

To end a line and start typing a new one, just type "Return" (generally known as "CR"). You can also type Return to break an existing line into two. A <Delete> after a Return will undo it.

If you add too many characters to one line, without breaking it with a Return, the line will grow to occupy two (or more) lines on the screen, with a "!" at the extreme right margin of all but the last of them. The "!" says that the following screen line is not really a distinct line in the file, but just the "continuation" of a line too long to fit the screen.

To do more than this, you have to know how to move the cursor around. Here are some of the characters provided for doing that.

C-A moves to the beginning of the line.

C-E moves to the end of the line.

C-F moves forward over one character. If you do C-F at the end of a line, you will see that a line-boundary is really made up of two characters. They are a CR and a LF. Even though it takes only one command to insert or delete a line boundary, it is still two characters being inserted or deleted.

C-B moves backward over one character.

C-N moves down one line, vertically. If you start in the middle of one line, you end in the middle of the next.
C-N on the last line of the buffer will create a new line.

C-P moves up one line, vertically.

C-L clears the screen and reprints everything.

C-T exchanges two characters (the ones before and after point).

M-< moves to the top of your text.

M-> moves to the end of your text.

With these commands, you can do anything. The other commands just make things easier. But to keep any text permanently you must put it in a file. You do that by choosing a filename, such as FOO, and typing C-X C-E FOO<cr>. This "visits" the file FOO on your working directory) so that its contents appear on the screen for editing. You can make changes, and then type C-X C-S to make the permanent and actually change the file FOO. Until then, the changes were only inside your EMACS, and the file FOO was not really changed. If the file FOO doesn't exist, and you want to create it, that's no problem; just visit it as if it did exist. When you save your text with C-X C-S the file will be created.

One thing you should know is that it is much more efficient to insert text at the end of a line than in the middle. So if you want to stick a new line before an existing one, it is better to make a blank line there first and then type the text into it, rather than just go to the beginning, insert the text, and then insert a line boundary. It will also be clearer what is going on while you are in the middle. To make a blank line, you could type Return and then two C-B's. There is an abbreviation: C-O. So instead of typing FOO Return, type C-O FOO. If you want to insert many lines, you should type many C-O's at the beginning (or you can give C-O an "argument" to tell it how many blank lines to make. You can then insert the lines of text, and you will notice that Return behaves strangely: it "uses up" the blank lines instead of pushing them down. This is so that EMACS can save itself the trouble of moving the non-blank lines that follow them. If you don't use up all the blank lines, you can type C-X C-O (^R Delete Blank Lines) to get rid of all but one of them (or all of them, if you do it on the last nonblank line).

V Giving Numeric Arguments to EMACS Commands

Any EMACS ^R 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 (move forward one character) will move forward ten characters. With the C-F command, no argument is equivalent to an argument of one.

Some commands care only about whether there is an argument, and not about its value; for example, the command M-G with no arguments fills text, but with an argument justifies the text as well.

Some commands use the value of the argument, but do something peculiar when there is no argument. For example, the C-K command with an argument <n> will kill <n> lines and the line separators that follow them. But C-K with no argument is special; it kills the text up to the next line separator, or, if there is none, the separator.

On terminals with Meta keys, entering any argument is easy. Simply type the digits of the argument, with the Control or Meta key held down (or both). The argument must precede the command it is intended for. To give a negative argument, type a "-", again with the Control or Meta key held down, before the digits.

On other terminals, an argument can be entered by typing C-U or <ESC> followed by an optional minus sign, and the digits. The next non-digit is assumed to be the command for which the argument was intended. Note that while C-U will work the same way on all terminals, <ESC> will not: its function on non-Meta terminals is produced by redefining all the Meta-digits to gobble all following digits, and that redefinition would be undesirable on Meta terminals where M-3 1 means "insert three 1's".

C-U (but not <ESC>!) followed by a non-digit other than "-" has the special meaning of "multiply by four". Such a C-U multiplies the argument for the next command by four. Two such C-U's will multiply it by sixteen. Thus, C-U 5 C-U C-F would move forward 5 * 4 characters. C-U C-U C-F would move forward sixteen characters. It is a good way to move forward "fast", since it moves about 1/4 of a line on most terminals. Other useful combinations are C-U C-N, C-U C-U C-N (move down a good fraction of a screen), C-U C-U C-O (make "a lot" of blank lines), and C-U C-K (kill four lines).

With commands like M-G that care only whether there is any argument, C-U by itself serves as a flag. Since the value of the

argument is ignored, a value of four is as good as anything. Because of the convenience of C-U, some commands check for the specific argument values 4 and 16 that result from one or two C-U's. Thus, C-Space sets a mark if given no argument, but with an argument of 4 it pops the mark into point, while with an argument of 16 it pops and discards the mark.

Users of non-Meta keyboards may prefer to operate in "Autoarg" mode, in which digits preceding an ordinary inserting character are themselves inserted, but digits preceding an <ESC> or control character serve as an argument to it and are not inserted. To turn on this mode, use MM Alter Options to set the variable Autoarg Mode nonzero. Autoargument digits will echo at the bottom of the screen; the first nondigit will cause them to be inserted or will use them as an argument. To insert some digits and nothing else, you must follow them with a space and then rub out the space. C-G will cancel the digits, while <Delete> will insert them all and then rub out the last.

VI The Mark and the Region

C-@	Set the mark where point is.
C-Space	The same.
C-X C-X	Interchange mark and point.
M-@	Set mark after end of next word.
C-M-@	Set mark after end of next s-expression.
C-<	Set mark at beginning of buffer.
C->	Set mark at end of buffer.
M-H	Put region around current paragraph.
C-M-H	Put region around current DEFUN.
C-X C-P	Put region around current page.
C-X H	Put region around entire buffer.

In general, a command which processes an arbitrary part of the buffer must be able to be told where to start and where to stop. In EMACS, such commands start at the pointer and end at a place called the "Mark". This range of text is called "the Region". For example, if you wish to convert part of the buffer to all upper-case with the C-X C-U command, you could first set go to the beginning of the text to be capitalized, put the mark there, move to the end, and then issue the C-X C-U command. Or, you could set the mark at the end of the text, move to the beginning, and then issue the C-X C-U. C-X C-U's full name is ^R Uppercase Region, signifying that the Region, or everything between point and the mark, is capitalized.

The most common way to set the mark is with the C-@ command or the C-Space command. They set the mark where the pointer is. Then you can move the pointer away, leaving the mark behind. It isn't actually possible to type C-Space on non-Meta keyboards. Yet the command works anyway! This is because trying to type a Control-Space on a non-Meta keyboard will actually send a C-@, which is an equivalent command. A few keyboards will just send a space. If you have one of them, you suffer, or customize your EMACS.

If you want to reassure yourself that the mark is where you think it is, you can use the command C-X C-X (^R Exchange Point and Mark) which puts the mark where point was and point where the mark was. Thus, the previous location of the mark is shown, but the Region specified is not changed. C-X C-X is also useful when you are satisfied with the location of point but want to move the other end of the Region; just do C-X C-X to put point at that end and then you can adjust it. The end of the region which is at point can be moved, while the end which is at the mark stays fixed.

There are commands for placing the mark on the other side of a certain object such as a word or a list, without actually having to go there. M-@ puts the mark at the end of the next word, while C-M-@ puts it at the end of the next S-expression. C-> puts the mark at the end of the buffer, while C-< puts it at the beginning. These characters allow you to save a little typing, sometimes.

Other commands set both point and mark, to delimit an object in the buffer. M-H puts point at the beginning of the paragraph it was inside of (or before), and puts the mark at the end. M-H does all that's necessary if you wish to indent, case-convert, or kill a whole paragraph. C-M-H similarly puts point before and the mark after the current or next top-level S-expression (typically, a DEFUN). C-X C-P puts point before the current page (or the next or previous, according to the argument), and mark at the end. The mark goes after the terminating page delimiter (to include it), while point goes after the preceding page delimiter (to exclude it). Finally, C-X H delimits the entire buffer by putting point at the beginning and the mark at the end.

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, 16 previous locations of the mark are remembered. Most commands that set the mark push the old mark onto this stack. To revisit a marked location, use the C-@ or C-Space command with an argument, as in C-U C-@. This puts point where the mark was, like C-X C-X, but whereas C-X C-X puts the mark where the point was, C-U C-@ pops the previous mark off the stack. Thus, successive C-U C-@'s will revisit previous marks. The stack is actually cyclic, so that enough C-U C-@'s will return to the starting Point.

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 M-<, which moves to the beginning of the buffer. It sets the mark first, so that you can use C-U C-@ later to go back to where you were. Searches sometimes set the mark - it depends on how far they move. Because of this uncertainty, searches type out "" if they set the mark. The normal situation is that searches leave the mark behind if they move at least 500 characters, but the user can change that value since it is kept in the variable Auto Push Point Option. By setting it to 0, the user can make all searches set the mark. By setting it to a very large number such as ten million, he can prevent all searches from setting the mark.

VII Deletion and Killing

C-D	Delete next character.
<Delete>	Delete previous character.
C-K	Kill rest of line or one or more lines.
C-W	Kill region (ie, from point to mark).

EMACS has several commands that erase variously sized blocks of text from the buffer. Most of them save the erased text so that it can be restored if the user changes his mind, or moved or copied to other parts of the file. 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 commands which Delete instead of Killing are the single-character delete commands C-D and <Delete>, and those commands that delete only spaces or CRLFs. Commands that can destroy significant amounts of nontrivial data generally kill. The commands' names and individual descriptions use the words "Kill" and "Delete" to say which they do.

The simplest Kill command is the C-K command (^R Kill Line). If given at the beginning of a line, it kills all the text on the line, leaving it blank. If given on a blank line, the blank 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 the pointer up to the end of the line, unless it is at the end of a line, in which case the CRLF separating the line from the following one is killed, thus merging the next line into the current one.

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

If C-K is given a positive argument, it kills that many lines, and the CRLFs that end them (however, text on the current line before the pointer is spared). With a negative argument, -5 for example, all text before the pointer on the current line, and all of the five preceding lines, are killed.

The most basic Delete commands are C-D and <Delete>. C-D deletes the character after the cursor - the one the cursor is "on top of" or "underneath". The cursor doesn't move. <Delete> deletes the character before the cursor, and moves the cursor back over it. If you are inserting text and notice a mistake in one of the last few characters, just <Delete> back past it and type the text over again from that point.

A Kill command which is very general is C-W (^R Kill Region), which kills everything between point and the mark. With this command, you can kill any contiguous characters, if you set the mark at one end of them and go to the other end, first. A command ^R Delete Region used to exist, but it was too dangerous.

VIII Un-Killing

C-Y	Get back (re-insert) last killed text.
M-Y	Replace re-inserted killed text with the previously killed text.
M-W	Save region as last killed text without killing.
C-M-W	Append next kill to last batch of killed text.

Killed text is pushed onto a list that remembers the last 8 blocks of text that were killed. The command C-Y (^R Un-kill) inserts into the buffer the most recent block of text in the list. C-Y leaves the cursor at the end of the text that it un-kills, and puts the mark before it. Thus, if you change your mind again, a single C-W will kill what the C-Y got.

If you wish to copy a block of text, you might want to use M-W, which copies the region into the killed text list without removing it from the buffer. This is approximately equivalent to C-W followed by C-Y, but M-W will not mark the file the text comes from as "changed". This may be convenient when copying text into another file.

Normally, each kill command pushes a new block onto the killed text list. However, when two or more successive kill commands are given, all the text they kill is combined into one block of text on the list - combined in the same order that they were in before they were killed. Thus, a single C-Y command will undo all of the killing (Thus we join TV in leading people to kill thoughtlessly). If a kill command is separated from the last kill command by other commands, it will start a new block on the list unless you tell it not to by saying C-M-W (^R Append Next Kill) in front of it. The C-M-W tells the following command, if it is a kill command, to append the text it kills to the last killed text, instead of pushing it separately.

If you wish to recover text that was killed some time ago (was not the most recent victim), you need the Meta-Y command. The M-Y command should be used only after a C-Y command or another M-Y. It "replaces" the text resurrected by the previous command, which is the most recent block of killed text, with the previous block of killed text. Thus, a C-Y and a M-Y would restore the next to the last block of killed text. The effect of the M-Y is permanent in that C-Y's following the M-Y will get the same text the M-Y got. The M-Y moves the most recent corpse to the end of the list, where it will be the first to be forgotten if more killing occurs, so that the previous corpse comes to the front of the list. If you attempt to use M-Y when the region is not identical to the most recent string of killed text, it will refuse to operate, since it would otherwise destroy the contents of the region unrecoverably.

Meta-Y is also useful for recovering from certain commands that can alter large portions of the buffer, such as M-G (^R Fill Region) and C-X C-U (^R Uppercase Region). Such commands save the region on the kill list before acting. After the command, do M-W followed by M-Y to replace the altered text with the previous text. Can you figure out why that works?

The procedure for finding text killed some time ago is simple: Just do a C-Y. If you see the text you want, you are done. Otherwise, do a M-Y. If that doesn't bring you what you are looking for, keep doing M-Y's until you find it. If after a while this starts to repeat, you have seen the whole kill-ring and the text you want is no longer in it. In this case, you have lost; at any point you can type a C-W to get rid of the un-killed text that you don't want.

If you do "too many" M-Y's, and go past the last remembered block of killed text (they aren't kept forever), you will get back to the most recent one again. Thus, if you accidentally hit one M-Y too many, you could recover by going all the way around. But an easier way is to do M-Y with an argument of -1. That moves backwards, to fresher corpses. One M-Y with -1 as argument undoes one M-Y with no argument (equivalent to +1).

If you know how many M-Y's it would take to find the text you want, then there is an easier alternative. C-Y with an argument greater than one restores the text the specified number of entries down on the list. Thus, C-U 2 C-Y is gets the next to the last block of killed text. It differs from C-Y M-Y in that C-U 2 C-Y does not move the blocks on the list.

IX MM Commands

While the most often useful EMACS commands are accessible via one or two characters, there are many commands for which the effort of remembering a cryptic name would not be worth while. Such commands have long descriptive names instead, and are accessed in a special way. They are known as "MM Commands", as opposed to the one or two character commands which are called "^R Commands" (unfortunately, only context can determine which of those the word "command" means). One example of an MM command is MM List Commands, which types out a list giving the full names of all the MM commands, together with brief descriptions of what they do.

The easiest way to invoke an MM command is to give the command Meta-X or <ESC> X. This command should be followed by the name of the MM command to be invoked, terminated by a Return. For example, the command List Commands would be specified by Meta-X List Commands Return.

You can abbreviate the name of the command, as long as the abbreviation is unambiguous. If the name you specify is ambiguous or impossible, you will get an error message.

When you type the Meta-X, you will notice that the cursor moves down to the echo area at the bottom of the screen, and "MM " appears there. This is to remind you that you should next type the name of an MM command. This sort of input is known as "reading a line in the echo area", or as "& Read Line input". The string (in this case, "MM") which appears in the echo area to support your memory is called a "prompt". Many commands read arguments in this fashion. The prompt always tells you what sort of argument is required and what it is going to be used for.

While you are typing the command name at the bottom of the screen, you can use <Delete> to cancel one character, C-U or C-D to cancel all the input you have typed and start over, and C-G to get back to top level (cancel the command name AND the Meta-X). If the command name is empty and you type a <delete>, that too cancels the Meta-X. Again, this is generally true whenever anything reads a line in the echo area.

Some MM commands require "string arguments" or "suffix arguments". For those commands, the command name should be terminated with a single <ESC>, after which should come the arguments, separated by <ESC>s. After the last argument, type a Return to cause the command to be executed. For example, the command MM Describe prints the full documentation of an MM command (or other EMACS subroutine) whose name

must be given as a string argument. An example of using it would be Meta-X Describe<ESC>Apropos Return, which would print the full description of MM Apropos.

Some MM commands can use numeric prefix arguments. Simply give the Meta-X command an argument and Meta-X will pass it along to the MM command. The argument will appear before the "MM" in the prompt, so you don't forget it was typed.

X Arcane Information about MM Commands

MM commands are called that, even though you usually run them by typing Meta-X and never say "MM" yourself, because "MM" is the TECO expression which looks up a command name to find the associated program, and runs that program. Thus, the TECO expression

MM Apropos<ESC>Word<ESC>

means to run the Apropos command with the argument "word". You could type this expression into a minibuffer and get the same results as you would get from Meta-X Apropos<ESC>Word Return. In fact, before there was Meta-X, that's what people did.

"MM" actually tells TECO to run the program named "M". This program uses a string argument as the command name and looks it up. Processing named commands isn't built into TECO; it is implemented by the program TECO calls "M". That's why "MM" is called that and not "Run" or "F

".

MM commands can also use one or two "prefix arguments" or "numeric arguments". These are numbers (actually, TECO expressions) which go before the "MM". Meta-X can only give the MM command one argument. If you want to give it two, you must type it in using the minibuffer.

A command can be given two numeric arguments, separated by a comma. Either one can be omitted. If the one before the comma is omitted, the comma should also be omitted. Note that the meaning of an argument is NOT determined by whether there is another argument before it, but by whether it precedes a comma. Thus, "1," omits the second argument, while "1" by itself omits the first. Of course, when a command is only interested in one argument, it looks at the second, so that you do not need to (and should not) type a comma. For example, the command MM Auto Save can meaningfully be given two arguments. The first, if present, says how often to auto-save, while the second says how many past savings should be remembered. Either argument can be omitted, in which case the associated quantity is not changed. Thus, "200,MM Auto Save" specifies saving every 200 characters, but not change the number of saves that are remembered, while "4MM Auto Save" specifies that the last four saves should be kept, but does not alter the period between saves.

Actually, the one and two character "^R Commands" all have names, too, and could be run with MM using them. MM commands are simply functions intended primarily to be called explicitly by the user. There are other functions intended to be accessed very easily; EMACS causes each defined ^R Command to invoke one such function. Their names

usually start with "^R" (Uparrow R) so that they will not interfere with attempts to abbreviate the names of other commands. In fact, as functions go, there is little special about them. Any function (except a few of the ones that read string arguments) can be put on any ^R Command for easy access. Usually, whenever a ^R Command is described, the name of the function which is its definition is given in parentheses. Knowing the function's name is not necessary for normal use, but it becomes important when you start customizing EMACS.

Yet other MM commands exist which are intended to be called as subroutines by other EMACS commands. Commands intended primarily as subroutines usually have names starting with "&", as in "& Read Line", the subroutine which reads a line in the echo area. Although most subroutines have such names, any MM command can be called as a subroutine.

XI EMACS Documentation Commands

First of all, you should know about the documentation files for EMACS,<DOCUMENTATION>EMACS.GUIDE and <DOCUMENTATION>EMACS.CHART and <DOCUMENTATION>EMACS.NCHART. They contain complete printouts of EMACS's built in documentation. The first has a brief description of all the ^R Commands, the second has a brief list of the most useful commands ordered by function while the last contains the full description of each ^R Command and each named function. EMACS.CHART and EMACS.NCHART are good to post on the wall near your terminal whereas <DOCUMENTATION>EMACS.GUIDE are good to have in your library. You can list yourself copies of these files if you want.

Aside from those documentation files and this file, the INFO documentation, EMACS has commands for telling you what a command does, how to do a particular thing, and what commands are available. In addition, there is an EMACS-wide Help character which can be used at all times to ask for help. This character is typed as C-_ and as C-_ H on non-Meta keyboards. At top level, this character will offer several options for asking for help. Inside a command, the help character will tell you about that command, and what you are supposed to be typing while within it. Unlike "?", which cannot always be used to request help since sometimes it must be treated as a text character, the Help character will ALWAYS ask for help, and it is always safe: you can type it in the middle of absolutely anything, whether at top level or not.

If you want to know what a particular one or two character ("^R") Command does, type the command Meta-?, followed by that command. In particular, Meta-? Control-X <subcommand> describes what the given C-X subcommand does. Note that Meta-? must be typed as <ESC> followed by ? on non-Meta keyboards. This operation is also available as Help C, normally.

If you want to know what a particular function ("MM command") does, use the MM command "Describe", giving the particular function's name as an argument. Or, type Help D, and the command name. For example, Meta-X Describe<ESC>LISP Mode Return would give you the full documentation of MM LISP Mode.

You can see the names of all the MM commands with MM List Commands. If you are interested in a certain topic, MM Apropos will show you just the functions that relate to that topic. Actually, it shows you all functions whose names contain your topic as a substring. If a function is accessible as a short ("^R") Command, MM Apropos will tell you that also. Help A followed by a substring also does this. Thus, Meta-X

Apropos<ESC>File Return (no spaces!) would list all the functions whose names contain the word "File". You can then run any of those functions using Meta-X. Some will also mention one or two character ways of running them.

If you just wish to know which, if any, short command, runs a specific function, give the command MM Where Is with the name of the function as a string argument: Meta-X Where Is<ESC>^R Describe Return will tell you that the function named ^R Describe can be invoked using Meta-?.

MM List Commands lists only the functions that are primarily provided to be run by name with MM, using a Meta-X, for example. Functions whose expected major use is to be placed on a ^R Command, and functions intended primarily to be called by other functions, are not mentioned. To see a list of them, use MM List ^R Commands or MM List Subroutines.

XII What To Do When EMACS Is Hung

There are three general "quitting" procedures in EMACS: C-G, MM Top Level, and restarting. The last is a drastic one for use when C-G does not work.

C-G is an interrupt character which is effective most of the time. If you have started to type a command but not finished, C-G will cancel it. If you have specified a numeric argument, C-G will cancel it. If a command is running, C-G will stop it and return to the ^R Editing level from which the command was issued. Inside the recursive ^R Editing levels established by many commands, C-G will often get you out one level. A similar "step-at-a-time" quitting process occurs in minibuffers: if the minibuffer is empty, C-G quits out of it, but if there is text present then the first C-G just kills the text, so that the second C-G will quit out.

C-G always takes effect between the TECO commands which make up an EMACS program, never in the middle of one (a few long commands allow quitting at any time). This is to prevent TECO from being destroyed by quitting. However, sometimes EMACS will be hung inside a TECO command (if, for example, TECO is waiting in the middle of reading in a file). In such a case, C-G will not work, but the more drastic procedure of restarting TECO may work. Type C-C and then START to restart it. While restarting TECO in this way is usually safe (especially at times when TECO is doing I/O), there are certain times at which it will cause the TECO data structures to be inconsistent, so do not try it unless other measures have failed.

Restarting TECO is also a good thing to do if you have moved from one terminal to another which is of a different type. Every time TECO is restarted, it finds out from the system what type of terminal you are using. If you stopped TECO when it was trying to read from the terminal, or by typing an EMACS command to return to the superior, then this is sure to be safe.

There is one other quit mechanism which serves a somewhat different purpose. MM Top Level is used, in a recursive ^R Editing level, to exit all the way to top level (top level is where the mode line says "EMACS (major ...) ..." with no submode). In most cases, typing C-G's until they ceased to have any effect would get you to the same place, but there are some recursive ^R Editing states which catch C-G's themselves. MM Top Level is the only way to quit out of all of those.

XIII EMACS's Major Modes

When EMACS starts up, it is in what is called "Fundamental Mode", which means that the short (^R) Commands are defined so as to be convenient in general. For editing any specific type of text, such as LISP code or papers, you can tell EMACS to change the meanings of a few commands to become more specifically adapted to the task. This is done by switching from Fundamental Mode to one of the other major modes. Most commands remain unchanged; the ones which usually change are Tab, <Delete>, and Linefeed. In addition, the commands which handle comments use the mode to determine how comments are to be delimited.

Selecting a new major mode is done with an MM command. Each major mode is the name of the MM command which selects it. Thus, you can enter LISP mode by executing MM LISP (short for MM LISP Mode). The major modes are mutually exclusive - you can be in only one major mode at a time. When at top level, EMACS always says in the Mode line which one you are in.

LISP Mode causes Tab to run the function ^R Indent for LISP, which causes the current line to be indented according to its depth in parentheses. Linefeed, as usual, does a CR and a Tab, so it will move to the next line and indent it. <Delete> is also redefined, so that rubbing out a tab will cause it to turn into Spaces. Thus, the fact that Tab uses Tabs as well as Spaces to indent lines is invisible when you try to rub the indentation out. Comments start with ";". When in LISP mode, the action of the word-motion commands is affected by whether you are in Atom Word Mode or not - see the section on minor modes.

Text Mode causes Tab to run the function ^R Tab to Tab Stop, which allows you to set any tab stops with MM Edit Tab Stops; however, in text mode, Auto Fill does not normally indent new lines that it creates. Comments are declared not to exist. Finally, the meaning of "word" as understood by the word-motion commands is changed so that "." is not considered part of a word, while "'" is.

MIDAS Mode is for editing assembler language programs - any assembler language. It is very close to Fundamental Mode, but a few special-purpose commands are provided: C-M-A, C-M-E, C-M-N, C-M-P, and C-M-D.

Tabular Text Mode is mostly the same as Text Mode, except that auto-fill will indent new lines. If you use this mode, you will probably want to call MM Edit Tab Stops to specify the Tab settings.

XIV EMACS's Minor Modes

Minor modes are options which you can use or not. They are all independent of each other and of the selected major mode. Each minor mode is the name of the MM command that can be used to turn it on or off. Usually, no argument turns the mode on, while an argument of zero turns it off.

Auto Fill Mode allows you to type text endlessly without worrying about the width or your screen. CR's will be inserted where appropriate to prevent lines from becoming too long. The column at which lines are broken defaults to ten columns before the right margin, but you can set it explicitly. C-X F (^R Set Fill Column) sets the column for breaking lines to the column point is at. The fill column is stored in the variable Q<ESC>Fill Column<ESC>.

Auto Save Mode protects you against system crashes by periodically saving the file you are visiting. Whenever you visit a file, auto saving will be enable if Q<ESC>Auto Save Default<ESC> is nonzero; in addition, the command MM Auto Save allows you to turn auto saving on or off in a given buffer at any time.

Atom Word Mode causes the word-moving commands, in LISP mode, to move over LISP atoms instead of words. Some people like this, and others don't. In any case, the S-expression motion commands can be used to move over atoms. If you like to use segmented atom names like FOOBAR-READ-IN-NEXT-INPUT-SOURCE-TO-READ, then you might prefer not to use Atom Word Mode, so that you can use M-F to move over just part of the atom, or C-M-F to move over the whole atom.

Overwrite Mode causes ordinary printing characters to replace existing text instead of shoving it over. It is good for editing pictures. For example, if the point is in front of the B in FOOBAR, then in Overwrite Mode typing a G would change it to FOOBAR, instead of making it FOOBAR as usual.

Word Abbrev Mode allows you to define abbreviations that automatically expand as you type them. For example, "wam" might expand to "word abbrev mode". The abbreviations may depend on the major (e.g. LISP, TEXT, ...) mode you are currently in. To use this, you must load the WORDAB library. To learn about this mode, see the menu below.

Some minor modes are actually controlled by variables. Setting the minor mode with a command just changes the variable. This means that you can turn the modes on or off with Alter Options, make their values local to a buffer or file, etc.

XV Named Variables in EMACS

EMACS and TECO allow variables with arbitrary names to be defined and given numbers or strings as values. EMACS uses many variables internally, but also has several whose purpose is to be set by the user to tell commands what to do. One example of such a variable is the Fill Column variable, which specifies the position of the right margin (in characters from the left margin) to be used by the fill and justify commands.

The easiest way for the beginner to set a named variable is to use the Alter Options command. This command shows you a list of selected variables which you are likely to want to change, together with their values, and lets you edit them with the normal editing commands. Don't make any changes in the names, though! Just change the values. When you are finished, type <ESC> to say so, and the changes will take effect. Until then, M-X Top Level will get you out with no ill effects.

However, Alter Options can be used only to set a variable which already exists, and is marked as an option. Some built-in commands refer to variables which do not exist in the initial environment. Such commands always use a default value if the variable does not exist. In these cases you must create the variable yourself if you wish to use it to alter the behavior of the command.

Variables can be made local to an individual file. The variable will then magically have its local value whenever you are looking at that file. When making variables local, you do not need to worry about whether the variables already exist, because they will be created automatically.

You can get a list of all variables, not just those you are likely to want to edit, by doing M-X List Variables. M-X Describe can be given a variable's name instead of a command's name; it will show the variable's value and its documentation, if it has any.

You can also set a variable with the TECO command <val>U<ESC><varname><ESC>, which you can give in the minibuffer or in your init file. This way is probably faster for the experienced user. However, there is a pitfall. Some variables, for the sake of efficiency, are not checked every time they are relevant; instead, commands are redefined based on their values. This command redefinition is not automatic, but is done by the subroutine & Process Options. EMACS calls this subroutine automatically when EMACS knows it is necessary (Alter Options calls it, and so do all MM commands which

change any of these variables), but otherwise you must call it yourself. In particular, init files are likely to need to call & Process Options. Here is a list of the variables which require that & Process Options be called:

Atom Word Mode
Autoarg Mode
Auto Fill Mode
FS CTLMTA
Lisp) Hack
Overwrite Mode

In addition, the mode line shows the settings of some of these variables. In order to cause the mode line to be updated, call the subroutine & Set Mode Line.

XVI Controlling the EMACS Display

C-L	Clear and redisplay screen, with point at specified place.
C-V	Scroll upwards (a screen or a few lines).
M-V	Scroll downwards.
M-R	Move point to given vertical position.

A file is rarely short enough to fit on the screen of a terminal all at once. If the whole buffer doesn't fit on the screen, EMACS shows a contiguous portion of it, containing the pointer. It will continue to show the same portion until the pointer moves outside of it; then EMACS will show a new portion centered around the new pointer. This is EMACS'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 basic display control command is C-L (^R New Window). In its simplest form, with no argument, it tells EMACS to display a portion of the buffer centered around where the pointer is currently located (actually, the pointer goes 35% of the way down from the top; this percentage can be set by the user by executing <pct>FS % CENTER<ESC> in a minibuffer). Normally, EMACS only recenters the pointer in the screen when it moves past the edge.

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. The pointer does not move with respect to the text; rather, the text and point move rigidly on the screen. If you decide you want to see more below the pointer this is the command to use. C-L with a negative argument puts point that many lines from the bottom of the window. For example, C-U -1 C-L puts point on the bottom line, and C-U -5 C-L puts it five lines from the bottom.

If you want to see a few more lines at the bottom of the screen and don't want to guess what argument to give to C-L, you can use the C-V command. C-V 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 will show you more lines at the top of the screen, as will Meta-V with a positive argument.

Often you want to read a long file sequentially. For this, the C-V command is ideal; it takes the last two lines at the bottom of the screen and puts them at the top, followed by a whole screenful of lines not visible before. The pointer is put after those two lines. Thus, each C-V shows the "next screenful", except for two lines of overlap to

provide continuity. To move backward, give C-V a negative argument, or use the M-V command which is like C-V but reverses the sign of its argument.

If ALL you want to do to a file is read it sequentially, you might want to use MM View File. Ask for a description of it with MM Describe if you are interested. For in-between situations, MM View Buffer might be useful.

C-L in all its forms changes the position of point on the screen, carrying the text with it. Another command moves point the same way but leaves the text fixed. It is called Meta-R. With no argument, it puts point at the center of the screen. An argument is used to specify the line to put it on, counting from the top if the argument is positive, or from the bottom if it is negative. Thus, Meta-0 Meta-R will put the cursor on the top line of the screen. Meta-R never causes any text to move on the screen; it causes point to move with respect to the screen and the text.

XVII Two Window Mode

C-X 2	Start showing two windows.
C-X 3	Show two windows but stay "in" the top one.
C-X 1	Show only one window again.
C-X O	Switch to the other window (when both are being shown).
C-X ^	Make this window bigger.
C-M-V	Scroll the other window.

Normally, EMACS is in "One-window mode", in which a single body of text is visible on the screen and can be edited. At times, one wants to have parts of two different files visible at once. For example, while adding to a program a use of an unfamiliar feature, one might wish to have the documentation of that feature visible. "Two-window mode" makes this possible.

The command C-X 2 (^R Two Windows) enters two-window mode. A line of dashes will appear across the middle of the screen, dividing the text display area into two halves. Window one, containing the same text as previously occupied the whole screen, fills the top half, while window two fills the bottom half. The pointer will move to window two. If this is your first entry to two-window mode, window two will contain a new buffer named W2. Otherwise, it will contain the same text it held the last time you looked at it. The mode line will now describe the buffer and file in window two. Unfortunately, it's almost impossible to provide a mode line for each window, so making the mode line apply to the window you are in is the best we can do.

You can now edit in window two as you wish, while window one remains visible. If you are finished editing or looking at the text in the window, C-X 1 (^R One Window) will return to one-window mode. Window one will expand to fill the whole screen, and window two will cease to be visible until the next C-X 2.

While you are in two window mode you can use the command C-X O to switch between the windows. After doing C-X 2 the cursor is in window two. Doing C-X O will move the cursor back to window one, to exactly where it was before the C-X 2. The difference between this and doing C-X 1 is that C-X O leaves window two visible on the screen. A second C-X O will move the cursor back into window two, just where it was before the first C-X O. And so on.

Often you will be editing one window while using the other just for reference. Then, the command C-M-V (^R Scroll Other Window) is very useful. It scrolls the other window without switching to it and

switching back. It scrolls just the way C-V does: with no arg, a whole screen up; with an argument, that many lines up (or down, for a negative argument).

When you are finished using two windows, the C-X 1 command will make window two vanish. It doesn't matter which window the cursor is in when you do the C-X 1; either way window two vanishes and window one remains.

The C-X 3 command is like C-X 2 but leaves the cursor in window one. That is, it makes window two appear at the bottom of the screen but leaves the cursor where it was. C-X 2 is equivalent to C-X 3 C-X O. C-X 3 is equivalent to C-X 2 C-X O, but C-X 3 is much faster.

Normally, the screen is divided evenly between the two windows. You can also redistribute the lines between the windows with the C-X ^ (^R Grow Window) command. It makes the currently selected window get one line bigger, or as many lines as is specified. If given a negative argument, the selected window gets smaller. The allocation of space to the windows is always remembered and changes only when you give a C-X ^ command.

You can view the same buffer in both windows. Give C-X 2 an argument as in C-U C-X 2 and you will go into window two viewing the same buffer as in window one. Although the same buffer appears in both windows, they have different cursors, so you can move around in window two while window one continues to show the same text. Then, having found in window two the place you wish to refer to, you can C-X O back to window one to make your changes. Finally you can do C-X 1 to make window two leave the screen. If you have set the variable Tags Find File nonzero, indicating that you normally use the C-X C-F command, then when you enter two window mode for the first time the same buffer will be viewed in both windows, rather than creating a buffer named W2. In this situation, the C-X 2 command acts like a C-U C-X 2 command.

Buffers can be selected independently in each window. The C-X B command selects a new buffer in whichever window the cursor is in. The other window's buffer does not change. When you do C-X 2 and window two appears it will show whatever buffer used to be visible in it when it was on the screen last. C-U C-X 2 makes window 2 select whichever buffer was selected in window 1.

While in one-window mode, you can still use C-X O, but the effect is slightly different. Window two does not appear, but whatever was being shown in it appears, in window one (the whole screen). Whatever buffer used to be in window one is stuck, invisibly, into window two. Another C-X O will reverse the effect of the first. For example, if window one shows buffer B and window two shows buffer W2 (the usual case), and only window one is visible, then after a C-X O window one will show buffer W2 and window two will show buffer B.

XVIII Visiting Files

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" the file. You can only edit a file when you are visiting it. Unless you use the multiple buffer and window features of EMACS, you can only be visiting one file at a time. The name of the file you are visiting in the currently selected buffer is visible in the Mode line when you are at top level, unless you aren't visiting any file.

The changes you make to a file you are visiting are not actually made in the file itself, however, but in a copy inside the EMACS. The file itself is not changed, and the editing is not permanent, until the file is "saved". You can save the file at any time by command. EMACS will normally save the file whenever you switch files, so you don't have to worry about forgetting to save before reading in another file over your edited text. In addition, for those who are afraid of system crashes, "Auto Save" mode is available. It performs a save at regular intervals automatically.

To visit a file, use the commands C-X C-E, C-X C-V, or C-X C-R (any one). Follow the command with the name of the file you wish to visit, terminated by a carriage return. If you can see a filename in the Mode line, then that name is the default, and any component of the filename which you don't specify will be taken from it. If you can't see a filename in the Mode line (because you are on a printing terminal, or because you hadn't visited any file yet), EMACS will print what the defaults are. You can abort the command by typing C-G, or cancel your type-in for the filename with C-U. If you do type a CR to finish the command, the new file will appear on the screen, and its name will show up in the Mode line.

If you alter one file and then visit another, EMACS may offer to save the old one. You should not type ahead after a file visiting command, because your type-ahead might answer an unexpected question in a way that you will regret.

When you wish to save the file and make your changes permanent, type C-X C-S (^R Save File). After the save is finished, C-X C-S will print "Written: <filenames>" in the echo area at the bottom of the screen. EMACS won't actually write out the file if the contents have not been changed since the last save or since the file was read in. So you don't have to try to remember that yourself; when in doubt, type a C-X C-S.

If you edit one file and then read in another without saving the first, the changes are lost. EMACS can protect you from this, if you wish. You have three options for such protection:
1) EMACS can allow you to clobber yourself. You must remember to save your file explicitly with C-X C-S when you have changed it.
2) EMACS can automatically save the file for you when you ask to read in another file.
3) EMACS can ask you, each time you deselect a file you have edited, whether to write it out.
You select one of these options each time you visit a file, by using either C-X C-R to get option 1, C-X C-E for option 2, and C-X C-V for option 3. The option selected controls what EMACS will do later, when you leave the file you are now visiting. It does not affect EMACS's decision to write out the old file, which was determined when you visited that file.

Thus, if you are SURE that you want to change a file, you should read it in with C-X C-E. If you are sure that you DON'T want to change a file, you should use C-X C-R. That way, any accidental changes you make will not be saved. If you aren't sure, you might want to use C-X C-V. Of course, you can use C-X C-R all the time if you prefer, but you must take the responsibility for remembering to C-X C-S. Note that the old file will be saved if IT had been read in with C-X C-E.

C-X C-E, as opposed to C-X C-R or C-X C-V, causes Auto Save to behave slightly differently. If you use Auto Save, you should be aware of this. Also, users of C-X C-E MUST read about MM Revert!

If EMACS is about to save a file automatically and discovers that the text is now a lot shorter than it used to be, it will tell you so and ask for your consent ("Y" or "N"). If you aren't sure what to answer (because you are surprised), type C-G to abort the visiting of a new file, and take a look around.

What if you want to create a file? Just visit it. If you save the "changes", the file will be created. If you visit a nonexistent file unintentionally (because you typed the wrong file name), just switch to the file you meant. If you didn't "change" the nonexistent file (ie, you left it empty the way you found it) no file will be created.

To look at a part of a file directory, use the C-X C-D command (^R Directory Display). With no argument, it shows you the file you are visiting, and related files. C-U C-X C-D reads a filespec from the terminal and shows you the directory based containing that filespec.

XIX What to do if you make some changes to a file and then regret them

If you read in a file with C-X C-E, make changes, and then regret having made them, then simply not typing C-X C-S isn't enough. You can't just type another C-X C-E intending to read in the old file, because that will first write back the very changes that you want to throw away.

What you should do in such a situation is MM Revert File<ESC><ESC>. It will read in the last saved version of the file. If you have been using Auto Save mode, the last Auto Save file will be read in, no matter where your auto-saving is going. MM Revert File<ESC> does not change point. If the file was only edited slightly, you will be at approximately the same piece of text after the Revert as before. If you have made drastic changes, the same value of point in the old file may address a totally different piece of text.

If you make such a mistake fairly often, you might want to make C-X C-V your habitual read-a-file command. Then EMACS will always ASK whether to write back a file when you switch to another.

XX Buffers: How to Switch between Several Bodies of Text

When talking of "the buffer", which contains the text you are editing, I 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.

Each buffer in EMACS has a single name, which normally doesn't change. A buffer's name can be any length, and can't conflict with any other kind of name. The name of the currently selected buffer is visible in the Mode line when you are at top level. A newly started EMACS has only one buffer, named "Main".

To create a new buffer, you need only choose a name for it (say, "FOO") and then do C-X B FOO<CR>, which is the command C-X B (Select Buffer) followed by the name. This will make a new, empty buffer and select it for editing. The buffer will start out in Fundamental mode, but you can change each buffer's major mode independently. A new buffer is not visiting any file, so any text you type into it will not be saved anywhere automatically.

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

You can also read a file into its own newly created buffer, all with one command: C-X C-F, followed by the filename. C-F stands for "Find", because if the specified file already resides in a buffer in your EMACS, that buffer will be 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 be by visiting them in an existing buffer.

C-X C-F normally visits the file a la C-X C-V, meaning ask about writing back if you visit some other file in that buffer, on the assumption that a user of C-X C-F doesn't intend to visit any other file in THAT buffer. However, Auto Save mode users might prefer to have the file visited a la C-X C-E so that auto saving works more nicely. To

bring that about, set the variable Q<ESC>Find File Inhibit Write<ESC> to zero.

To get a list of all the buffers that exist, do C-X C-B or do MM List Buffers. Each buffer's name, major mode, and visited filenames will be printed. A star at the beginning of a line indicates a buffer which contains changes that have not been saved. The number that appears before a buffer's name in a C-X C-B listing is that buffer's "buffer number". You can select a buffer by giving its number as a numeric argument to C-X B, which then will not need to read a string from the terminal.

If several buffers have stars, you should save some of them by doing MM Save All Files<ESC>. This will find all the buffers that need saving and ask about each one individually. Saving the buffers this way is much easier and more efficient than selecting each one and typing C-X C-S.

After using an EMACS for a while, it may fill up with buffers holding files that you no longer need to edit. If you have enough of them, you can reach a point where you can't read in any more files. So whenever it is convenient you should do MM Kill Some Buffers<ESC>, which will ask about each buffer individually. You can say Y or N to kill it or not. Or you can say ^R To take a look at it first. This does not actually select the buffer, as the mode line will show, but you can move around in it and look at things. When you have seen enough to make up your mind, exit the ^R With a C-C C-C and you will be asked the question again. If you say to kill a buffer that needs saving, you will be asked whether to save it.

You can kill the buffer FOO by doing C-X K FOO<cr>. You can kill the current buffer, a common thing to want to do if you use C-X C-F, by doing C-X K<cr>. If you kill the current buffer, in any way, EMACS will ask you which buffer to select instead. Saying just <cr> at that point will ask EMACS to choose one reasonably.

As well as the visited file and the major mode, a buffer can, if ordered to, remember many other things "locally" - independently of all other buffers. Any TECO Variable can be made local to a specific buffer by doing M.L<variable name><ESC>. Thus, if you want the comment column to be column 50 in one buffer, whereas you usually like 40, then in the one buffer you should do M.LComment Column<ESC>. Then, you can do 50U<ESC>Comment Column<ESC> in that buffer and all other buffers will not be affected. To get a list of the local variables in the current buffer, do MM List Locals<ESC><ESC>. Note that every buffer has a few local variables made automatically for internal use. The Local Modes feature makes use of these local variables, which are created when you visit the file and destroyed (made no longer local) when you visit some other file in the buffer. For information on how local variables work, and additional related features, see

MM Rename Buffer<ESC><new name><ESC> changes the name of the currently selected buffer. If <new name> is the null string, the first filename of the visited file is the name of the buffer.

XXI Local Variables in Files:

By putting "local mode specifications" in a file you can cause certain major or minor modes to be set, or certain character commands to be defined, whenever you are visiting it. For example, EMACS can select LISP Mode for that file, or it can turn on Auto Fill Mode, set up a special Comment Column, or put a special command on the character C-M-,. Local modes can specify the major mode, and the values of any set of named variables and ^R Command characters. Local modes apply only while the buffer containing the file is selected; they do not extend to other files loaded into other buffers.

The simplest kind of local mode specification sets only the major mode. You put the desired mode's name in between a pair of "-*-"'s, anywhere on the first nonblank line of the file. For example, if this line were the first nonblank line of this file, -*-Text-*- would enter Text Mode. In fact, a -*-Text-*- appears at the front of this file, but you won't normally see it since it is before the start of the first node. The -*- can appear on the first nonblank line after the edit history, if somebody insists on putting in an edit history.

To specify more that just the major mode, you must use a "local modes" list, which goes in the LAST page of the file (it is best to put it on a separate page). The local modes list starts with a line containing the string "Local Modes:". Whatever precedes that string on the line is the "prefix", and whatever follows it is the "suffix" (either or both may be null). From then on, each line should start with the prefix, end with the suffix, and in between contain one local mode specification in the form <varname>:<value>. To set a variable, make <varname> the name of the variable and <value> the desired value. If <value> is a numeral (which means no spaces!), the value will be a number; otherwise, it will be <value> as a string. To set a ^R Command character, make <varname> the name of the character as a q-register, such as ...^R, For C-M-Comma, and make <value> be a string of TECO commands which will return the desired value (this is so you can write M.MFoo<ESC> to define the character to run the command Foo). The local modes list is terminated by a line which contains "End:" instead of <varname>:<value>.

The major mode can be set by specifying a value for the variable "Mode" (don't try doing it this way except in a local modes list!). An MM command named MM Foo can be defined locally by putting in a local setting for the variable named "MM Foo".

Here is an example of a local modes list:

```
;;; Local Modes: :::
;;; Comment Column:0 :::
;;; Comment Start:;;;  :::
;;; ..^R/: m,m^R My Funny Meta-Slash<ESC> :::
;;; Mode:LISP :::
;;; End: :::
```

Note that the prefix is ";;; " and the suffix is " :::". Note also that the value specified for the Comment Start variable is ";;; ", which is the same as the prefix (convenient, eh?).

If you have a local modes list, the last page of the file must be no more than 10000 characters long or it will not be recognized. This is because EMACS finds the local modes list by scanning back only 10000 characters from the end of the file for the last ^L, and then looking forward for the "Local Modes:" string. This accomplishes these goals: a stray "Local Modes" not in the last page is not noticed; and visiting a long file that is all one page and has no local mode list need not take the time to search the whole file.

If you wish to make local one of the variables whose values take effect only through the & Process Options subroutine, you must also set the variable Switch Mode Process Options locally to 1, to make sure that & Process Options is called at all the times necessary to make the local values of the other variables to be activated and deactivated at the right times.

XXII Auto Save Mode -- Protection Against Crashes

If you turn on Auto Save Mode, EMACS will save your file from time to time (based on counting your commands) without being asked. Your file will also be saved if you stop typing for more than a few minutes when there are changes in the buffer. This prevents you from losing more than a limited amount of work when the system crashes.

You can turn auto saving on or off in an individual buffer with MM Auto Save. In addition, you can specify whether to do auto saving by default in all buffers by setting Q<ESC>Auto Save Default<ESC>. The frequency of saving, and the number of saved versions to keep, can both be specified.

Each time you visit a file, no matter whether you use C-X C-V, C-X C-R, C-X C-E or C-X C-F, auto saving will be on for that file if Q<ESC>Auto Save Default<ESC> is nonzero. However, by giving a nonzero argument to the file-visiting command, you can turn off auto saving FOR THAT FILE ONLY, without changing the default. Once you have visited a file, you can turn auto saving on with MM Auto Save, or off with 0MM Auto Save.

If you start typing a new file into a buffer without visiting anything, auto save mode has no effect. However, you should not be doing this. You should instead visit the non-existent file and then type in its contents. It will not bother EMACS that the "previous copy" was imaginary; if you change the file new copies will be saved by C-X C-S in the normal manner, and auto save mode will take effect.

Let us suppose that it is time for an automatic save to be done: where should the file be saved?

Two workable methods have been developed: save the file under the names you have visited, or save it under some special "auto save file name". Each solution has its good and bad points. The first one is excellent some of the time, but intolerable the rest of the time. The second is usually acceptable. Auto saving under the visited file's actual names means that you need do nothing special to gobble the auto save file when you need it; and it means that there is no need to worry about interference between two users sharing a directory, as long as they aren't screwing up by editing the same file at once. However, this method can have problems:

If you visit a file with other than C-X C-E, then you are unwilling to have EMACS rewrite that file even to avoid screwing you, so you presumably don't want mere auto saves to go there.

So in all those cases, auto saves are written as the filenames in Q<ESC>Auto Save Filenames<ESC>. But if none of those cases apply -- if you visit file ANYTHING with C-X C-E -- then auto saves are written out with the very same filenames you visited. Users of C-X C-F and the TAGS package should note that TAGS normally visits the file a la C-X C-V. Setting Q<ESC>Find File Inhibit Write<ESC> to 0 will cause them to visit the file a la C-X C-E, and use the more winning auto save mode when possible.

When auto saving is being done, C-X C-S writes out to the same place that auto saves are going. Doing C-U C-X C-S will file out "permanently" under the visited file names.

When it is time to recover from a system crash by reloading the auto save file, if auto saving was using the visited file names you have nothing special to do. If auto saving was using special auto save filenames, you should read in the last auto save file (whose name was printed out when the auto save was done), and then use C-X C-W to write it out elsewhere. Alternatively, you can visit the file you were editing, do HK in the minibuffer to kill the whole contents, and then do MM Insert File to read in the auto save file instead.

MM Revert knows how to find the most recent save, permanent or not, under whatever filenames. I repeat that if you change your mind about changes, DO NOT simply revisit the file, because if you used C-X C-E to visit it then the changed version will first be written out. Instead, use MM Revert File, which is specifically designed for backing up to the last version on disk.

For your protection, if a file has shrunk by more than 30% since the last save, auto saving will ask for confirmation before saving it.

Although auto saving will generate large numbers of files, it will not clog directories, because it cleans up after itself. Only the last Q<ESC>Auto Save Count<ESC> auto save files are kept; as further saves are done, old auto saves are deleted. However, files which were not made by auto saves (or manual but non-permanent C-X C-S saves) -- such as, old versions that were there before you started, or versions written with C-U C-X C-S -- are never deleted in this way. Giving an argument to MM Auto Save sets Q<ESC>Auto Save Count<ESC>, which starts at 2.

A pre-comma argument to MM Auto Save sets the number of commands between auto saves. Another way of setting this is to do <n>FS ^RMDLY<ESC> in the minibuffer.

Q<ESC>Auto Save Filenames<ESC> is usually set up by the default init file to _^RSV >, but you can change that. Or, if you use auto saving in multiple buffers a lot, you might want to have a Buffer Creation Hook which sets Auto Save Filenames to a filename based on the buffer name, so that different buffers don't interfere with each other.

XXIII Cleaning a File Directory

The normal course of editing constantly creates new versions of files. If you don't eventually delete the old versions, the directory will fill up and further editing will be impossible. EMACS has commands that make it easy to delete the old versions.

For complete flexibility to delete precisely the files you want to delete, you can use the DIRED package.

But there is a more convenient way to do the usual thing: keep only the two (or other number) most recent versions.

MM Reap File<ESC><file><ESC> counts the number of versions of <file>. If there are more than two, you are told the names of the recent ones (to be kept) and the names of the older ones (to be deleted), and ASKED whether to do the deletion (answer "Y" or "N"). In addition, if there is a file with an FN2 of MEMO, @XGP, XGP or UNFASL, you are asked whether it too should be deleted (this is actually controlled by a TECO search string in Q<ESC>Temp File FN2 List<ESC>).

If you give MM Reap File a null argument (i.e., if you use Meta-X, don't give it an argument) then it will apply to the file you are visiting.

MM Clean Directory<ESC><dirname><ESC> cleans a whole directory of old versions. Each file in the directory is processed a la MM Reap File. MM Clean Dir<ESC><ESC> (null argument, or no argument using Meta-X) cleans the directory containing the file you are visiting.

MM Reap File and MM Clean Dir can be given a numeric argument which specifies how many versions to keep (instead of two). For example, 4MM Reap File<ESC><ESC> would keep the four most recent versions. The default when there is no argument is the value of Q<ESC>File Versions Kept<ESC>, which is initially 2.

XXIV DIRED the Directory Editor Subsystem

DIRED makes it easy to delete many of the files in a single directory at once. It presents a copy of a listing of the directory, which you can move around in, marking files for deletion. When you are satisfied, you can tell DIRED to go ahead and delete the marked files.

Invoke DIRED with MM DIRED<ESC><ESC> to edit the current default directory, or MM DIRED<ESC><dir><ESC> to edit directory <dir>. (Use meta-X) You will then be given a listing of the directory which you can move around in with all the normal EMACS motion commands. However, some EMACS commands are illegal and others do special things.

You can mark a file for deletion by moving to the line describing the file and typing "D", "C-D", "K", or "C-K". The deletion mark is visible as a "D" at the beginning of the line. Point is moved to the beginning of the next line, so that several "D"'s will delete several files. Alternatively, if you give "D" an argument it will mark that many consecutive files. Given a negative argument, it will mark the preceding file (or several files) and put point at the first (in the buffer) line marked. Most of the DIRED commands (D, U, !, <ESC>, P, S, C, E, space) repeat this way with numeric arguments.

If you wish to remove a deletion mark, use the "U" (for Undelete) command, which is invoked just like "D": it removes the deletion mark from the current line (or next few lines). However, if the current line has no D on it, and the previous line does have a D, U removes that deletion mark and moves upward. This allows you to undo a D with just a U.

For extra convenience, Space is made a command similar to C-N. Moving down a line is done so often in DIRED that it deserves to be easy to type.

If you are not sure whether you want to delete a file, you can examine it by typing "E". This will enter a recursive ^R Mode on the file, which you can exit by typing Control-<ESC> or C-C C-C. The file is not really visited at that time, and C-X C-S will not work, so it is unwise to make changes to the file. After you exit the recursive ^R, You will be back in DIRED.

When you have marked the files you wish to mark, you can exit DIRED by typing "X", "Q", or "Control-<ESC>". If there were any files marked, DIRED will then list them in a concise format, several per line. A file with "!" appearing next to it in this list has not been saved on tape

and will be gone forever if deleted. A file with ">" in front of it is the most recent version of a sequence and you should be wary of deleting it. Then DIRED asks for confirmation of the list. You can type "YES" (Just "Y" won't do) to go ahead and delete them, "N" to return to editing the directory so you can change the marks, or "X" to give up and delete nothing. Anything else typed will make DIRED print a list of these responses and try again to read one.

There are some other ways to invoke DIRED. The Emacs command ^XD puts you in DIRED on the directory containing the file you are currently editing. With a numeric argument of 1 (^1^XD, <ESC>1^XD, or ^U1^XD), only the current file is displayed instead of the whole directory. In combination with the "H" command this can be useful for cleaning up excess versions of a file after a heavy editing session. With a numeric argument of 4 (^U^XD), it will ask you "Directory;". Type a directory name followed by a semicolon, and/or a file name. If you explicitly specify a file name only versions of that file are displayed, otherwise the whole directory is displayed.

It is unwise to try to edit the text of the directory listing yourself, without using the special DIRED commands, unless you know what you are doing, since you can confuse DIRED that way. To make it less likely that you will do so accidentally, the self-inserting characters and <Delete> are all made illegal inside DIRED. However, deleting whole lines at a time is certainly safe. This does not delete the files described by those lines; instead, it makes DIRED forget that they are there and thus makes sure they will NOT be deleted. Thus, MM Keep Lines<ESC> is useful if you wish to delete only files with a FOO in their names.

For more complicated things, you can use the minibuffer. When you call the minibuffer from within DIRED, you get a perfectly normal one. The special DIRED commands are not present while you are editing in the minibuffer. To mark a file for deletion, replace the space at the beginning of its line with a "D". To remove a mark, replace the "D" with a Space.

XXV Miscellaneous File Operations

MM Insert File<ESC> <file> <ESC> inserts the contents of <file> into the buffer at point, leaving point unchanged before the contents and mark after them. The current defaults are used for <file>, and are updated.

MM Write Region<ESC> <file> <ESC> writes the region (the text between point and mark) to the specified file. It does not set the visited filenames. The buffer is not changed.

MM Append to File<ESC> <file> <ESC> appends the region to <file>. The text is added to the end of <file>.

MM Prepend to File<ESC> <file> <ESC> adds the text to the beginning of <file> instead of the end.

MM Set Visited Filename<ESC><file><ESC> changes the name of the file being visited without reading or writing the data in the buffer. MM Write File is equivalent to this command followed by a C-X C-S.

MM List Files<ESC><dir spec><ESC> lists just the names of all the files in <dir>, several to a line.

The default filenames for all of these operations are the "TECO default filenames". Most of these operations also leave the TECO default names set to the file they operated on. The TECO default is NOT ALWAYS the same as the file you are visiting. When you visit a file, they start out the same, but the commands mentioned above can change the TECO default, thought they certainly don't change the visited filenames (so C-X C-S knows where to save your file). Each buffer has its own TECO default filenames.

If you wish to use the visited file's names as the defaults for a random file operation, use the C-X C-T command. It gets you a minibuffer initialized with an ET command containing the name of the visited file. That ET will, when executed, set the TECO default to that name. You can type the operation you want into the minibuffer after the ET. C-U C-X C-T will put the old TECO default names into the ET instead of the visited file's name. This just serves to show you what the TECO default is.

At times you want to "Change the name" of a file you are editing. The command C-X C-W allows you to save the buffer into a specified file, which then becomes the file you are visiting. It takes its arguments

and handles the defaults just like C-X C-E. In fact, it is equivalent to writing the buffer out to the file and then visiting that file.

The operation of visiting a file is available as a random file command under the name MM Visit File<ESC><file><ESC>. In this form, it uses the TECO default as its defaults, though it still sets both the TECO default and the visited filenames.

The variable Q<ESC>Auto Directory Display<ESC> can be set to make many file operations display the directory automatically. The variable is normally 0; making it positive causes write operations such as MM Write File to display the directory, and making it negative causes reads (MM Read File) to display it as well. The display is done using the default directory listing macro which is kept in Q<ESC>Directory Lister<ESC>. Normally, in EMACS, this is the macro that displays only the files related to the current default file.

XXVI Searching

Searching moves the cursor to the next occurrence of a string of characters which you specify. In EMACS, searching is "incremental", which means that as you type in the search string EMACS shows you where it would be found. When you have typed enough characters to identify the place you want, you can stop.

EMACS also has commands to find all or some occurrences of a string and replace them, print them, or count them. The command to search is C-S (^R Incremental Search). C-S reads in characters and positions the cursor at the first occurrence of the characters that you have typed. If you type C-S and then F, the cursor will move to right after the next "F". Type an "O", and see the cursor move to after the next "FO". Type another "O", and the cursor will be after the first "FOO" after the place where you started the search. At the same time, the "FOO" has echoed at the bottom of the screen.

If you type a mistaken character, you can rub it out. After the FOO, typing a <delete> will make the "O" disappear from the bottom of the screen, leaving only "FO". The cursor will move back to the "FO". Rubbing out the "O" and "F" will move the cursor back to where you started the search.

When you are satisfied with the place you have reached, you can type an <ESC>, which will stop searching, leaving the cursor where the search brought it. Any command except a printing character or CR will stop the searching and then be executed. Thus, if you find the word FOO you can type Meta-<Delete> to exit the search and kill the FOO. <ESC> is necessary only if the next thing you want to do is insert text, or rub out, since those things are special while searching.

Sometimes you search for "FOO" and find it, but not the one you expected to find. There was a second FOO that you forgot about, before the one you were looking for. Then, just 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 rub out the C-S's.

If your string is not found at all, the mode line starts saying "Failing I-Search". The cursor will be after the place where EMACS 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 <ESC> or some other EMACS command to

"accept what the search offered". Or you can type C-G, which will throw away the characters that could not be found (the "T" in "FOOT"), leaving those that could be found (the "FOO" in "FOOT"). A second C-G at that point would undo the search entirely.

The C-G "quit" command does special things during searches. What C-G does depends on the status of the search. If the search has found what you specified and is waiting for input, C-G cancels the entire search. The cursor moves back to where you started the search. If C-G is typed while the search is actually searching for something, or after search failed to find some of your input (searched all the way to the end of the file), then only the characters which have not been found are discarded. Having discarded them, the search is now successful and waiting for more input, so a second C-G will cancel the entire search. Make sure you wait for the first C-G to ding the bell before typing the second one; if typed too soon, the second C-G may be confused with the first and effectively lost.

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 will keep looking for more occurrences backwards. A C-S will start going forwards again. C-R's can be rubbed out just like anything else. If you know that you want to search backwards, you can use C-R instead of C-S to start searching backwards.

A non-incremental search is also available. Just type <ESC> right after the C-S to get it. Do MM Describe<ESC>^R String Search<ESC> for details. Some people who prefer non-incremental searches put that function on Meta-S, and ^R Character Search (do MM Describe<ESC> for details) on C-S.

XXVII Replacement Commands

To replace every instance of FOO with BAR, you can do MM Replace<ESC>FOO<ESC>BAR<ESC>. Replacement is done only after the pointer, so if you want to cover the whole buffer you must go to the beginning first. Unless Q<ESC>Case Replace<ESC> is zero, an attempt will be made to preserve case; give both FOO and BAR in lower case, and if a particular FOO is found with a capital initial or all capitalized, the BAR which replaces it will be likewise.

To restrict the replacement to a subset of the buffer, set the region around it and type C-X N ("N" for "Narrow"). This will make all of the buffer outside that region temporarily invisible (but the commands that save your file will still know that it is there!). Then do the replacement. Afterward, C-X W ("Widen") to make the rest of the buffer visible again.

Even if you are afraid that there may be some FOO's that should not be changed, EMACS can still help you. Use MM Query Replace<ESC>FOO<ESC>BAR<ESC>. This will display each FOO and wait for you to say whether to replace it with a BAR. The things you can type when you are shown a FOO are:

Space	to replace the FOO (preserving case, just like plain Replace, unless Q<ESC>Case Replace<ESC> is zero);
<Delete>	to skip to the next FOO without replacing this one;
Comma	to replace this FOO and display the result. You are then asked for another input character, except that since the replacement has already been made, <Delete> is treated like Space.
<ESC>	to stop replacing, not replacing this FOO;
Period	to replace this FOO but not look for any more;
!	to replace all remaining FOO's without asking (Replace actually works by calling Query Replace and pretending that a ! was typed in);
^	to go back to the previous FOO (or, where it was), in case you have made a mistake. This works by jumping to the mark (Query Replace sets the mark each time it finds a FOO).
C-R	to enter EMACS recursively, in case the FOO needs to be edited rather than just replaced with a BAR. When you are done, exit the recursive EMACS with C-C C-C.
C-W	to delete the FOO, and then call EMACS recursively. When you are finished editing whatever is to replace the FOO, exit the recursive EMACS with C-C C-C.

If you type any other character, the Query Replace will be exited, and the character will be then executed as a ^R Command. To restart the Query Replace, assuming you invoked it using the mini-buffer, use C-X <ESC>, which is a command to re-execute the previous minibuffer command.

If you give Replace or Query Replace an argument, then it insists that the occurrences of FOO be delimited by break characters (or an end of the buffer). So you can find only the word FOO, and not FOO when it is part of FOOBAR.

Meta-% will give you a mini-buffer pre-initialized with "MM Query Replace<ESC>". This is the easiest way to invoke Query Replace.

The FOO string argument to Replace and Query Replace is actually a TECO search string. This means that the characters C-X, C-B, C-N, C-O, C-Q and C-] have special meanings. C-X matches any character, C-B matches any "delimiter" character (anything which the word commands consider not part of a word, according to the delimiter dispatch table in Q-register ..D). C-N negates what follows, so that C-N A matches anything but A, and C-N C-B matches any non-delimiter. C-O means "or", so that XYXY C-O ZZZ matches EITHER XYXY or ZZ. C-Q quotes the following character, in case you want to search for one of the special control characters. C-] is special in a more fundamental way, which is probably useless to you; to search for a C-], you must put TWO C-]'s in the string you specify. Remember that these characters (and all control characters) must be quoted by C-Q just to insert them in the minibuffer. Thus, to insert C-Q C-B in the minibuffer, you must type 3 C-Q's and a C-B.

Here are some other commands related to replacement. Their arguments are TECO search strings. They all operate starting from the pointer, on to the end of the buffer (or where C-X N stops them).

MM Occur<ESC>FOO<ESC>
: which finds all occurrences of FOO after the pointer. It prints each line containing one. With an argument, it prints that many lines before and after each occurrence.

MM How Many<ESC>FOO<ESC>
: types the number of occurrences of FOO after the pointer.

MM Keep Lines<ESC>FOO<ESC>
: kills all lines after the pointer that don't contain FOO.

MM Flush Lines<ESC>FOO<ESC>
: kills all lines after the pointer that contain FOO.

XXVIII Indentation Commands for Code

Most programming languages have some indentation convention. For LISP code, lines are indented according to their nesting in parentheses. For MIDAS code, almost all lines start with a single tab, but some have one or more spaces as well. Indenting TECO code is an art rather than a science, but it is often useful to indent a line under the previous one.

The way to request indentation is with the Tab command. Its precise effect depends on the major mode. In LISP mode, Tab aligns the line according to its depth in parentheses. No matter where in the line you are when you type Tab, it aligns the line as a whole. In MIDAS mode, Tab inserts a tab, that being the standard indentation for assembly code. PL/I mode knows in great detail about the keywords of the language so as to indent lines according to the nesting structure.

The command Linefeed does a Return and then does a Tab on the next line. Thus, Linefeed at the end of the line makes a following blank line and supplies it with the usual amount of indentation, just as Return would make an empty line. Linefeed in the middle of a line breaks the line and supplies the usual indentation in front of the new line.

The inverse of Linefeed is Meta-^ or C-M-^. This command deletes the indentation at the front of the current line, and the line boundary as well. They are replaced by a single space, or by no space if before a ")" or after a "(" or "'". To delete just the indentation of a line, go to the beginning of the line and use Meta-\, which deletes all spaces and tabs around the cursor.

To insert an indented line before the current one, do C-A, C-O, and then Tab.

To make an indented line after the current one, use C-E Linefeed.

To move over the indentation on a line, do Meta-M or C-M-M (^R Back to Indentation). These commands, given anywhere on a line, will position the cursor at the first nonblank character on the line.

XXIX Automatic Indication Of How Parentheses Match

If you enable the closeparen-matching feature, whenever you type a closeparen EMACS will automatically show you, for an instant, the openparen which matches it.

To enable the feature, set Q<ESC>LISP) Hack<ESC> to 1 using MM Alter Options. The typing of a closeparen will then move the cursor to the matching openparen for 1 second, and then back again to the closeparen. If you type another command before the second is up, the cursor moves immediately back to its real location and the command is executed with no delay.

It is worth emphasizing that the real location of the cursor -- the place where your type-in will be inserted -- is not affected by the closeparen matching feature. It stays after the closeparen, where it would normally be. Only the spot on the screen moves away and back. You can type ahead freely as if the matching feature were not there. In fact, if you type fast enough, you won't see the cursor move. You must pause after typing a closeparen to see the openparen shown. Also, if there is no matching closeparen, nothing will happen.

Setting Q<ESC>LISP) Hack<ESC> to -1 instead of 1 will cause the matching openparen to be shown only if it is already on the screen.

While this feature was intended primarily for LISP, it can be used just as well for any other language, and it is not dependent on what major mode you are in (it originally was enabled only in LISP mode). If you wish to use it in a language which has several parenthetical characters, you can do so as long as you set up ..D appropriately and then put MM ^R Lisp) on the appropriate close-characters. Better yet, you can make define characters to indirect through the definition of ")". See the definition of the Muddle Mode command.

XXX Moving Over and Killing Lists and S-expressions

By convention, EMACS commands that deal with LISP objects are usually Control-Meta- characters. They tend to be analogous in function to the Control- and Meta- characters with the same basic character.

To move forward over an S-expression, use C-M-F (^R Forward Sexp). If the first non-"useless" character after point is an "(", C-M-F moves past the matching ")". If the first character is a ")", C-M-F just moves past it. If the character begins an atom, C-M-F moves to the atom-break character that ends the atom. C-M-F can with an argument repeats that operation the specified number of times; with a negative argument, it moves backward instead.

The command C-M-B (^R Backward Sexp) moves backward over an S-expression; it is like C-M-F with the argument negated. If there are "'"-like characters in front of the S-expression moved over, they are moved over as well. Thus, with the pointer after " 'FOO ", C-M-B would leave the pointer before the "'", not before the "F".

These two commands (and most of the commands in this section) do not know how to deal with the presence of comments. Although that would be easy to fix for forward motion, for backward motion the syntax of LISP makes it nearly impossible. Comments by themselves wouldn't be so bad, but handling comments and "|" both is impossible to do locally. In a line " ((FOO ; | BAR ", are the open parentheses inside of a "| ... |" atom? I do not think it would be advisable to make C-M-F handle comments without making C-M-B handle them as well.

For this reason, two other commands which move over lists instead of S-expressions are often useful. They are C-M-N (^R Forward List) and C-M-P (^R Backward List). They act just like C-M-F and C-M-B except that they don't stop on atoms; after moving over an atom, they move over the next expression, stopping after moving over a list. With this command, you can avoid stopping after all of the words in a comment.

Killing an S-expression at a time can be done with C-M-K and C-M-<Delete> (^R Forward Kill Sexp and ^R Backward Kill Sexp). C-M-K kills the characters that C-M-F would move over, and C-M-<Delete> kills what C-M-B would move over.

C-M-F and C-M-B stay at the same level in parentheses, when that's possible. To move UP one (or n) levels, use C-M-(or C-M-) (^R Backward Up List and ^R Forward Up List). C-M-(moves backwards up past one containing "(". C-M-) moves forwards up past one containing ")". Given

a positive argument, these commands move up the specified number of levels of parentheses. C-M-U is another name for C-M-(, which is easier to type, especially on non-Meta keyboards. If you use that name, it is useful to know that a negative argument makes the command move up forwards (ie, behave like C-M-)).

To move DOWN in list structure, use C-M-D (^R Down Sexp). It is nearly the same as searching for a "(".

A somewhat random-sounding command which is nevertheless easy to use is C-M-T (^R Exchange Sexps), which moves the cursor forward over one s-expression, dragging the previous s-expression along. An argument serves as a repeat count, and a negative argument drags backwards (thus cancelling out the effect of a positive argument). An argument of zero, rather than doing nothing, exchanges the s-expressions at the point and the mark.

To perform a miscellaneous operation on the next S-expression in the buffer, use or C-M-@ (^R Mark Sexp) which sets mark at the same place that C-M-F would move to. C-M-@ takes arguments like C-M-F. In particular, a negative argument is useful for putting the mark at the beginning of the previous S-expression.

In EMACS, a list at the top level in the buffer is called a Defun, because such lists usually are DEFUNs, regardless of what function it actually calls. There are EMACS commands to move to the beginning or end of the current Defun: C-M-[(^R Beginning of DEFUN) moves to the beginning, and C-M-] (^R End of DEFUN) moves to the end. If you wish to operate on the current Defun, use C-M-H (^R Mark DEFUN) which puts point at the beginning and mark at the end of the current or next Defun. Alternate names for these two commands are C-M-A for C-M-[and C-M-E for C-M-]. The alternate names are easier to type on many non-Meta keyboards.

The list commands' understanding of syntax is completely table-driven. Any character can, for example, be declared to act like an open paren. If you find that you are being "screwed by macro characters" then you may be able to win by changing their syntax. The table is the TECO "delimiter dispatch" held in q-register ..D. You can use MM Edit ..D<ESC> to change the entries in it. See TECO ORDER for full details.

XXXI Commands for Manipulating Comments

The command that creates a comment is Control-; or Meta-; (^R Indent for Comment). It moves to the end of the line, indents to the comment column, and inserts whatever string EMACS believes comments are supposed to start with (normally ";"). If the line goes past the comment column, then the indentation is done to a suitable boundary (usually, a multiple of 8).

Control-; can also be used to align an existing comment. If a line already contains the string that starts comments, then C-; just moves point after it and indents it to the right place (where a comment would have been created if there had been none).

Even when an existing comment is properly aligned, C-; is still useful for moving directly to the start of the comment.

If you wish to align a large number of comments, you can give Control-; an argument and it will indent what comments exist on that many lines, creating none. Point will be left after the last line processed (unlike the no-argument case).

If you are typing a comment and find that you wish to continue it on another line, you can use the command Meta-J or Meta-Linefeed (^R Indent New Comment Line), which terminates the comment you are typing, creates or gobbles a new blank line, and begins a correctly indented comment on it. Note that if the next line is not blank, a blank line is created, instead of putting the next line of the comment on the next line of code.

The comment column is stored in the variable Q<ESC>Comment Column<ESC>. Thus, 40U<ESC>Comment Col<ESC> sets it to column 40. Alternatively, the command C-X; (^R Set Comment Column) sets the comment column to the column point is at. C-U C-X; sets the comment column to match the last comment before point in the buffer, and then does a Meta-; to align the current line's comment under the previous one.

Because the major modes supply default values for the comment column, switching buffers will reset it to the default unless you make the variable Q<ESC>Comment Column<ESC> local in the buffer in which you want to respecify it. C-X; and C-U C-X; do this automatically.

C-M-; (^R Kill Comment) kills the comment on the current line, if there is one. The indentation before the start of the comment is killed as well. If there does not appear to be a comment in the line, nothing is done.

The string recognized as the start of a comment is stored in the variable Q<ESC>Comment Start<ESC>, while the string used to start a new comment is kept in Q<ESC>Comment Begin<ESC> (if that is zero, Comment Start is used for new comments). This makes it possible for you to have any ";" recognized as starting a comment but have new comments begin with "; ".

The string used to end a comment is kept in the variable Q<ESC>Comment End<ESC>. This is used only rarely; usually you must insert the end of a comment yourself. In many languages no comment end is needed as the comment extends to the end of the line.

XXXII LISP Grinding

The best way to keep LISP code indented properly is to use EMACS to re-indent it when it is changed. EMACS has commands to indent properly either a single line, a specified number of lines, or all of the lines inside a single S-expression.

The basic indentation function is ^R Indent for LISP, which gives the current line the correct indentation as determined from the previous lines' indentation and parenthesis structure. This function is normally found on C-M-Tab, but when in LISP mode it is placed on Tab as well (Meta-Tab inserts a Tab). When given at the beginning of a line, it leaves point after the indentation; when given inside the text on the line, point remains fixed with respect to the characters around it.

When entering a large amount of new code, it becomes useful that Linefeed is equivalent to a CR followed by a Tab. In LISP mode, a Linefeed will create or move down onto a blank line, and then give it the appropriate indentation.

If you wish to join two lines together, you can use the Meta-^ or Control-Meta-^ command (^R Delete Indentation), which is approximately the opposite of Linefeed. It deletes any Spaces and Tabs at the front of the current line, and then deletes the CRLF separating it from the preceding line. A single space is then inserted, if EMACS thinks that one is needed there.

If you are dissatisfied about where Tab wants to place the second and later lines of an S-expression, you can override it. If you alter the indentation of one of the lines yourself, then Tab will indent successive lines of the same list to be underneath it. This is the right thing for functions which Tab indents unaesthetically. Of course, it is the wrong thing for PROG tags (if you like to un-indent them), but it's impossible to be right for both.

When you wish to re-indent code which has been altered or moved to a different level in the list structure, you have several commands available. You can re-indent a specific number of lines by giving the ordinary indent command (Tab in LISP mode) an argument. It will indent as many lines as you say and move to the line following them. Thus, if you underestimate, you can give a similar command to indent some more lines.

You can re-indent the contents of a single S-expression by positioning point before the beginning of it and typing Control-Meta-Q

(^R Indent Sexp). The line the S-expression starts on is not re-indented; thus, only the relative indentation with in the S-expression, and not its position, is changed. To correct the position as well, simply give a Tab before the C-M-Q.

Another way to specify the range to be re-indented is with point and mark. The command C-M-\ (^R Indent Region) applies Tab to every line whose first character is between point and mark. In LISP mode, this does a LISP indent.

XXXIII Commands for Editing Assembly Language Programs

MM MIDAS Mode<ESC> provides several commands that know the format of labels and instructions in MIDAS and other PDP-10 and PDP-11 assemblers. These commands are

C-M-N	Go to Next label
C-M-P	Go to Previous label
C-M-A	Go to Accumulator field of instruction
C-M-E	Go to Effective Address field
C-M-D	Kill next word and its Delimiting character
M-[	move up to previous paragraph boundary
M-]	move down to next paragraph boundary

Any line which is not indented and is not just a comment is taken to contain a label. The label is everything up to the first whitespace (or the end of the line). C-M-N and C-M-P both position the cursor right at the end of a label; C-M-N moves forward or down and C-M-P moves backward or up. At the beginning of a line containing a label, C-M-N moves past it. Past the label on the same line, C-M-P moves back to the end of it. If you kill a couple of indented lines and want to insert them right after a label, these commands put you at just the right place.

C-M-A and C-M-E move to the beginning of the AC or EA fields of a PDP-10 instruction. They always stay on the same line, and will move either forward or backward as appropriate. If the instruction contains no AC field, C-M-A will position to the start of the address field. If the instruction is just an opcode with no AC field or address field, a space will be inserted after the opcode and the cursor left after the space.

Once you've gone to the beginning of the AC field you can often use C-M-D to kill the AC name and the comma which terminates it. You can also use it at the beginning of a line, to kill a label and its colon, or after a line's indentation to kill the opcode and the following space. This is very convenient for moving a label from one line to another. In general, C-M-D is equivalent to M-D C-D, except that all the characters are saved on the kill ring, together. C-D, a "deletion" command, doesn't save on the kill ring.

The M-[and M-] commands are not, strictly speaking, redefined by MIDAS mode, since they always go up or down to a paragraph boundary. However, in MIDAS mode the criterion for a paragraph boundary is redefined by changing Q<ESC>Paragraph Delimiter<ESC> (so that only blank lines (and pages) delimit paragraphs. So, M-[will move up to the previous blank line and M-] will move to the next one.

XXXIV Commands Good for English Text

EMACS has commands for moving over or operating on words. By convention, they are all Meta- characters.

M-F	Move Forward over a word
M-B	Move Backward over a word
M-D	Kill up to the end of a word
M-<Delete>	Kill back to the beginning of a word
M-@	Mark the end of the next word
M-T	Exchange two words; drag a word forward or backward across other words.

Notice how these commands form a group that parallels the character based commands C-F, C-B, C-D, C-T and <Delete>. M-@ is related to C-@.

The commands Meta-F and Meta-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, Meta-F and Meta-B will move several words if given an argument, and can be made to go in the opposite direction with a negative argument. 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. Meta-D or Meta-K (^R Forward Kill Word) kills the word after point. To be precise, it kills everything from point to the place Meta-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 Meta-F to get the end, and kill the word backwards with Meta-<Delete>. Meta-D takes arguments just like Meta-F.

Meta-<Delete> (^R Backward Kill Word) kills the word before point. It kills everything from point back to where Meta-B would move to. If point were after the Space in "FOO, BAR", "FOO, " would be killed. In such a situation, to avoid killing the Comma and Space, do a Meta-B and a Meta-D instead of a Meta-<Delete>.

Meta-T (^R Exchange Words) moves the cursor forward over a word, dragging the word behind (or around) the cursor forward along with it. A numeric argument serves as a repeat count. A negative argument undoes the effect of a positive argument; it drags the word behind the cursor backward over a word. An argument of zero, instead of doing nothing, exchanges the word at point with the word at mark.

To operate on the next n words with an operation which applies between point and mark, you can either set the mark at point and then move over the words, or you can use the command Meta-@ (^R Mark Word) which does not move point, but sets the mark where Meta-F would move to. They can be given arguments just like Meta-F. The case conversion operations alternative forms that apply to words, since they are particularly useful that way.

Note that if you are in Atom Word Mode and in LISP mode, all the word commands move over/kill/mark LISP atoms.

The word commands' understanding of syntax is completely table-driven. Any character can, for example, be declared to be a delimiter. The table is the TECO "delimiter dispatch" held in q-register ..D. See TECO ORDER for full details. Also look at the documentation of MM Edit ..D<ESC>, a command for interactively changing the delimiter dispatch.

XXXV Sentence and Paragraph Commands

The EMACS commands for manipulating sentences and paragraphs are all Meta- commands, so as to resemble the word-handling commands.

M-A move back to the beginning of the sentence.
M-E move forward to the end of the sentence.
M-[move back to previous paragraph boundary.
M-] move forward to next paragraph boundary.
M-H put point and mark around this paragraph (around the following one, if between paragraphs).
C-X <Delete> kill back to beginning of sentence.

The commands Meta-A and Meta-E (^R Backward Sentence and ^R Forward Sentence) 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, but unlike those Control characters Meta-A and Meta-E will if repeated move over several sentences. EMACS considers a sentence to end wherever there is a ".", "?" or "!" followed by the end of a line or two Spaces, with any number of ")"'s, "]"'s, "'"'s, or '"' 's allowed in between. The sentence which is ending there is currently considered to include the Spaces or CRLF, so that Meta-E moves past them while Meta-A stays after them.

The command C-X <Delete> (^R Backward Kill Sentence) is useful when you change your mind about a thought after a phrase or two.

There are similar commands for moving over paragraphs. Meta-[(^R Backward Paragraph) moves to the beginning of the current or previous paragraph, while Meta-] (^R Forward Paragraph) moves to the beginning of the next paragraph. More precisely, a paragraph boundary occurs on any blank line followed by a non-blank line, and also on any non-blank line following a non-blank line and beginning with one of a user-specifiable set of characters, usually Space, Tab and Period, or beginning with a page-separator. Meta-[moves backwards till it finds such a boundary, while Meta-] moves forwards till it finds one. In addition, whenever a line starting with Period or "-" starts a paragraph, that line is a paragraph to itself; the next nonblank line always starts another paragraph. This is useful for text justifier commands.

In major modes for programs (as opposed to Text mode), paragraphs are determined only by blank lines. This makes the paragraph commands continue to be useful even though there are no paragraphs per se.

When you wish to operate on a paragraph, you can use the command Meta-H (^R Mark Paragraph) to prepare. This command puts point at the beginning and mark at the end of the paragraph point was in. Before setting the new mark at the end, a mark is set at the old location of point; this allows you to undo a mistaken Meta-H with two C-U C-@'s. If point is between paragraphs (in a run of blank lines, or at a boundary), the paragraph following point is surrounded by point and mark. Thus, for example, Meta-H C-W would kill the paragraph around or after point.

One way to make an "invisible" paragraph boundary that will not show if the file is printed is to put Space-Backspace at the front of a line. The Space will make the line appear (to the EMACS pararaph commands) to be indented, which will usually mean that it starts a paragraph (according to Q<ESC>Paragraph Delimiter<ESC>).

The variable Q<ESC>Paragraph Delimiter<ESC> should be a TECO search string composed of various characters separated by C-O's. A line is taken to start a paragraph if it begins with one of those characters (or with a page separator: see Q<ESC>Page Delimiter<ESC>). In addition, if the first character of Q<ESC>Paragraph Delimiter<ESC> is a ".", then any line starting with a "." is considered to be a complete paragraph by itself. This is the right thing for most text justifiers. Of course, any blank line followed by a nonblank line always starts a paragraph regardless of Q<ESC>Paragraph Delimiter<ESC>.

XXXVI Indentation Commands for Text

Tab	Indents line "appropriately".
M-Tab	Inserts a Tab character.
Linefeed	Is the same as Return and Tab.
M-^	Undoes a Linefeed. Merges two lines.
M-M	Moves to the line's first nonblank character.
M-I	Indent to Tab stop
C-M-\	Indent several lines to same column.
C-X Tab	Shift block of lines rigidly right or left.

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. You can set the tab stops with the Edit Tab Stops command.

For English text, the usual convention is that only the beginning of a paragraph is indented. You don't need any special hackery not to indent lines. But sometimes you want to have an indented paragraph. For this, use the Edit Indented Text command, which enters a submode in which Tab and the paragraph commands are suitably redefined.

When Auto-Fill mode needs to break an overlong line, it usually indents the newly started line. However, in Text mode, this is inhibited by clearing Q<ESC>Space Indent Flag<ESC>. This is so that Auto Fill can avoid indenting without denying you the use of Tab to indent. The Edit Table and Edit Indented Text commands temporarily change the value of Space Indent Flag.

To undo a line-break, whether done manually or by Auto Fill, use the Meta-^ command to delete the indentation at the front of the current line, and the line boundary as well. They are replaced by a single space, or by no space if before a ")" or after a "(" or "'". To delete just the indentation of a line, go to the beginning of the line and use Meta-\, which deletes all spaces and tabs around the cursor.

To insert an indented line before the current one, do C-A, C-O, and then Tab.

To make an indented line after the current one, use C-E Linefeed.

To move over the indentation on a line, do Meta-M or C-M-M (^R Back to Indentation). These commands, given anywhere on a line, will position the cursor at the first nonblank character on the line.

For typing in tables, you can use Text mode's definition of Tab, ^R Tab to Tab Stop, which may be given anywhere in a line, and indents from there to the next tab stop. Set the tab stops using the Edit Tab Stops command, which displays for you a buffer whose contents define the tab

stops. The second line contains a colon or period at each tab stop. Colon indicates an ordinary tab, which you fill to with whitespace; a period specifies that characters be copied from the corresponding columns of the line above it. Thus, you can tab to a column automatically inserting dashes or periods. The third line in the tab stops buffer contains column numbers for your convenience in editing the first two lines.

There are also commands for changing the indentation of several lines at once. Control-Meta-\ (^R Indent Region) gives each line whose first character is between point and mark the "usual" indentation (as determined by Tab). With a numeric argument, it gives each line precisely that much indentation. C-X Tab (^R Indent Rigidly) moves all of the lines in the region right by its argument (left, for negative arguments).

Usually, EMACS uses both tabs and spaces to indent. If you don't want that, you must redefine both & Indent and & Xindent to insert only spaces (do MM Describe<ESC> on them). To convert all tabs in a file to spaces, you can use MM Untabify<ESC>, whose argument is the number of positions to assume between tab stops (default is 8). MM Tabify<ESC> performs the opposite transformation, replacing pairs of spaces with tabs whenever possible.

XXXVII Text Filling

Space	in Auto Fill mode, breaks lines when appropriate.
M-Q	Fill paragraph.
M-G	Fill region (G is for Grind, by analogy with Lisp).
M-S	Center a line.

EMACS's auto fill mode lets you type in text that is filled (broken up into lines that just fit into a specified width) as you go. If you alter existing text and thus cause it to cease to be properly filled, EMACS can rearranged it to be filled if you ask.

Entering auto fill mode is done with the command MM Auto Fill, with no argument. From then on, lines will be broken automatically at spaces when they get longer than the desired width. New lines will usually be indented, but in Text Mode they are not. To leave auto fill mode, execute MM Auto Fill with an argument of zero.

When you finish a paragraph, you can type Space with an argument of zero. If the last word of the paragraph doesn't fit in its line, it will move to a new one. But no spaces will be inserted. Return will have the same effect, as well as inserting a line boundary.

If you edit the middle of a paragraph, it will cease to be correctly filled. To re-fill a paragraph, use the command Meta-Q (^R Fill Paragraph). It causes the paragraph that point is inside, or the one after point if point is between paragraphs, to be re-filled. All the line-breaks are removed, and then new ones are inserted where necessary. If the paragraph contains extra spaces (for example, if it was justified), giving Meta-Q a negative argument will remove them.

If you are not happy with Meta-Q's idea of where paragraphs start and end (the same as Meta-H's; Sometimes, it is ok to fill a region of several paragraphs at once. Meta-G will recognize a blank line or an indented line as starting a paragraph and not fill it in with the preceding line. The sequence Space-Backspace at the front of a line will prevent it from being filled into the preceding line but will be invisible when the file is printed. However, the full generality of Meta-H (such as recognizing TJ6 command lines) is not available.

Giving an argument to M-G or M-Q causes the text to be justified instead of filled. This means that extra spaces will be inserted between the words so as to make the right margin come out exactly even. You can unjustify it (remove the spaces) by giving M-G or M-Q a negative argument.

The command Meta-S (^R Center Line) centers a line within the current line width. With an argument, it centers several lines and moves past them.

The maximum line width for filling is stored in Q<ESC>Fill Column<ESC>. Both types of filling make sure that no line exceeds this width. The easiest way to set this variable is to use the command C-X F (^R Set Fill Column) which places the margin at the column point is on, or wherever you specify with a numeric argument. If the fill column has never been set (or has been set to zero) then the margin is defaulted to column 60.

The beginnings of a facility for filling text which is all indented is embodied in the variable Q<ESC>Fill Prefix<ESC>. If this variable is nonzero, it should be a string which is the current "prefix" for text lines. Before filling, the prefix is removed from the front of every line. After filling, the prefix is reinserted at the front of every line. The command C-X . (^R Set Fill Prefix) sets the fill prefix to what is on the current line up to the pointer. This feature doesn't totally win, so suggestions are solicited.

The variable Q<ESC>Space Indent Flag<ESC> controls whether auto fill mode indents the new lines that it creates. A nonzero value means that indentation should be done.

XXXVIII Case Conversion Commands

EMACS has commands for converting either a single word or any arbitrary range of text to upper case or to lower case.

M-L	Convert following word to lower case.
M-U	Convert following word to upper case.
M-C	Capitalize the following word.
C-X C-L	Convert region to lower case.
C-X C-U	Convert region to upper case.

The word conversion commands are the most useful. Meta-L (^R Lowercase Word) converts the word after point to lower case, moving past it. Thus, successive Meta-L's will convert successive words. Meta-U (^R Uppercase Word) converts to all capitals instead, while Meta-C (^R Uppercase Initial) puts the first letter of the word into upper case and the rest into lower case. All these commands will convert several words at once if given an argument.

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.

Note that if one of the word case conversion commands is given in the middle of the word, the part of the word before point is ignored.

The other case conversion commands are C-X C-U (^R Uppercase Region) and C-X C-L (^R Lowercase Region), which convert everything between point and mark to the specified case. Point and mark do not move. These commands ask for confirmation if the region contains more than Q<ESC>Region Query Size<ESC> characters; they also save the original contents of the region on the kill stack, so that M-W M-Y will undo them.

XXXIX Commands That Apply to Pages

C-M-L	Insert formfeed.
C-X C-P	Put point and mark around this page (or another page).
C-X [	Move point to previous page boundary.
C-X]	Move point to next page boundary.
C-X P	Narrow down to just this (or next) page.
C-X L	Count the lines in this page.

Most editors make the division of a file into pages extremely important. The page separators have fundamental effects on the editors' behavior. EMACS, on the other hand, treats a formfeed character just like any other character. It can be inserted or deleted in the ordinary way (but, for convenience, C-M-L inserts a formfeed). Thus, you are free to paginate your file, or not, according to other criteria. However, since pages are often meaningful divisions of the file, commands are provided to work on them.

The C-X [(^R Previous Page) command moves point to the previous page delimiter (actually, to right after it). If point starts out right after a page delimiter, it skips that one and stops at the previous one. A numeric argument serves as a repeat count. The C-X] (^R Next Page) command moves past the next page delimiter.

The C-X C-P command (^R Mark Page) puts point at the beginning of the current page and the mark at the end. The page terminator at the end is included (the mark follows it). That at the front is excluded (point follows it). Thus, this command can be followed by a C-W to kill a page which is to be moved elsewhere. Or, it can be used just to go to the top of the page.

A numeric argument to C-X C-P is used to specify which page to go to, relative to the current one (The design of TECO is such that it is not possible to know the absolute number of the page you are in). Zero means the current page. One means the next page, and -1 means the previous one.

The command C-X P (^R Set Bounds Page) narrows down to just one page. Everything before and after becomes temporarily invisible and inaccessible (Use C-X W to undo this. Both page terminators, the preceding one and the following one, are excluded from the visible region. Like C-X C-P, the C-X P command normally selects the current page, but allows you to specify which page explicitly relative to the current one with a numeric argument. However, when you are already narrowed down to one page, C-X P moves you to the next page (otherwise,

it would be a useless no-op). One effect of this quirk is that several C-X P's in a row get first the current page and then successive pages.

Just what delimits pages is controlled by the variable Page Delimiter, which should contain a TECO search string which will find a page separator. Normally, it contains a string containing just ^L. For this file, it contains (In any case, page separators are recognized as such only at the beginning of a line. The paragraph commands consider each page boundary to delimit paragraphs as well.

The C-X L command (^R Count Lines Page) is good for deciding where to break a page in two. It first prints (in the echo area) the total number of lines in the current page, and then divides it up into those preceding the current line and those following, as in "96 (72+25)". Notice that the sum is off by one; that will happen whenever point is not at the front of a line.

XL Narrowing

C-X N Narrow down to between point and mark.
C-X W Widen to view the entire buffer.
C-X P Narrow down to the page point is in.

"Narrowing" means focusing in on some portion of the buffer, making the rest temporarily invisible and inaccessible.

When you have narrowed down to a part of the buffer, that part appears to be all there is. You can't see the rest, you can't move into it (motion commands won't go outside the visible part), you can't change it in any way. However, it is not gone, and if you save the file you are editing all the invisible text will be saved. In addition to sometimes making it easier to concentrate on a single subroutine or paragraph by eliminating clutter, narrowing can be used to restrict the range of operation of a replace command.

The primary narrowing command is C-X N (^R Set Bounds Region). It sets the "virtual buffer boundaries" at point and the mark, so that only what was between them remains visible. Point moves to the top of the now-visible range, and the mark is left at the end, so that the region marked is unchanged.

The way to undo narrowing is to widen with C-X W (^R Set Bounds Full). This makes all text in the buffer accessible again.

Another way to narrow is to narrow to just one page, with C-X P.

Note that the virtual buffer boundaries are a powerful TECO mechanism used internally in EMACS in many ways. While only the commands described here set them permanently, many others set them temporarily.

XLI Libraries of EMACS Commands.

All EMACS commands, including the ones described in this document, reside in sharable libraries. A command is not accessible unless the library that contains it is loaded. Every EMACS starts out with one library loaded: the EMACS library, which contains all of the commands described in this document, except those explicitly stated to be elsewhere. Other libraries full of commands are provided with EMACS, and can be loaded explicitly when requested to make the commands in them available.

To load a library permanently, say M-X Load Library<ESC><libname><ESC>. The library will be found, either on your own directory or whichever one you specify, or on the EMACS directory, and loaded in. All the commands in the library are then available for use. Whenever you use M-X, the command name you specify is looked up in each of the libraries which you have loaded. If any library contains a definition for that name, the definition is executed.

You can also load a library temporarily, just long enough to use one of the commands in it. This avoids taking up space permanently with the library. Do this with the MM command Run Library, as in M-X Run<ESC><libname><ESC><command name><ESC>. The library <libname> will be loaded in, and <command name> executed. Then the library will be removed from the EMACS job. You can load it in again later.

M-X List Loaded Libraries types the names and brief descriptions of all the libraries loaded, last loaded first. You will see that the last one is always the EMACS library.

Libraries are loaded automatically in the course of executing certain commands. You will not normally notice this. For example, the TAGS library is automatically loaded in whenever you use the M-. or Visit Tag Table commands for the first time. This process is known as "autoloading". It is used to make the commands in the TAGS library available without the user's having to know to load the library himself, while not taking up space in EMACS's of people who aren't using them.

You can make your own libraries, which you and other people can then use, if you know how to write TECO code.

XLII The Minibuffer

The minibuffer is a facility by means of which EMACS commands can read input from the terminal, allowing you to use EMACS commands to edit the input while you are typing it. The minibuffer also allows the EMACS user to type in a TECO program and execute it.

You can always tell when you are in a minibuffer, because the Mode line will contain something in parentheses, such as "(Minibuffer)" or "(Query Replace)". There will also be a line of dashes across the screen a few lines from the top. Strictly speaking, the minibuffer is actually the lines above the line of dashes, for that is where you edit the input that the minibuffer is asking you for. The editing region has been limited to a few lines so that most of the screen can continue to show the file you are editing.

Often, a minibuffer will start out with some text in it. This means that you are supposed to add to that text, or, sometimes, to delete some of it so as to choose among several alternatives. For example, Meta-% (^R Query Replace) provides you with a minibuffer initially containing the string "MM Query Replace<ESC>". The cursor comes at the end. You are then supposed to add in the arguments to the Query Replace. MM @ TECO<ESC> needs to know how you want it to output your listing, so it offers you a minibuffer containing three lines, each of which will output the listing in one way, and asks you to delete the lines you don't want.

In a minibuffer, you can edit your input until you are satisfied with it. When that happens, you have to tell EMACS that you wish to submit the input. You do that by typing two <ESC>s. An <ESC> not followed by another <ESC> is simply inserted in the buffer. This is because it is common to want to put <ESC>s into the minibuffer, which usually contains a string of TECO commands. For example, in Meta-% (^R Query Replace) each argument must be ended by an <ESC>. The two <ESC>s you use to exit are automatically moved to the end, since <ESC>s are so often needed there.

Since <ESC> is self-inserting, typing Meta characters can be a problem. You can do it by using C-\ instead of <ESC> as the Meta-prefix. If you type a Control-Meta character on your keyboard, the corresponding ASCII control character is inserted in the minibuffer. This is because the Lisp commands are rarely useful when editing TECO commands, but insertion of control characters is frequent. If you really want to use a Control-Meta EMACS command, you must use C-C to type it.

You can cancel your input in a minibuffer and start all over again by typing C-G. That will kill all the text in the minibuffer. A C-G typed when the minibuffer is already empty will exit from the minibuffer. Usually, this will abort whatever command was using the minibuffer, so it will return without doing anything more. For example, if you type two C-G's at Meta-%'s minibuffer, you will return to top level and no query replace will be done. Typing a single C-G at a preinitialized minibuffer to empty the buffer is not very useful, since you would have to retype all the initial text.

If you want to type in a TECO command, you can invoke the minibuffer with the EMACS command Meta-<ESC>, or <ESC> <ESC> (^R Execute Minibuffer). An empty minibuffer will appear, into which you should type the TECO command string. When you exit with <ESC> <ESC>, the two <ESC>s that you exit with will be moved to the end of the command string, so if you wanted an <ESC> where the cursor was you should have inserted one with a C-Q <ESC>. Then the command string will be executed, in the environment from which the minibuffer was invoked.

The last five distinct minibuffer commands you have issued are remembered in a q-vector in q-register .M. (M-X commands are also remembered there). The C-X <ESC> command (^R Re-execute Minibuffer) will re-execute the last command (minibuffer or M-X) in that list. With an argument <n>, it will execute the <n>'th previous command. Just in case the command is an MM Revert, it is printed out (only the first 40 characters or so) and you are asked to confirm with "Y" or "N".

You can also get your previous minibuffer and M-X commands back into the minibuffer to be edited and re-executed with changes. Giving M-<ESC> and argument, as in C-U M-<ESC>, causes the minibuffer to be loaded up with the most recent previous command, just as if you had typed it in again from scratch. You can then edit it, execute it by typing two <ESC>s, or cancel it with C-G. To get an earlier command string instead of the most recent one, use the command C-C C-Y once you are in the minibuffer. This command will "rotate" the ring of saved commands much as M-Y rotates the ring of killed text. Each C-C C-Y will reveal an earlier command string, until the ring has rotated all the way around and the most recent one reappears. C-C C-Y is actually a way of saying C-M-Y, but in the minibuffer that's the only way to type it, since <ESC> inserts itself and Control-Meta characters insert control characters.

If you exit from Meta-<ESC> with a C-G, nothing is executed and the previous minibuffered command string is still remembered as the last one.

The command Meta-X (^R MM Via Minibuffer) which is used for executing a named EMACS command works by means of a minibuffer which is initialized with an "MM". You can get the same result with Meta-<ESC> if you type the MM yourself. Or, if you type Meta-X and delete the MM

which is provided for you, you can execute any TECO command string. Most commands that provide an initialized minibuffer simply run whatever is in the minibuffer when you exit.

While in a minibuffer, if you decide you want the minibuffer to use more lines on the screen, you can use C-X ^ (^R Grow Window) to get more. It gets one more line, or as many lines as its argument says.

XLIII Keyboard Macros.

A keyboard macro is a command defined by the user to abbreviate a sequence of other commands. If you discover that you are about to type C-N C-D forty times, you could define a keyboard macro to do C-N C-D and call it with a repeat count of forty. To use keyboard macros, you must load the KBDMAC library.

Keyboard macros differ from ordinary EMACS commands, also commonly known as macros, in that they are written in the EMACS command language rather than in TECO. This makes it easier for the novice to write them, and makes them more convenient as temporary hacks. However, the EMACS command language is not powerful enough as a programming language to be useful for writing anything intelligent or general. For such things, TECO must be used.

You define a keyboard macro while executing the commands which are the definition. Put differently, as you are defining a keyboard macro, the definition is being executed for the first time. This way, you can see what the effects of your commands are, so that you don't have to figure them out in your head. When you are finished, the keyboard macro is defined and also has been, in effect, executed once. You can then do the whole thing over again by invoking the macro.

To start defining a keyboard macro, type the C-X (command (^R Start Kbd Macro). From then on, your commands continue to be executed, but also become part of the definition of the macro. When you are finished, the C-X) command (^R End Kbd Macro) terminates the definition (without becoming part of it!). The macro thus defined can be invoked again with the C-X E command, which may be given a repeat count as a numeric argument to execute the macro many times. An argument of zero means to repeat the macro indefinitely (until it gets an error). C-X) can also be given a repeat count as an argument, in which case it re-invokes the macro, but defining the macro counts as one repetition (since it is executed as you define it). So, giving C-X) an argument of 2 would execute the macro one additional time.

If you wish to save a keyboard macro for longer than until you define the next one, you must give it a name. Do M-X Name Kbd Macro<ESC>FOO<ESC> and the last keyboard macro defined -- the one which C-X E would invoke -- is given the name FOO. M-X FOO will from then on invoke that particular macro. Name Kbd Macro will also read a character from the keyboard and redefine that character to invoke the macro. If you don't want to redefine any character, type a Return or <Delete>. Only self-inserting and undefined characters, and those that are already

keyboard macros, can be redefined in this way. Prefix characters may be used in specifying the character to be redefined.

To examine the definition of a keyboard macro, use the View Kbd Macro command. Either supply the name of the MM command which runs the macro as a string argument, or type the character which invokes the command on the terminal when the command asks for it.

You can get the effect of Query Replace, where the macro asks you each time around whether to make a change, by using the command C-X Q (^R Kbd Macro Query) in your keyboard macro. When you are defining the macro, the C-X Q does nothing, but when the macro is invoked the C-X Q reads a character from the terminal to decide whether to continue. The special answers are Space, <Delete>, <ESC>, C-L, C-R. A Space means to continue. A <Delete> means to skip the remainder of this repetition of the macro, starting again from the beginning in the next repetition. An <ESC> ends all repetitions of the macro, but only the innermost macro (in case it was called from another macro). C-L clears the screen and asks you again for a character to say what to do. C-R enters a recursive ^R Mode edit; when you exit, you are asked again (if you type a space, the macro will continue from wherever you left things when you exited the C-R). Anything else exits all levels of keyboard macros and is reread as a command.

00 BURT DECGRAM ACCEPTED 8 6592 0 477 31-JUL-81 19:25:56

****
* d i g i t a l *
****

file

TO: see "TO" DISTRIBUTION

cc: see "CC" DISTRIBUTION

DATE: FRI 31 JUL 1981 19:13 EST
FROM: GORDON BELL
DEPT: ENG STAFF
EXT: 223-2236
LOC/MAIL STOP: ML12-1/A51

SUBJECT: VT/Z8 THE NEXT STEP, AN INSTANT ENTRY TO PC MARKET?

Now that John has the 64Kbyte z80 running that could be put in a vt100 or a vt125, Ken has asked us to take the next step and determine what is required to make this a full blown product.

Can we get together to explore this early Monday morning?

The product as explained to me:
vt100 + optional graphics board + Z board + 2 minifloppies
= DEC's first personal computer

Z board = Z80 + 65Kbytes + 4 ports + floppy plus + IEEE 488 plus

The questions we should answer:
1. Is it real and solidly ready to be sent to PCB layout?
2. What's the software acquistion process?
3. Which floppies? 3-6581
4. What would be the timescale?
5. Who will take the engineering responsibility?
6. Where would we build it (eg. Mill)?

Ideally we would answer all these on Monday morning and then set out to make the product in a very fast fashion. The announce schedule would be about August 21, or ONE week after the IBM announcement!

This would mean we would really have to get on the stick to get the work done for productization. I think we could do it. ... what could give? (It does more than what GIGI+ is to do, but the computer doesn't fit in the keyboard, so there could be some resources if we can avoid making GIGI+ ... and losing our shirt. This would also give the education group a personal computer in the shortest, possible time. Similarly, the Terminals Eng. group in MR could also do this since they have very fast response times.)

The product would be ideal from many aspects:
- full upgradeability from dumb to smart
- available in either alpha or graphics
- way to enter PC market
- way to get the largest possible PC base ... 200K terminals in field we instantly try to convert into PC's
- cuts off a whole bunch of competitors to VT100's

Anyway, what you think? Let's see what happens at the meeting.

"TO" DISTRIBUTION:

BILL AVERY
BOB GLORIOSO
AVRAM MILLER
BARRY JAMES FOLSOM
ANDY KNOWLES
DON GAUBATZ
SI LYLE

"CC" DISTRIBUTION:

ART CAMPBELL
JOE MEANY
MARY JANE FORBES
KEN OLSEN
TED JOHNSON
CHARLES A RUSE

Also Strauss

VT/2

%%%%%%%%%%%%%%%%
% d i g i t a l %
%%%%%%%%%%%%%%%%

TO: ART CAMPBELL

cc: see 'CC' DISTRIBUTION

DATE: MON 3 AUG 1981 13:06 EST
FROM: SI LYLE
DEPT: <CPG>
EXT: <264-5001>
LOC/MAIL STOP: <MK1-2>/<2C36>

SUBJECT: CP/M ON VT100

SL1/S1/10

Research has CP/M running on a VT100. Their estimate of transfer cost is $250 to $300. They are working to complete putting the IEEE bus on it. The current requirement is 1 amp and they are fairly confident that the board can be added to VT125 and they will look if it can be added to VT105.

Further to our conversation, the project should be:

1. get option board done ASAP supporting commodity floppies, and

2. get option board with TEAC or RX50 floppies with T-11 and 280 option done ASAP.

Some detail on 1.

The target is to do the add-on board to support commodity floppies. This approach eliminates the need to wait for TEAC/278 or RX50 and FCC.

Barry should get the board into layout ASAP and into manufacturing. We must break all barriers and get it into production so that an inventory is available for a December announcement and ship.

In fact, we should look at the impact on the market and our distribution if we announce in September with December deliveries.

The announcement has got to be:

"Digital announces largest CP/M deal ever." '300,000 VT100's now potentially CP/M machines." Ken Olsen, the President of Digital Equipment Corporation, the President of Digital Research, and the President of Lifeboat seen cutting a ribbon on 'The Deal.' Instant impact on our customers, the software industry, and our competitors. (The VT200 look alikes have to scramble - maybe many locked out because of insufficient power capacity and Xerox left without an encore.)

What you must do -

a) get Barry behind doing the option board, do not let him get all hung up on NIM.

b) get Barry pushing the project through layout and into manufacturing as fast as possible. Let's get the project on a 3

shift, 7 day basis and reward the engineers for working this way.

c) use Gordon and I if any roadblocks occur with manufacturing and customer service. These roadblocks should not occur because I believe Dick Esten, Jay Atlas and Kerry Bensman are as determined to create this business impact as anybody, but they might need help within their organizations.

Some detail on 2.

Two is the follow-up announcement where we include TEAC or RX50 floppies, VT100/floppy FCC systems, and T-11 for RT-11 users (over 50,000). This must follow on (1) as quickly as possible. The floppies are the long lead item so if the T-11 gets done ahead of the floppies, we must take it to market without waiting and make a big splash aimed at the installed base:

'Now VT100 off-loads host - applications run in the terminal.' "Digital makes VT100 into a PDP-11 personal computer." 'For all you RT-11 users - Digital now has a system under $2500.'

As in the case of 1, Barry must also get this board into manufacturing. If we do this right, I see us ending up with two option boards, one Z-80 and the other T-11, both able to support commodity drives or RX50's and both meet FCC in VT100/floppy system configurations.

We must shelve the one board all encompassing approach (Barry's recommended alternative) until 1, and 2, above are completed. Then we can look at the market acceptance and decide how much one plusing we want to do.

If you have any problems getting this started, give me a call.

Si

'CC' DISTRIBUTION:

JAY ATLAS
DICK ESTEN
JACK SHIELDS
*GORDON BELL
BILL HANSON
KERRY BENSMAN
KEN OLSEN

****
* d i s t s l *
****

VT/I

TO: see 'TO' DISTRIBUTION

cc: JAY ATLAS
DICK ESTEN
AL HUEFNER

DATE: WED 5 AUG 1981 15:43 EST
FROM: BARRY JAMES FOLSOM
DEPT: TPG ENGINEERING
EXT: 231-6629
LOC/MAIL STOP: MRG-1/M64

SUBJECT: PROJECT ROBIN STATUS

This morning the Project Robin Task Team met. (Project Robin is the ASAP VT100 CP/M upgrade board.) Key resources are in place, being committed and we are moving ahead rapidly.

Our next milestone is the August 17th Operations Committee review. On this date I would like to address the following three items:

1. Is it feasible, i.e., can it be done by Christmas? What are the risks?

2. What resources and commitments are needed? What's our plan? Can the Organization execute our plan?

3. What is our announcement plan? Can we announce one week after IBM (around August 19th)?

Si/Gordon, does this meet your needs and can you schedule us on the agenda?

Our goal is to do project Robin quick with emphasis on reliability and we want to beat the socks off the Xerox 820 and the IBM Chess.

The assumptions we are operating under are:

1. TPG unique, TPG distributed, and TPG funded.

2. Target FCS for Thanksgiving and ramp by Christmas, 1981.

   NOTE: THIS IS NOT A COMMITMENT THAT WE CAN DO IT, BUT IT'S THE GOAL THE TEAM IS WORKING TOWARDS.

3. First year volume between 13K and 30K units.

4. Assume we have Corporate commitment and that the program is feasible so that we don't lose the next week and a half.

Action to Date:

1. Design being handed over from Research to TPG Engineering.

2. Design team met with FCC expert.

3. Evaluation review set for this Friday.

4. Functional specification review Thursday evening.

5. Disk decision to make 8/12.

6. 20+ actions items plus first pass plans being developed by team.

/p

"TO" DISTRIBUTION:

BILL AVERY	*GORDON BELL	ART CAMPBELL
SI LYLE	KEN OLSEN	

A forecast of the characteristics of computer systems, peripherals, and software during the latter half of the 1970s

The Next (and Last?) Generation

by Frederic G. Withington

The pace of technological evolution in the computer field is as fast as ever. Indeed, it may be accelerating: probably more new computers and computer-related products appeared during 1970 and 1971 than in any comparable previous period. Apparently new generations of computers will continue to appear. The technical staff of Arthur D. Little, Inc., is frequently called upon to forecast technological developments in the various component sectors relevant to computer systems and to advise system manufacturers' product planners. This article contains a distillation of our current thinking, and a view of what the result will be as realized in the computer systems of 1975 or so.

Fig. 1 forecasts the cost of computers (central processors and memories only) during the 1970s. Three bands are shown: large computers (which today cost about $1 million), small but complete business computers (today costing about $15,000), and the least expensive minicomputers (today costing about $4,000). The large computers of 1980 should cost no more than one tenth as much as comparable machines today, because of the extreme drop expected in the manufacturing cost of electronics for both logic and memory. Larger machines than these will exist, costing more, but their power will be so great that few users can be expected to need them. The smaller computers can also be expected to decline in cost, though less sharply; a factor of approximately five can be expected. Late in the decade low manufacturing costs for monolithic circuits will make possible memories cheaper than any today (two cents per bit) with high performance by today's standards, while the highest performance memories will offer cycle times well under 100 nanoseconds.

Fig. 2 (page 73) forecasts the trend of random-access mass storage devices, probably as important as the computer trend for most users. It shows cost per bit as a function of time; average access times are assumed to remain tens of milliseconds (though the more exotic devices may—late in the decade—be much faster).

Rotating magnetic devices today dominate the field, and we believe they still have considerable improvement potential (perhaps another fourfold increase in packing density). A host of new technologies are in development, however, including magneto- and electro-optical technologies, lasers, magnetic bubble and holographic approaches. Each of these has its problems, and in each case after cost-effective devices have been produced new software and system approaches will be required to exploit them. We therefore doubt that they will seriously challenge magnetic devices for another five years, but in the latter part of the 1970s the tremendous performance improvements they promise should be realized in one or more of the competing technologies.

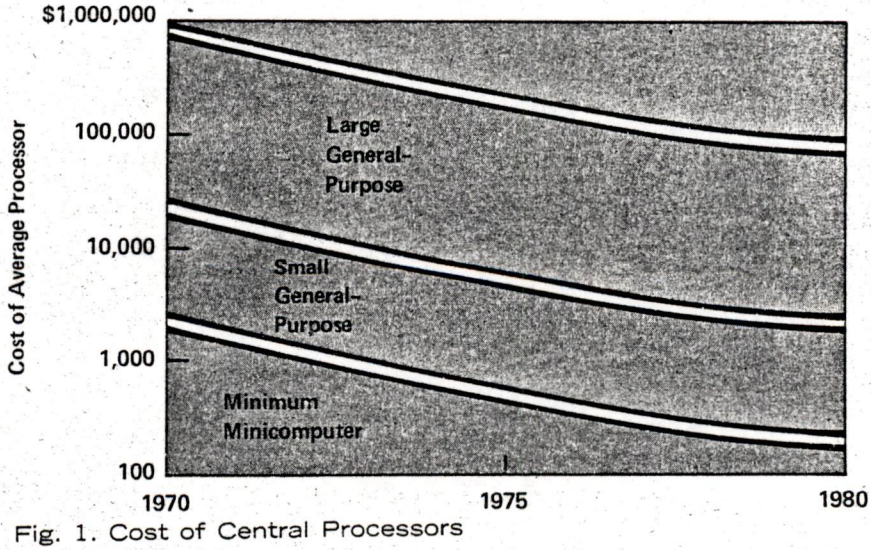


Fig. 1. Cost of Central Processors

Highly mechanical input-output devices (impact printers, tape drives) are already quite mature and are unlikely to improve a great deal: in 1980 they may at best cost one half as much as comparable devices today (in constant dollars). The devices containing a higher proportion of electronics are likely to show greater improvement, however: these include nonimpact printers, cathode ray tube terminals and optical character readers. We believe each of the latter will decline in cost to such a degree that they will be considered inexpensive options for modular terminals. Some manufacturers' announcements already foreshadow the result: a terminal subsystem based on a versatile minicomputer; incorporating small discs, cassettes (eventually monolithic stores) for data retention; employing a crt display and keyboard either for regular use or for system control; several levels of printer; and optical reading modules for data entry and also for facsimile transmission and even office copy. Such terminal subsystems will combine in surprising ways the functions of data processing with those of today's office machines, and will probably cause considerable change in today's office routines.

These impressive improvements in

DATA 100 Model 78, Model 70, and Model 88-23 Terminals are at work right now saving users money and speeding up data communications in talking with 360's, 370's, 6600's, 1108, and Spectras throughout the world. DATA 100, the leading supplier of plug-in replacement Batch Terminals, offers the following products:

Model 70 Remote Batch Terminal truly plug-compatible with 2780 featuring—

- Faster throughput on lines up to 9600 BPS.
- Selection of following peripherals: 300 & 600 CPM card readers, 300, 400, 600 LPM line printers, card punch.
- 15% to 30% savings in monthly rental.

Model 78 Programmed Batch Terminal, plug compatible with 360/20 featuring—

- Simulation of 2780, 1004, DCT 2000, and 200 UT Terminals.
- All Model 70 peripherals plus magnetic tape, paper tape, CRT's and TTY's.
- Capability to concentrate data input from low speed terminals for high speed transmission to central computer.
- 30% to 50% savings in monthly rentals.
- Interleaving data transmission.
- Magnetic tape applications to fit your requirements.

Compat Model 88-23 family of Data Entry Terminals that—

- Validate input data at source and transmit directly to 360/370 central computer.
- Offers optional central off-line pooling of remote data entry terminals, thus freeing CPU of communications processing.
- Offers application software for order processing, billing, inventory control, and many others.

Find out today what DATA 100 can do for you to help solve your data communications problems with quality products, on-time delivery, and competitive pricing—all backed up with an established sales and service organization.

MIDWEST:
Minneapolis, MN 612/941-6500
Chicago, IL 312/992-0850
Detroit, MI 313/642-3383
Lansing, MI 517/489-1700
St. Louis, MO 314/878-4911
Dallas, TX 214/638-7946
Houston, TX 713/772-2483
Dayton, OH 513/278-6723

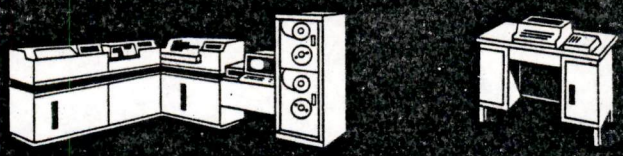
SOUTHEASTERN:
Washington, DC 703/893-4356
Nashville, TN 615/329-3699
Greensboro, NC 919/273-6789
Raleigh, NC 919/782-2185
Atlanta, GA 404/432-7791

WESTERN:
Los Angeles, CA 213/645-4300
Palo Alto, CA 415/328-2100
Seattle, WA 206/228-4770

EASTERN:
New York, NY 212/868-7590
Nutley, NJ 201/667-2960
Philadelphia, PA 215/643-7677
Pittsburgh, PA 412/833-3633
Boston, MA 617/749-2683
Buffalo, NY 716/691-6036
Hartford, CT 203/525-7701

CANADIAN:
Montreal, Quebec 514/842-1721
Toronto, Ontario 416/447-6413

ENGLAND:
Watford, Herts Watford 39611
Hemel Hempstead, Herts
Hemel Hempstead 61711
Poynton, Cheshire Poynton 2129



posting, program compilations, report requests) will be received by a conversational scheduler, which verifies the acceptability of the stimulus and schedules the required operation. The conversation with users will be conducted primarily in an improved command language, simpler and closer to English than the present primitive command languages. New programs, report specifications, etc., will be stated by the users in improved versions of present programming languages.

File organization, memory allocation, linking and all other internal system management functions will be conducted by automatic, universal methods; the user will have no knowledge of the location of data or the detailed status of the system except through monitor programs. This will be inefficient, but the use of hardware assists such as virtual addressing and stacks will improve the efficiency, and improvements in the price-performance of memories and processor electronics will help. The ambitious and skilled user will, of course, have the opportunity to optimize the system's internal operation.

The system will be configured so that it cannot be incapacitated by any single element's failure, and diagnostics will be run automatically to detect failures. Detection of a failure will cause automatic rescheduling so that operations on high priority jobs can continue without pause at a reduced rate.

Stored logic will be used extensively to perform these automatic functions without intolerable loss of efficiency, and will also make possible versatile emulation of past generations of software and hardware—conversion pains should have dwindled to mere twinges.

The peripheral equipment used will (through 1975) consist mainly of improved versions of present devices. The bulk of them will be at remote locations, because fast-response processing will be the primary justification for such systems. In addition to conversational and remote batch terminals we believe there will be widespread use of satellite systems: small (under $5,000 per month) but fully equipped modular systems used most of the time for local processing but connected to the central complex and serving when necessary as terminals.

Fig. 4 depicts the kind of equipment we believe will be employed in the central complex of such a system. Interleaved, separately powered memory banks (for both speed and reliability) will be connected via a network of busses to numerous processing modules. Two or more identical computers will be used, each relatively inexpensive. Any one can arbitrarily be designated to run the scheduler, which will function whenever a new stimulus enters the system to generate queues of interlinked task descriptions that the various processors call upon asynchronously for instructions. Two or more independently programmed input-output controllers will handle local peripherals; communications processors will deal with remote ones; and file controllers will handle file building and reference (the functions of these controllers will probably be combined into multipurpose units of lower speed for less expensive systems). In general, the emphasis will be on the use of multiple relatively low cost units distributing a large workload in parallel fashion, rather than on single fast, sequential units. However, batch applications with lower reliability requirements can still be served adequately by single modules.

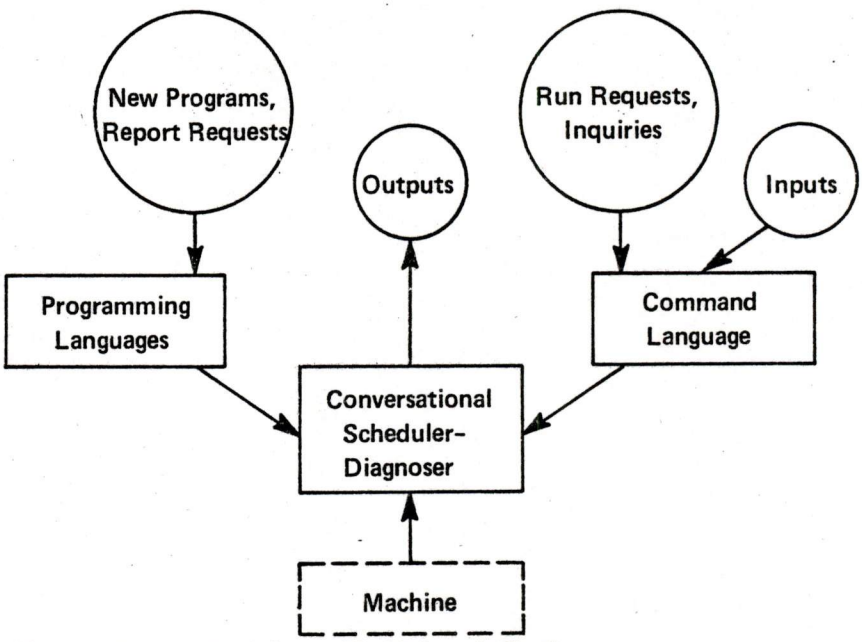
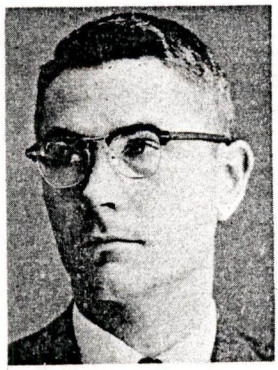


Fig. 4. User Interaction with Network System

The user will probably pay as much for such systems as he does today for systems with comparable throughput; the brute force automatic methods used will offset the lower component costs. However, implementation of advanced information systems should become much easier. Independence of program modules from one another, from files, and from equipment configurations should be complete; evolution of the information system will be facilitated. The same should be true of the equipment. The operating system will permit the network of processing modules to be widely varied; the user will be able to add, subtract, and upgrade them individually, without undergoing the pain of a conversion. The "generation cycles" we have become familiar with should end, and orderly evolution should become the rule.

The user's personnel problems will also be eased. If he is willing to use the methods built into the system he will no longer need system software specialists (though they may often pay their way by achieving higher efficiencies through optimization). Programming will also become easier; programmers will have less to think about in the way of machine constraints, and their individual programs will be more independent. Even system analysis will become a little easier. The systems will accommodate evolutionary change easily, and the costs of adding a forgotten function or superseding an old method will be much lower than they are today. The man-machine interface will have shifted a little closer to the man. □



Mr. Withington has been with Arthur D. Little for over 11 years as a data processing consultant. He was previously with Burroughs Corp. and the National Security Agency. He is a Datamation contributing editor and has published two books on the use of computers in business; a third is nearing completion.

technology will be of only limited value if they are not matched by improvements in software and system design. Indeed, most users' problems with advanced information systems arise more from inadequate software than from any shortcoming in the cost-effectiveness of the hardware. It is therefore of at least equal importance to review the improvements to be expected in software and computer system design.

Perhaps most important, machine independence of programming and operations seems likely. Procedure-oriented languages are already nearly at the point where any program can be compiled and executed on a wide variety of machines. Data management languages have not reached this state but are advancing fast; they seem likely to reach maturity within five years. Command languages (by which computer systems are directed in the operation of completed programs) are still in a primitive state, but the manufacturers have become aware of command languages' importance and are working hard on improving them. Command languages can become machine-independent when the system is capable of automatically scheduling itself and allocating its resources to meet the commands given it; a standard command language can then be applied to any system. A few computer systems already approach this capability, and all may be expected to.

Among their operating system functions, the new machines are likely to offer completely automatic tools for file, communication and input-output management. Completely general methods of performing these functions are already known, but they are rarely used unmodified because of their extreme inefficiency. The general methods are likely to become more sophisticated and efficient. More important, perhaps, the cost-performance improvement in the machines will render the inefficiencies more tolerable. It should be possible for the user of the late 1970s to employ computer systems without ever "lifting the hood"; without knowing (for example) how his files are structured or indexed, or what the machine is doing at any given moment. Of course, those who want to optimize their systems' performance will continue free to do so.

On-line information systems demand a higher level of equipment availability than was needed for batch processing systems; in the worst case, the organization must stop functioning when its machine is down. It seems that the best ways of attaining the required reliability will be found in improved system designs rather than just in component reliability improvement. IBM's 370s and Burroughs' 6700 incorporate diagnostic programs which intermix with the normal job stream. When these are combined with multiprocessing and an automatic self-scheduling capability, we have systems that can truly "fail soft": diagnose, cut out and schedule around any failed module without interrupting operations and continue to function, though at a reduced performance level.

Finally, there is a ferment of new ideas in data capture and data entry. In many cases error control can be assured if the original entry of a transaction is immediately compared to the record to be updated, to a catalog of items and prices, to range and reasonableness guidelines, or to a combination of these. Where this can be done, the costly key verifying operation can be eliminated. The self-scheduled system can accommodate the necessary interrupts: in fact, the data entry and inquiry functions merge (as they already do in an airline reservation system). Lower data entry cost and improved responsiveness appear to be arriving together.

How will these technological and systems improvements be combined into the next generation of computer systems? We think the more advanced systems of today point the way clearly, and that the next generation will have the following characteristics.

Fig. 3 depicts the system (called a "network system" for convenience) as it appears to the user. It operates entirely in higher level languages, and is entirely self-scheduling and self-organizing in the sense of memory, file and communication management. All stimuli presented to the system (run requests, inquiries, transactions for

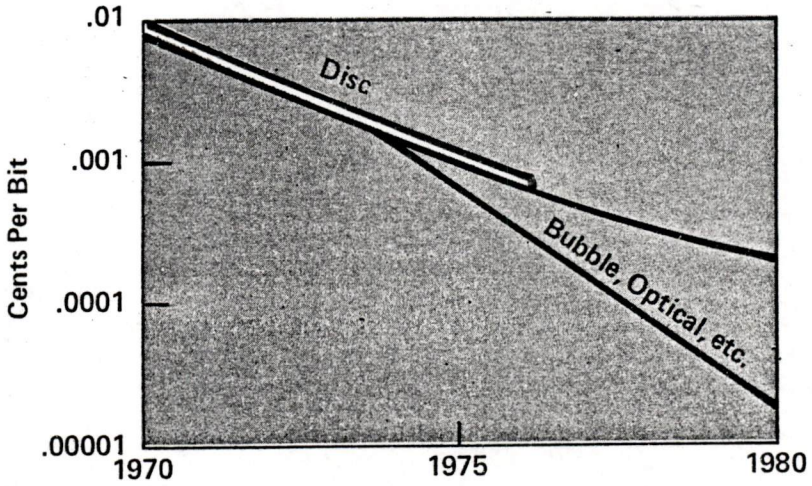


Fig. 2. Cost of Mass Storage

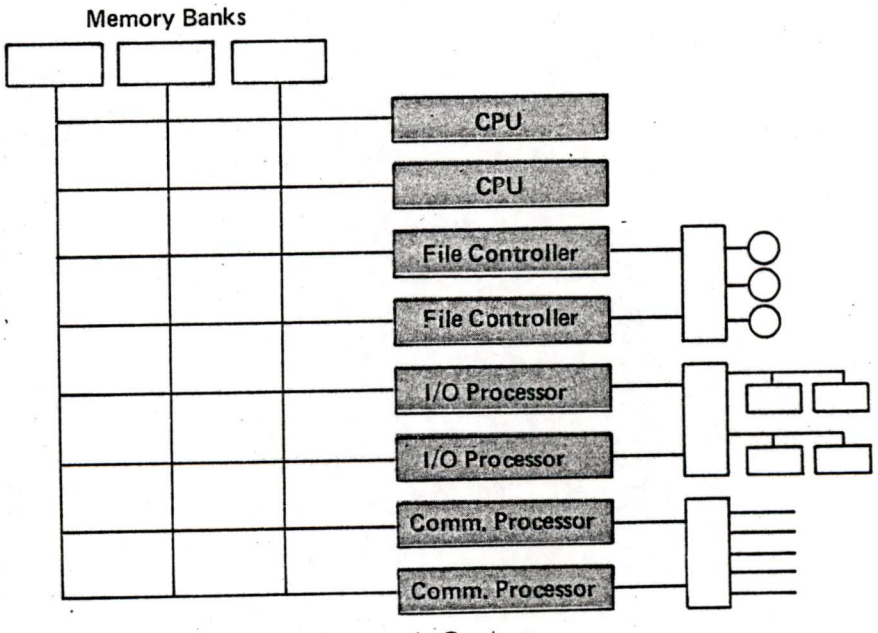


Fig. 3. Components of Network System

Strauss 9/79

OBJECTIVES

1. GET EVERYONE EXCITED ABOUT THE INTEGRATION OF COMPUTATIONAL NEEDS WITH PUBLIC NETWORK ACCESS AND TO IMMEDIATELY INTEGRATE THESE INTO A "PERSONAL" COMPUTER PRODUCT.

2. UNDERSTAND DIGITAL'S POSITION RELATIVE TO THESE PRODUCTS.

3. EVOLVE A STRATEGY FOR DEVELOPMENT OF SOLUTION ORIENTED PRODUCTS WHICH INTEGRATE STAND ALONE COMPUTER USAGE WITH PUBLIC NETWORK ACCESS AND CAPTURE THIS MARKET.

TERMS

SOLUTION: A SYSTEM WITH ALL THE SPECIFIC CHARACTERISTICS NECESSARY TO BE MARKETABLE (AND COST EFFECTIVE) TO THE SOLUTION'S POTENTIAL CUSTOMER. (NOT JUST PRODUCT, BUT SUPPORT, TRAINING, DOCUMENTATION ETC.)

PERSONAL COMPUTER: A SINGLE USER SYSTEM MARKETED AS A COMMODITY.

THE NON-TECHNICAL USER WANTS SIMPLE ACCESS TO MANY SOURCES OF INFORMATION FROM HIS TERMINAL. THOSE WHO SEVERELY LIMIT ACCESS BY TYING A TERMINAL TO A SPECIFIC INFORMATION SOURCE WILL SOON LOSE MARKET.

THE STRATEGY IS TO TIE A NUMBER OF SOURCE INDEPENDENT ACCESS (BOTH ACCESS TO DATA AND ACCESS TO COMPUTATION) TOOLS TOGETHER WITH LOCAL COMPUTATION CAPABILITIES IN ORDER TO GIVE REASONABLE PRICE PERFORMANCE IN:

SMALL BUSINESS

DESKTOP USE FOR LARGER BUSINESSES

PART OF THIS PRODUCT IS DEPENDENT ON ACCESS THROUGH PUBLIC NETWORKS TO PUBLIC AND PRIVATE DATA BASES.

LOOK AT WHAT TELENET OFFERS TODAY THAT, WHEN INTEGRATED WITH PROPER COMPUTATION AND INFORMATION HANDLING WOULD BE SALEABLE IMMEDIATELY.

SOME PUBLIC INFORMATION AVAILABLE TODAY FROM TELENET

AMERICAN HOSPITAL SUPPLY ORDERING

MAJOR BIBLIOGRAPHIC DATA BASES

BOOK ORDERING

REAL TIME PRICING FOR BONDS/STOCKS/OPTIONS/AND COMMODITIES

ELECTRONIC MAIL

CLASSIFIED ADS

SPORTS SCORES

RESTAURANT/THEATRE GUIDE

HOROSCOPE/BIORHYTHMS

GLOBAL WEATHER

NEW YORK TIMES DATA BASE

AIRLINES SCHEDULES

FUNDING ORGANIZATIONS/SOURCES OF GRANTS

HISTORICAL AND FORECAST DATA FOR INDUSTRIES, PRODUCTS, AND WORLDWIDE END USE.

LEGISLATION

FINANCIAL MODELING AND ANALYSIS

EDUCATIONAL PACKAGES

TWO EXAMPLES:

PHARMACY:

LOCAL: DATA BASE/RECORDS
INVENTORY
ACCOUNTS RECEIVABLE
ACCOUNTS PAYABLE
LABEL PRINTING
CONTROLLED DRUG ACCOUNTING
PAYROLL

PUBLIC: ORDERING OF SUPPLIES
ADVERTISEMENT OF NEW PRODUCTS
SHORTEST - ROUTE CALCULATIONS FOR DELIVERIES
INCOME TAXES
STOCKS/BONDS
ELECTRONIC FUNDS TRANSFER
BLUE CROSS/BLUE SHIELD/MEDICARE REPORTING
DRUG INTERACTIONS

YOUR DESK:

LOCAL:
- DRAFT PREPARATION
- CALCULATOR
- CALENDAR
- TICKLER
- ORGANIZATION CHARTS
- PRESENTATION PREPARATION
- PHONE LOG
- PHONE DIRECTORY

PUBLIC:
- AIRLINE SCHEDULES
- ELECTRONIC MAIL
- BIBLIOGRAPHIC SEARCH
- WEATHER
- RESTAURANT GUIDE
- NEW YORK TIMES DATA BASE
- CURRENT GRANTS
- LEGISLATION
- CURRENT PRODUCT COST/SHIPMENT/REVENUE DATA
- MARKETING RESEARCH DATA

WHERE IS DIGITAL?

PLUSES:

- WE HAVE THE CONCEPT
- WE HAVE THE TOOLS
- WE HAVE THE TECHNOLOGY
- WE HAVE THE DESIRE

MINUSES:

- WE DO NOT HAVE THE SOLUTIONS OR SOLUTION BUILDERS
- WE DO NOT YET HAVE THE PROPER DISTRIBUTION CHANNELS
- WE DO NOT HAVE THE CORRECT ORGANIZATION

IN THE RED BOOK STRATEGY GORDON WROTE:

"PROVIDE GENERAL APPLICATIONS-LEVEL PRODUCTS THAT RUN ON 8, 10/20 AND 11/VAX-11 ABOVE THE LANGUAGE-LEVEL TO MINIMIZE USER COSTS, INCLUDING:

- WORD PROCESSING, ELECTRONIC MAIL, PROFESSION-BASED CRT-ORIENTED CALUCULATORS;
- TRANSACTION PROCESSING AND DATA BASE QUERY;
- GENERAL LIBRARIES, SUCH AS PERT, SIMULATION, ETC. AIMED AT MANY
- PROFESSIONS THAT CROSS MANY INSTITUTIONS (INDUSTRY, GOVERNMENT, EDUCATION, HOME); AND
- GENERAL MANAGEMENT LIBRARIES FOR VARIOUS SIZED BUSINESS.

PROVIDE SPECIFIC PROFESSION (E.G. ELECTRICAL ENGINEERING, ACTUARIAL STATISTICIAN) AND INDUSTRY (E.G. DRUG DISTRIBUTOR, HEAVY MANUFACTURER) PRODUCTS AS NEEDED VIA THE PRODUCT LINE GROUPS."

YET THERE IS LITTLE OR NOTHING IN PROCESS TO ACCOMPLISH THIS OBJECTIVE. LET'S LOOK AT POSSIBLE SOLUTION BUILDERS:

POSSIBLE SOLUTION BUILDERS - CONS

1. PRODUCT LINES:
    - SOLUTION SPACES DO NOT FOLLOW SIC CODES
    - CAUSE INDEPENDENT PRODUCTS LIKE WP.
    - GET BETTER ROI ON LARGER SYSTEMS.

2. ENGINEERING
    - TOOL ORIENTED
    - FOR BUILDING SOLUTIONS IT IS MORE IMPORTANT TO KNOW USAGE THAN TECHNOLOGY (GRADED ON USABILITY, NOT ELEGANCE)
    - TOTAL BUSINESS MODELS NECESSARY

3. CSS AND PL90
    - HARD TO COORDINATE
    - DIFFICULT TO GENERALIZE SOLUTIONS
    - NEED TOTAL BUSINESS MODELS

4. "VENTURE" CAPITAL
    - CAUSES UNIQUE PRODUCTS LIKE WP
    - HARD TO STRATEGIZE
    - CUASES TOO MUCH INTERNAL COMPETITION

5. "INTERNAL" PRODUCT LINE:

- A SIGNIFICANT CHANGE IN THE WAY PRODUCT LINES ARE GRADED
- CAN EASILY BE PUSHED IN TOO MANY DIRECTIONS AT THE SAME TIME
- WHAT HAPPENED WITH GRAPHICS?

6. OEM'S

- TOO LOW PRICE FOR SMALL OEM'S TO MERCHANDISE
- LITTLE CONTROL ON IMAGE
- LOSE A GREAT DEAL IN FOLLOW-ON OR CENTRAL FACILITY BUSINESS
- THIS IS GOOD BUSINESS WE COULD GET FOR OURSELVES
- IF THIS IS A GOOD SOLUTION WHY ISN'T IT WORKING ALREADY?

7. OTHER?

- COMPLETE BUSINESS MODEL
- ABLE TO DEVELOP STRATEGY
- MUST KNOW CUSTOMERS NEEDS INTIMATELY
- PARTITIONED INTO USER SPACES
- CAUSE "CORPORATE" SOLUTIONS

MY SUGGESTED STRATEGY

IN CENTRAL ENGINEERING SET UP SWAT (SOLUTIONS BY WORKING AS A TEAM) GROUPS.

EACH HAS 4-7 MEMBERS

1- PRODUCT BUSINESS MANAGER

- DEVELOPS BUSINESS MODEL (INCLUDING SUPPORT/SALES)
- DEVELOPS BUSINESS PLAN
- IS RESPONSIBLE FOR PRODUCT'S BUSINESS SUCCESS

1- USER

- SOMEONE WHO HAS USED THE APPLICATION
- HAS DEVELOPMENT EXPERIENCE
- IS RESPONSIBLE FOR PRODUCTS USABILITY
- IS RESPONSIBLE FOR PRODUCT TESTING

1- DOCUMENTER

- IS RESPONSIBLE FOR MANUALS
- IS RESPONSIBLE FOR USER TRAINING
- IS RESPONSIBLE FOR ALL DISPLAYS
- IS RESPONSIBLE FOR PRODUCT MESSAGE

1-4 DEVELOPERS

- SOME HARDWARE AND SOFTWARE CAPABILITY
- PROBLEM SOLVING RATHER THAN TECHNOLOGY ORIENTATION
- RESPONSIBLE FOR PRODUCT'S QUALITY AND RAMP
- RESULTS ORIENTATION

EACH SWAT GROUP HAS A PRESET LIFETIME OF BETWEEN 6-18 MONTHS.

SUPPORT HANDLED ELSEWHERE. (SHIELDS)

NEED A CENTRAL ORGANIZATION TO STRATEGIZE AREAS OF ATTACK AND TO ADMINISTRATE/MOTIVATE/EVALUATE SWAT GROUPS.

PROPOSAL

$.7 MILLION FY80

$1.2 MILLION FY81 GIVE YOU 5 SALEABLE SOLUTION PRODUCTS (5 SWAT TEAMS).

THESE WILL BE MARKETED BY ALL PRODUCT LINES WITH SPECIAL INTEREST FROM:

RETAIL PRODUCTS GROUP

COMMERCIAL OEM

SMALLER COMMERCIAL END USER PRODUCT LINES (TURNER)

AND BE THE BASIS OF ESTABLISHING DIGITAL IN THE PERSONAL COMPUTER MARKET.

OTHER POTENTIAL DISTRIBUTION CHANNELS TO BE EXPLORED:

PUBLIC NETWORKS

TERMINAL DISTRIBUTORS

TODAYS INVESTMENT IN PDT SOFTWARE IS:

$1.15 MILLION FY80

$1.38 MILLION FY81

WITH NO END USER PRODUCTS.

Shaw

EXAMPLE PERSONAL COMPUTER APPLICATIONS

NO PRIORITIES

ELECTRONIC MAIL RELATED APPLICATIONS:

- WORD PROCESSING
- DATA ENTRY - FORMS PROCESSING
- DRAFT ENTRY
- DISPLAY REPROCESSING
- CALENDAR
- TICKLER FILE
- MAIL ENTRY
- MAIL PICKUP
- RUNNING MONTHLY REPORT
- SCHEDULING MEETINGS
- AIRLINES/HOTEL RESERVATIONS
- WEATHER REPORTS
- CALCULATOR
- STOCKMARKET
- INTERNATIONAL TIME
- PHONE BOOK (DIRECTORY)
- CURRENCY CONVERSION
- LIBRARY BOOK ORDER
- HEALTH INSURANCE FORMS
- TRIP EXPENSE REPORTS
- SYSTEM "HELP" MESSAGES
- ORDER ENTRY
- "TROUBLE DESK SYSTEM" (E.G. FIELD SERVICE AT EACH UNIT WITH ELECTRONIC MAIL TO LARS)
- ROI CALCULATIONS

GAMES

- TRIVIA QUIZ
- CHESS
- BRIDGE
- ART
- BIORHYTHMS
- ADVENTURE
- STAR TREK
- CARIBBAGE
- BACKGAMMON
- SLOT MACHINE
- ETC.

OTHER APPLICATION AREAS:

- PROGRAMMER'S WORKBENCH
- SLIDE PREPARATION
- TABLE TO GRAPHICS CONVERSION
- ACCOUNTS RECEIVABLE
- ACCOUNTS PAYABLE
- "CHECKING ACCOUNT"
- INVENTORY
- KEYLESS ENTRY
- HANDICAPPED PERSON TERMINAL
- EDUCATIONAL MACHINE
- INCOME TAX PREPARATION
- ELECTRONIC FUNDS TRANSFER
- RECIPES/DIET
- MEDICAL CONSULTING
- REAL ESTATE
- SPORTS SCORES
- BUY BY COMPUTER ADVERTISING
- LOTTERY
- RESTAURANT/BOOK/MOVIE REVIEWS
- AGRICULTURAL INFORMATION
- CAREER/PERSONNEL PLACEMENT
- FIRST AID
- MAPS
- CONSUMER INFORMATION
- CREDIT CHECKING
- COMPUTER CONTROLLED MICROFILM RETRIEVAL
- CRYPTOLOGY
- NAVIGATION
- COMMUNICATION/INFORMATION CENTER
- ENVIRONMENTAL CONTROL SYSTEM
- SECURITY SYSTEMS
- STATISTICAL PACKAGES
- CASH REGISTER

DIGITAL EQUIPMENT CORPORATION INTEROFFICE MEMORANDUM

TO: George Poonen
Rick Peebles
Chuck Kaman

DATE: 12-Nov-79

FROM: Ed Lowry

DEPT: Corp. Research Group

EX-MS: 223-7686 ML3-2/E41

CC: Gordon Bell
Dave Butchart
Bruce Delagi
Ulf Fagerquist
Bob Grimes
Per Hjerppe
Jan Jaferian
Bob Lynch
James Nickson
Jeff Rudy
Dick Snyder
Ollie Stone
Corporate Research Group
RAD Committee

SUBJECT: Self-Teaching Profession Based System - A Research Proposal

I think Dawn can provide a sound architecture for the user interface of a Profession Based System like the one described by Gordon Bell on November 1. Reasons why include the following:

1. A basic requirement is that the system lead the user on how to do things while leaving the user fully in charge of what is to be done. Doing that with a reasonable amount of software requires that knowledge of the problem environment be shared by the user and the system and be coded in a form that is usable by both. This way the system can make "intelligent" suggestions and the user can make high level decisions. The declarations of Dawn are at the leading edge of this capability. They make it possible for Dawn's interactive facilities (as now planned) to provide users with a substantially higher level of control than alternative systems. They also contribute to the natural language style and the elimination of unnecessary proceduralness.

2. The professional will need highly readable, evolvable application packages. Ninety-five percent of application programs continue to be written or adapted by someone who understands the specific end user's needs. Dawn seems to have a strong lead in that area.

3. A high degree of consistency in the interface is needed while supporting a variety of professional users. This requires much generality. The freedom from representa-

tional irrelevancies enables Dawn to be substantially more general than other candidates.

4. Professionals will access information from many sources through heterogeneous networks. Since representational irrelevancies are compounded when viewing data through multiple interfaces it is important to use a language like Dawn which minimizes that problem. The canonical form of Dawn data looks like the best way to transmit data between systems because it is self-describing, strongly typed, machine indpenedent, implementation independent, and it accurately describes any kind of digital data without adding to its representational irrelevancies.

5. The editor generator facility of Dawn can make good use of the declarations which describe the permissible structure of the problem environment to easily customize editors which allow the users to easily manipulate models of the environment and other data.

Taken as a whole I think Dawn is very far ahead of whatever is in second place as an effective basis for the user interface of a Profession Based System. I would be interested in hearing of contrary views. I recommend the use of Dawn in the Profession Based System and appropriate integration of projects.

One desirable characteristic of the Profession Based System which was emphasized by Gordon Bell but that Dawn does not satisfy is that the system have zero manuals. I think this is an important and difficult problem, and the most significant research challenge presented. Again the generality of Dawn provides major advantages in reducing the stack of manuals and improving ease of learning over other possible candidates but by itself it is unsatisfactory for untrained users with sizable tasks. I therefore propose that a project be started soon to make the Dawn system self-teaching and to enhance the self-teaching capabilities of applications written in it. The main goal would be to develop facilities which teach the user just what he needs to know when he needs to know it.

The declarative style of Dawn helps the prospects for success. One way is that both questions and answers on both sides of the dialog can deal with relatively small pieces of information which are mutually intelligible. In adddition the system can use whatever declarations it has to specialize its communications in terms of the user's problem.
The following approaches may be developed:

1. Provide a parameterized form of the language manual internally, and an associated program analyzer so the user can point to any line or phrase in an application program and get a carefully specialized statement of what it means geared to his level of expertise.

2. Develop some sample application programs in a style that makes their utilization as self-explanatory as possible and develop guidelines and other tools to encourage this quality in later application programs. The editor generator and report generator may be expanded to provide more control over levels of explanation provided with application outputs and requested inputs. Answers to natual language questions about an application may be attempted by extracting apparatently relevant statements from a full description of the application. The relevance may be judged by word associations through a thesaurus.

3. Develop a semi-automated on-line diagnostic and user assistance facility. A user who is having serious difficulty, or who does not mind paying for the service, should be able to quickly communicate his situation to a system expert (local or remote) for consultation. Formalized methods for expressing the user's distress and the consultant's replies should be developed. Like telephone operators, the role of such consultants should increasingly be automated, but not entirely.

4. Enhancements to the natural language style of Dawn should be provided. This may include langauge changes but the emphasis would be on a front end analyzer which extracts all the useful information it can out of intuitively expressed statements by users. The analyzer should interactively resolve ambiguities, confirm intended meanings, and optionally teach the correct forms.

5. An interactive statement building facility is needed which responds to "How do I...?" questions and responds with menus of possible approaches. It should explain the possible approaches as needed, and recursively present skeleton statements and expressions to be filled in. Menus of plausible things to be inserted should be presented based on whatever knowledge of user's problem environment has been previously declared.

6. A book may be published in both paper and electronic versions which provides Dawn declarations describing a wide variety of plausible professional environments and many of the functions the professional can use to manipulate models in those environments. When installing his system the professinal would have a dialog with the system answering questions about the major parameters of his environment as compared with those typical of his field. The system would select, specialize, and explain the resulting proposed initial data base description and whatever usable functions can be provided that way. During the dialog the user could watch the way in which his answers affect the form of the declarations and interrupt with questions about the meaning of the variant

forms.

7. An interactive program customizng facility could operate by explaining the program using a mixture of Dawn and English (or French etc.). It could be dynamically adjusted to the user's improving skill. The user could interrupt with questions for clarification and complaints when the system description looks inappropriate for his need. The system would then attempt to propose variations and pose clarifying questions.

8. A number of subsets of the language and interactive facilities may be defined. The user could specify what subset he would like to restrict himself to. All diagnostics and explanatory messages would try to respect such limits.

Before delimiting the scope of such a project, I would like to see some thought given to the probable effect of our overall expertize in this area on DEC's profitability during the 1980's. I would welcome suggestions, prioritizations, counterproposals, and other comments from any source.

Gordon Bell

ML12-1/AS1