Table of Contents

Preface

The major contributors to this user guide were: Chris J. Barter, Chris D. Marlin, Kelvin B. Nicolle, Michael A. Petty, M. Fiona Symon and Francis A. Vaughan.

Ludwig is a text editor developed at the University of Adelaide, designed and implemented by: Wayne N. Agutter, Chris J. Barter, Jeff A. Blows, Bevin R. Brett, Peter J. Cassidy, David A. Hodson, Kevin J. Maciunas, Kelvin B. Nicolle, Blair A. Phillips, Mark R. Prior, and Francis A. Vaughan

The comments of the several people who read early drafts of this manual are gratefully acknowledged. This manual is intended for both novice and experienced users of Ludwig.

Conventions Used in This Document

RETURN	Keyboards have standard key names, used in this guiode
Control/Z	Indicates that the Control key should be held down while the Z key is pressed.

Before reading this user guide, you should examine the keyboard of the terminal you will be using, and find where the RETURN, DELETE, TAB, Control and arrow keys are located.

1. Introduction to Ludwig

Ludwig is an interactive, screen-oriented text editor. It may be used to create and modify computer programs, documents or any other text which consists only of printable characters. During an editing session, the text is displayed on the terminal screen and the user may type in further text and enter commands to modify the text. The changes made to the text will in most cases be immediately visible on the screen. Ludwig may also be used on hardcopy terminals or non-interactively, but it is primarily an interactive screen editor. Ludwig may be used under both the VMS and Unix operating systems.

Ludwig is named after Ludwig Wittgenstein, a philosopher who said that one can indicate an object either by a detailed description or by pointing at it. The Ludwig editor allows one to indicate a specific piece of text either by a complex description or by moving the screen cursor to point at it.

Ludwig offers many features to make text editing easy and convenient. These features include the following:

. An on-line Help facility. This may be used to obtain help without losing the context of the editing session.

. Command procedures. Procedures may be written to automate many editing tasks. These may also be saved for use in later editing sessions.

. Initialization command procedures. These enable certain characteristics to be set up automatically at the start of every editing session, thus allowing tor personalization of the editing environment.

. A key definition facility. Keys may be defined to execute editing commands and procedures, further personalizing the editing environment.

. Multiple environments. More than one editing environment may exist at any one time, and it is possible to switch environments, move text between environments, and access different files from different environments.

Whilst it is possible to write programs and prepare documents with only a minimal knowledge of the editor, a reasonable level of proficiency can be attained in a short time and will give an increase in the speed and efficiency with which editing tasks may be accomplished.

1.1 Files and Editors

This section is aimed at those readers who are learning to use an editor for the first time. If you have already had some experience in using a text editor to create and modify files, you may wish to proceed directly to Section 1.2.

A file may contain personnel records, data from an experiment, a program, an essay, or any other collection of information. The file is given a name, usually indicative of the type of information it contains. An *editor* is a tool which may be used to create and name files, and to revise them.

Such files contain information which can be displayed directly on a screen or on paper and are called text files. As the name suggests, the information is made up of those symbols we normally associate with printed text. These symbols are called printable characters, and are generally those which appear on the keys of a typewriter. These are the upper and lower case letters (A-Z and a-z), digits (0-9), the blank space, punctuation marks . , ; : ! ? ' "

and other symbols + - * / = < > ( ) [ ] { } $ % # @ & A ~ \ _ |

There also exist other characters which are called non-printable characters. See Appendix C for a table of the complete ASCII character set, which contains both printable and non-printable characters.

The characters in a text file are grouped together into lines. These lines may be of different lengths. A text file has a beginning and an end. The text on the screen looks very much as if it were printed on a piece of paper (often a very long one), and so the beginning and end of the file are often also referred to as the top and bottom, respectively.

An editor which can be used to create and modify text files is known as a text editor. Ludwig is an interactive, screen-oriented text editor. The editor displays the contents of the text file on the terminal screen, and the person using it interacts directly with the text, and any additions, deletions and modifications appear on the screen as they are made. While some text editors can, in fact, handle files which contain non-printable characters, Ludwig will remove any such characters it encounters in a file.

From this user guide, you will team how to use the text editor Ludwig to create new files with a given name, to enter text into such files, which can then be stored, and also to modify existing text files. Tasks which are mechanical and prone to human error can be done quickly and accurately using the editor. This enables you to focus your attention more on the information to be contained in the file, rather than on the means of putting it there.

1.2 Invoking and Terminating Ludwig

Ludwig may be used to edit an existing file, or to create a new file. To invoke Ludwig, enter the command

ludwig *filename*

where *filename* is the name of the file you wish to modify or create.
If the file already exists, this will display the beginning of the file on the screen and allow you to edit it. The end of the file will be marked by a line which looks like this:

<End of File> **LUDWIG**

If the file contains more text than can fit on the screen, you will not see this line at the start of the editing session.

If the file does not exist, an empty tile will be created. Thus the line marking the end of the file will appear at the very top of the screen, showing that the file contains no text. Text may be entered simply by typing, and the <End of File> line will move down to remain at the end of the file. The text so entered may also be edited, just as if it had been present in an existing file.

To terminate an editing session, enter the Ludwig Quit command by typing \Q. If a new tile was created in the session, it will be saved under the name you gave it when you invoked Ludwig. If an existing file was modified, a tile containing the edited version will be created. As the old version is not removed, there must be some way to distinguish the new version from the old. This is done differently depending on the operating system under which you are working.

Under the VAX/VMS operating system, different versions of the tile are distinguished By a *version number*, which is separated from the rest of the filename by a ";". The first time a file with a certain name is created, it is given a version number of "l", unless you explicitly give it a different version number. Subsequent versions are given successively increasing numbers. So, for example, if you edit filename;2 the resulting file will be filename;3. You do not have to specify the version number when you invoke Ludwig, although you may if you wish. If you do not, the file with the highest version number will be edited.

Under the Unix operating system, a different scheme is used. The file created during the editing session is given exactly the name you specified when you invoked Ludwig, no matter whether you were creating the file for the first time, or modifying an existing file. If the latter is the case, then the previously existing file is then distinguished from the new file by a "~" at the end of the filename, followed by a number. As under VMS, successively higher numbers are given to subsequent versions of the file. However, by default only the two most recent versions of the file are kept, namely, that which does not have "~" and a number at the end of the filename, and that which has the highest number following the "~". It is possible to specify that more versions should be kept. (See Section 11.2)

Under either operating system, if you wish the modified version of a file to have a completely different name, rather than just being distinguished by a number, you may specify this when Ludwig is invoked by entering

ludwig *old-filename new-filename*

rather than just

ludwig *fileneneme*

Under both operating systems, messages will be displayed on the screen whenever you quit Ludwig, giving the name of the old file, if there was one, and of the file just created.

Ludwig "remembers" the name of the file which was previously created. To edit the same file subsequently, the filename may be omitted when entering the command. Thus, you need only type

ludwig

Each time you edit a different file, this name is then the one stored. Under the VMS operating system, the name is lost once you log out of the computer. However, under the Unix operating system, the name is stored even when you log out.

1.3 Using the On-Line Help Facility

If, during the editing session, you require help on a command or a list of the available commands, the on-line help facility provides a means for doing this without losing your place in the text. Enter the Ludwig Help command by typing \\H. This will cause the prompt

Topic :

to be displayed. If you do not know the topic on which you want help, or need further information, press Return. The text on the screen will be replaced by a menu of the available topics, preceded by instructions on how to access them. In addition to these topics, help is available on every Ludwig command and family of commands. This may be obtained by typing the name of the command in response to the prompt.

When you have finished with the Help facility, pressing Return will restore the screen to the state it was in at the time the Help command was entered.

2. Basic Editing Commands

It is intended that you read this chapter whilst logged in at a terminal. In this way, you can try out each new feature of Ludwig as it is introduced. If you have access to the Interactive Ludwig Tutorial, this will give a more vivid introduction than the printed manual. However, reading this chapter in conjunction with using Ludwig is still beneficial in learning to use the editor.

2.1 **Entering Text**

During an editing session, the current editing point is indicated by the cursor. Typing text at the keyboard causes the text to be entered at the cursor position. If you are creating a new file or adding to the end of an existing one, simply type, pressing the Returm key to begin a new line.

Pressing the delete key causes the last character typed to be deleted. The cursor is left in the place of the deleted character. In this way, any number of characters to the left of the cursor can be deleted.

Normally when Ludwig is invoked, text is entered in what is called keyboard insert. In this mode, text entered at the current cursor position is inserted into the existing text, pushing the existing text aside. The alternative method of keyboard text entry is keyboard overtype which overtypes any text already in that position. The relative merits of the two methods of text entry and the way to switch between them will be discussed in Section 2.5.1.

2.2 Moving the Cursor

You may wish to enter text at a position other than that at which the cursor is currently located. There are many different ways of moving the cursor around the file. You have already seen that pressing RETURN moves the cursor to the start of the next line. The four arrow keys move the cursor one character in the direction indicated.

If you try to move beyond the end of file marker, above the first line of the text, or past the Left hand edge of the screen, Ludwig will ring the terminal bell. This is merely to let you know that you cannot proceed any further in that direction. If you try to move beyond the right hand edge of the screen, the behaviour is different, as will be explained in Section 2.4.

The TAB key can be used to move the cursor to the right in jumps. If your terminal has a BACKTAB key, this will move the cursor to the left in jumps. The cursor jumps to predefined tab stops. They are normally set at the first character position and every 8 character positions thereafter. However, tab stops may be set in any position across the screen.

2.3 The Command Introducer

Normally Ludwig assumes that anything you type is to be entered as part of your text. However, you may want to instruct the editor to take some action other than merely entering the typed text. In order for Ludwig to recognize commands as distinct from text, you must press a special key before typing the command. This is known as the command introducer. By default this is the backslash \ , but it may be redefined to be any key which is not normally used to type a letter or a digit. However, \ will be used throughout the first part of this user guide to indicate when the command introducer should be pressed. Further into the user guide use of the command introducer will be assumed.

You will remember the commands Quit (Q) and Help (H) introduced in the previous chapter Ludwig commands consist of one, two or three letters, depending on the command, and in most cases this is indicative of the type of command. Q for Quit and H for Help are good examples. The command may be typed in either upper or lower case, although all commands will be shown in upper case in this user guide.

When the \ is typed, it is not echoed on the screen, and neither is the command which follows. As the command introducer is used by Ludwig to recognize commands, if you wish to have it appear in the text, it must be entered with the Insert Command Introducer command (UC).

2.4 Window Onto the Text

Ludwig views the text being edited through the screen of a terminal (normally an area 80 characters wide and 24 lines high). If the text is too long or wide to fit in the window, it can be moved to the part you wish to see.

You may have noticed that trying to move the cursor off the right hand edge of the screen caused the window to move to the right, so that the area past the right hand edge became visible. If your text is longer than the height of the window, moving the cursor to the bottom of the screen causes the window to move downwards.

It is also possible to move the window explicitly. The command Window Forward (WF) scrolls the window forward over the text by the window height (which is usually 24 lines). Similarly, Window Back (WB) scrolls the window backwards by the window height. Window Right (WR) and Window Left (WL) move the window half the width of the screen (usually 40 character positions), to the right and left respectively.

When editing long files, it is useful to be able to jump to the beginning and end of the text. This is done with the commands Window Top (WT) and Window End (WE), respectively.

Some of the above commands may be entered in a different manner. This is done by holding down the Control key while typing another character, as shown in the following table.

Command	Meaning	Control Code
WF	Window Forward	Control/F
WB	Window Back	Control/B
WT	Window Top	Control/T
WE	Window End	Control/E
WN	Window New	Control/N

The commands which cannot be entered in this alternate manner are Window Left and Window Right.

2.5 Inserting and Deleting Text

2.5.1 Overtype and Insert Text Entry

When text is entered using Ludwig, the new text may either overtype any text already at that position, or the existing text may move aside so that the new text is inserted before it. These two methods of entering text are known as keyboard overtype and keyboard insert.

There are arguments for and against either, but there are certain situations where one is clearly preferable. For example, when adding entries to an existing table with fixed columns, such as a weekly timetable, use of overtype will ensure that the columns remain aligned. However, most users find the behaviour of keyboard insert more convenient when drafting a document, and it has the advantage of making it more difficult to accidentally overtype, and hence lose, existing text.

Keyboard insert is the default in Ludwig. Inside any editing session the it is possible to change from keyboard insert to keyboard overtype and vice versa with the commands \O and \I, for keyboard overtype and keyboard insert respectively. It is also possible to change the default setting at startup. This is described in Section 3.

If you are working with keyboard overtype and wish to insert some text you may change to keyboard insert, then type the required text and revert to keyboard overtype. Alternatively you can use the Insert Character (C) command to insert spaces to the right of the cursor and type over the spaces so inserted.

2.5.2 Deleting Text

Deletion of text is analogous to its insertion. The same units of text may be deleted as inserted, and Ludwig has commands for deleting characters and lines, from which commands to delete other units of text may be constructed by the user.

As with the insertion of characters, the procedure for deleting characters varies depending on the current mode of text entry.

Using keyboard insert, characters to the left of the cursor may be deleted by pressing the Delete key. As a character is deleted, the characters to the right of it in the line will move to the left to take its place. Note that when using the Delete key, you always position the cursor on the character to the right of that which you wish to delete, not on the character to be deleted itself.

Using keyboard overtype deletion of characters is accomplished slightly differently. If youposition the cursor on the character to the right of those to be deleted and use the Delete key,the characters are deleted and replaced by spaces. You may now type over the spaces in theusual way. If you wish to delete characters entirely, rather than having them replaced by spaces,use the Delete Character (D) command. To delete a single character, position the cursor on thatcharacter, and enter \D (Delete Character). The part of the line to the right of the deletedcharacter will move to the left to fill the space.

The cursor will be positioned on the character which was immediately to the right of the deleted character. Thus, when using this method todelete multiple characters, you must remember that the deletion proceeds from left to right,rather than from right to left, as when using the Delete key in insert mode.

Line deletion is done in the same manner in both insert and overtype modes. To delete a line,position the cursor anywhere on that line, and enter\K (Kill line). The lines below the deletedline will move up to fill the space created by the deletion. The cursor will be positioned on theline which was previously immediately below the deleted line.

Similar to the window commands introduced in Section 2.4, the commands for inserting anddeleting text also have equivalent control code commands, as shown in the table below. It isoften easier and quicker to enter the commands in this fashion, particularly when inserting ordeleting multiple lines or characters.

Command	Meaning	Control Code
C	insert Character	Control/^ *
D	Delete character	Control/D
L	insert Line	Control/L
K	Kill line	Control/K

* Note that this does not follow the pattern of the other commands. One would expect this to be Control/C. However, Control/C is used for another purpose, which will be described inSection 2.7. If you delete a line by accident, it is possible to recover it (from a frame called OOPS) see Section 4.

2.6 An Introduction to Leading Parameters

So far, when you have wanted to do something which required a Ludwig command, you havetyped the command introducer (which was probably \) followed by the name of the command,for example C to insert a character. If you wished to repeat the same command several timesover, you have repeated this process the required number of times. This was made somewhat easier when the command had an equivalent control code, for at least then you only had to hold down the Control key and press one other key the required number of times.

If you have had some experience with using Ludwig, you will realize that these sorts of tasks could be done quickly and more accurately if it were possible to specify exactly how many times a command should be done at the time the command was entered. Where meaningful Ludwig commands take a repeat count. When you are entering the command, the repeat count is entered between the command introducer and the name of the command. The Jump command(J) moves the cursor one character, so to jump 5 characters to the right, you would enter \5J (remembering that none of this will actually appear on the screen). As you have already seen, when the count is 1, it may be left out, so \J is really the same as \1J. Something which appears between the command introducer (\) and the name of the command is known as a "leading parameter".

Now, what if you want a command to be executed as often as possible, but you are not sure exactly how many times this is? For example, you may want to jump as many character positions to the right as possible, until you reach the blank space at the end of the line. You don't know how many characters there are between your current position and the end of theline, and you certainly don't want to waste time by counting them first. Moreover, if you did this, it is always possible that you may miscount them. One of the prime functions of a text editor is to save you from having to do such mechanical, error-prone tasks. To cover cases such as this, many commands also take an indefinitely large repeat count. This is specified bythe greater-than sign ">". So to jump to the end of the line, you would enter \>J.

The "-" prefix, when allowed with a particular command, means that the command applies inthe opposite sense (eg. direction) to that usually taken. For example, whereas J jumps one character position to the right, -J jumps one character position to the left and -5J will jump 5 characters positions to the left.

Analogous to ">", there is also "<". This is used to specify an indefinitely large repeat count in the opposite sense. So, <J will advance as far as possible in the reverse direction, that is, to thebeginning of the line. Unlike the end of a line, which varies according to the length of the line,the beginning of a line is always considered to be the first character position in the line, even if the non-blank text of the line begins further to the right.

These leading parameters have a similar effect when used with the Advance (A) command which advances the cursor to the beginning of the next line. So 5A will advance down 5 lines, while -5A will advance up 5 lines. >A will advance down as far as possible, that is, to the endof the file. <A advances up as far as possible, leaving the cursor at the beginning of the file.<A is similar to WT and >A to WE (see Section 2.4), except that the cursor is always left at the beginning of the line when the A commands are used. With WT and WE, the position ofthe cursor along the line is not changed.

If you have entered \10A and there are less than 10 lines to the end of the file, the terminal bell will ring and the cursor will not move. The same is true if you enter \-10A when there are less than 10 lines to the beginning of the tile.

You can find out which of these leading parameters is allowable with a particular command by looking at the Help entry for that command. If you enter \H, and then respond to the "Topic:" prompt with A or a, followed by Return, you will find a brief explanation of the Advance command A, and some examples of its use with different leading parameters. Near the bottom ofthe screen, you will see a line like this:

LEADING PARAMETER; [none, + , - , +n , -n , > , < , @] A

Symbols for the leading parameters are given inside the square brackets.

"none" indicates that the command can be used without any leading parameter at all, as youhave been doing up to now. "+" and "-" indicate that leading parameters of "+" and “-“ may be used. "+n" and "-n" indicate that positive or negative integers, respectively, may be used asleading parameters. When the number is positive, the "+" is usually omitted, though it may beincluded if you wish. ">" and "<" indicate respectively that a positive or negative indefiniterepeat count may be used. The meaning of the leading parameter "@" will be given in Section2.12.2 .

It happens that all the possible leading parameters may be used with the Advance command.However, this is not true of most commands. If a leading parameter cannot be used with aparticular command, its place in the list is left blank.

2.7 Aborting Commands

Some actions can be aborted. This is accomplished by pressing Control/C. If it is possible, the text will be restored to the state it was in before the undesired action was performed. Theterminal bell is rung and the window of text on the screen is redrawn.

Up until now, all the commands which tell Ludwig to do something to your text have been one-or two-letter mnemonics which are suggestive of the command name. You have learned thatsome of these can be entered in an alternative manner --- by holding down the Control keywhile pressing one other key. However, here we have a command which can only be enteredin the latter way. There is no mnemonic corresponding to Control/C.

Any Ludwig command that is executing may be aborted at any time by pressing Control/(C. Forexample the Replace command may be aborted while it is prompting for an answer.

If you have been typing characters on one line, without moving the cursor or entering acommand, Control/C will erase all the characters you have typed since the last command orcursor movement. This also applies if you have been deleting characters using the Delete key.Similarly, if you have been moving around the text with any combination of the arrow, Return,Tab, or Backtab keys, without typing any characters or entering any commands, Control/C willput the cursor back to the position it was in before you pressed any of the cursor positioningkeys.

The range of Ludwig commands which can be undone by pressing Control/C is quite limited. However, it never hurts to try pressing Control/C before you resort to restoring the text manually.

2.8 A Word About Prompts

Some of the Ludwig commands introduced from now on will involve you making a choice or supplying further information. You have already seen one of these: the Help command. When you are required to supply information from the keyboard, a message will appear on the screen.This is called a prompt. It will stay there until you type your response, in most cases, followedby the Return key.

If a prompt appears because you accidentally entered the wrong command, or you decide thatyou do not want to do that command now, you can get rid of the prompt without executing the command by pressing Control/C, as explained in Section 2.7.

You can think of the prompt as overlaying your text on the screen. It will not alter the text in any way, or become part of it. A prompt will usually appear on the top line of the screen, although if you are at the beginning of your file, it will appear on the bottom line of the screen.

If you make a mistake while you are typing your response, you can use the Delete key to goback and correct it.

In all cases, the characters you type in response to the prompt must be printable. As Return isalways used to indicate that you have finished the response, the response can not extend overmore than one line.

2.9 Messages Which Appear on the Screen

You will almost certainly have done something by now to have caused the terminal bell to ring. This may have been because you tried to move the cursor beyond the beginning or end of the file, or you may have pressed a key on the keyboard which Ludwig does not recognize, or you may have mis-typed a command.

While it may be annoying, the ringing of the bell is no cause for alarm. It is simply the editor's way of letting you know that you cannot do what you just attempted to do, wittingly or unwittingly. Particularly if you are watching the keyboard as you type, the sound of the bellwill draw your attention to the screen. If this did not happen, you would probably continue what you were doing, unaware that perhaps your commands were no longer having the effect you had intended. As you become more experienced in using the editor, you will learn to anticipate this behaviour, and put it to good use.

There are two possible consequences of mistyping a command. The command which you accidentally typed may itself be a valid Ludwig command, in which case you will probably be startled by the resulting behaviour, especially if you are not familiar with that particular command. The best thing to do under these circumstances is to try to undo the command withControl/C. If the command which you accidentally typed is not a valid Ludwig command, then the terminal bell will ring, and the message

Syntax error.

will appear on the screen. This is to let you know that your error was in the syntax of the command. In other words, you did not obey the rules for contructing a valid Ludwig command. You will get the same message if you use a numerical leading parameter with a command with which it makes no sense, for example, 5Q. The name of a command, and the types of leading parameter which are allowed with it, are examples of rules of syntax.

This error message is one example of a range of messages which may appear on the screen atdifferent times. The appearance of such a message is not always due to an error on your part. For example, messages also appear when Ludwig is invoked, and when it is exited. These are informational messages, to let you know what is going on, particularly when you have to wait for a little while.

As with the prompts described in the preceding section, you can think of a message as overlaying your text on the screen. It will not alter your file in any way, or become part of it. A message will always appear at the bottom of the screen. Usually, the message will disappear by itself when you type the next character of text or start to enter the next command. However, sometimes the message

Pausing until RETURN pressed:

will also appear at the top of the screen. Only when you press Return will both messages disappear and the screen be restored to its previous state.

2.10 Finding Text

This section will deal with the Ludwig command for Ending occurrences of strings. The word "string" is used here to mean any sequence of printable characters, as defined in Section 1.1.

To find the next occurrence of the required string from the current cursor position to the end of the file, enter \G (Get). The prompt

Get:

Will appear. You then type in the string which you want to find (known as the "target string" using the Delete key to correct any mistakes, and pressing Return when completed. Ludwig will then begin searching for the target string. When the nearest occurrence is found, the cursor will be positioned on the character immediately following the end of the string. The prompt

This one?

will appear somewhere on the screen. You must respond to this prompt with one of several possible answers. You may type Y or y (for "yes") if this is the occurrence you wanted to find, or N or n (for "no") if it is not.

If you reply with N or n, the search will continue for the next occurrence and prompt in the same manner. If there are no more occurrences, the terminal bell will ring and the cursor will be left in the position that it was in when the Get command was entered. This is also the behaviour if you attempt to search for a string which does not occur in the file at all. This is extremely useful if you wish to be sure that there are no occurrences of a particular string in a file. For example, if you frequently misspell a certain word, you can use the Get command to check that there are no occurrences of the erroneous spelling. In Section 2.11, you will see how you can automatically correct each mis-spelled occurrence of that word.

If the target string does not occur in the current screenful of text, then when it is found, only the line containing it, and the two lines immediately above and below it, are displayed. If this is too little information to allow you to decide whether or not it is the occurrence you are seeking, typing M or m (for "more") in response to the "This one?" prompt will cause one more line both above and below the existing five lines to be displayed. The "This one?" prompt is then redisplayed. You may continue to ask for further lines of context to be displayed, if necessary, or respond in the usual manner if you are now able to make a decision regarding this occurrence of the string. If you respond by typing Y or y when there are less than the full number of lines on the screen, the remaining lines will be filled in.

If you do not give one of the correct responses to the "This one?" prompt, the terminal bell will ring and a line of text will appear to let you mow what the possible responses are.

Reply Y(es),N(o),A(1ways),Q(uit),M(ore)

Of these, the responses A and Q have not yet been explained. Typing Q or q causes the search to end. The terminal bell is rung and the cursor is left in the position that it was in when the Get command was entered, just as if the target string had not been found at all. In this example, typing A or a will have the same effect as typing Y or y, but its true purpose will be made apparent in Section 2.11.

Pressing Return in response to the prompt has the same effect as typing N or n, while pressing Space has the same effect as typing Y or y. Note that, unlike the string you type in response to the "Get:" prompt, your response to the "This one?" prompt is not echoed on the screen.

Sometimes you may be surprised when Ludwig finds an occurrence of the target string in a place you did not expect, such as inside another string. For example, searching for the string "the" will find occurrences of the word "the", but it will also find the string "the" in the strings "then", "there", "another", and other words containing the characters "t", "h" and "e" in that order. There are ways to prevent this, which are discussed in Section 7.4. At the moment, it is enough simply to be aware of this behaviour.

If you entered the target string in the manner described, the case of letters will be ignored, and so the string the" will also match "The" "THE" or any other combination of the upper and lower case letters "t", "h" and "e", in that order. This is known as "inexact case matching". In order to specify to Ludwig that you wish it to also consider the case of letters when searching for the target string (using what is known as "exact case matching") you must begin and end the target string with the double quotation mark (").

For example, to find an occurrence of the word "The", with exact case matching such as may occur at the beginning of a sentence, you would enter the target string as shown below,

Get:"The"

rather than

Get:The

However, giving a special meaning to a character such as ", which may itself occur in a string causes some problems. What would you enter if you wish to find a string which was itself surrounded by double quotation marks? The solution is to surround the whole target string, including the double quotation marks, by single quotation marks ('). This pair of single quotation marks has the effect of preventing the interpretation of any "special characters" within them. So, for example, to find the strlng "the", including the double quotes, you would enter the target string as shown below.

Get:' "the" '

A surrounding pair of single quotation marks will prevent the interpretation of any special characters, including the single quotation mark itself. So, to find the string 'the', you would enter the target string thus:

Get:' 'the' '

You will recall that the usual effect of surrounding a target string by a pair of double quotation marks is to specify that the case of letters must be considered when searching for matching strings. Like the single quotation marks, they also prevent the further interpretation of any other special characters which may appear between them. So, to find only the string 'The', and not 'the', including the single quotes, or 'THE', you would enter the target string as shown below.

Get:" 'The' "

Similarly, to find the string "The", including the double quotes, you would enter the target string as shown.

Get:" "The" "

So, both single and double quotation marks prevent the further interpretation of special characters, but they differ in that double quotation marks cause exact case matching to be used, whereas single quotation marks cause inexact case matching to be used.

Single and double quotation marks surrounding a string are examples of "delimiters", There are also other delimiters which may surround strings given to the Get Command, all of which may themselves legally appear in the target string. They are the dollar sign ($), the ampersand (&), the question mark (?) and the backwards quotation mark (`). (Their interpretations when used as special delimiters of target strings will be given in Section 7.4.) If you wish to find a string surrounded by a pair of these delimiters, you must surround the target string by either single or double quotation marks, depending on whether or not the case of letters in the string is important.

The Get command has a feature which will save time and reduce the amount you must type. Ludwig stores the target string which you last gave to the Get command, so if you need to find the same string again, you need only press Rank in response to the Get: prompt.

The Get command searches the text for occurrences of the target string from the current cursor position to the end of the file. In other words, only occurrences at or to the right of the cursor in the current line, or anywhere on subsequent lines, are found. So any occurrences of the string which are *before* the current cursor position (to the left of it in .ac line, or on previous lines) will not be found. To specify that the search should be done from the current cursor position *backwards to* the top of the tile, use a leading parameter of "-", that is, enter \-G. When an occurrence is found, the cursor will be positioned on the first character, rather than the last, of the string.

It is also possible to give the Get command a positive or negative numerical leading parameter. fiG searches for the nth occurrence of the target string, from the current position to the end of the file. Of course, it will fail and ring the terminal bell there are less than n occurrences of the target string. If there are more than n occurrences, and you reply to the "This one" prompt with N or n, it will go on to search for more occurrences as usual. -fiG searches for the nth occurrence of the target string in the opposite direction.

2.11 Substituting Text

Now that you know how to find a particular string, the idea of replacing that string with another follows quite naturally. This facility is useful for such tasks as replacing all misspelled occurrences of a word by the correct spelling.

The command for replacing one string by another is really just an extension of the Get command, and most of the techniques you learned with the Get command can be applied. To replace the next occurrence of the required string by another string, from the current cursor position to the end of the file, enter \R (Replace). The prompt

Replace:

will appear. You then type the target string, as for the Get command. When you press Return on completion of the target string, another promp

By :

will appear. You then type the string with which you wish the target string to be replaced, known as the replacement string, pressing Return when completed. Ludwig will then begin searching for the target string. When the first occurrence is found, the cursor will be positioned on the character immediately following the end of the string. The prompt

Replace this one?

will appear. The possible responses, and their resulting actions, are the same as those for the "This one?" prompt of the Get command.

The rules regarding the use of the single and double quotation marks (' and ") to surround the target string of the Get command also apply to the target string of the Replace command. However, the case of the replacement string is always taken just as you give it, so there is no need to surround it by quotation marks unless the string itself happens to have these or one of the other special delimiters (' $ & ?) at each end.

When the replacement occurs, the string being replaced is deleted and the replacement string inserted. The replacement string can be longer or shorter than the string it replaces.

To specify that the searching and replacement should be done from the current position towards the top of the file, a leading parameter of "-" may be used with the Replace command, as with the Get command. A positive or negative numerical leading parameter may also be used. However, the Replace command differs from the Get command in that nR means replace the next n occurrences in the appropriate direction, not just the nth occurrence. The "Replace this one?" prompt will continue to appear until you have replaced n occurrences of the target string. If you respond with Q or q to quit the search, or run out of occurrences to replace before then, the terminal bell will ring and the cursor will return to the position of the last successful replacement.

The "quit" response to the "Replace this one?" prompt is more useful when multiple occurrences are being acted upon in this way, than with the Get command, when only one occurrence was being acted upon. You can use it if you decide that you want fewer than the specified number of occurrences to be replaced, or it can be used at any time during the search if you change your mind altogether. If no replacement was done, the cursor is left where is was when the command was entered.

Responding with A or a is also very useful when you are dealing with multiple replacements. If you are sure that you wish all the remaining replacements to occur, either at the beginning of the search, or at any stage during the search, the "always" response will cause the search and replacement to proceed without prompting for verification of each occurrence before it is replaced. When a total of n occurrences have been replaced, the cursor will be positioned on the character after the last character of the last replacement if +nR is being used and on the first character of the last replacement if -nR is being used.

Unlike the Get command, the Replace command takes both the indefinite leading parameters,">" and "<". >R and <R replace all occurrences of the target string downwards and upwards respectively. The "Always" response to the "Replace this one?" prompt may be used at any stage to specify that the replacement should proceed without further verification. The "quit" response terminates the search and replacement, and returns the cursor to the last successful replacement, as explained earlier. Unlike the replacement of a specific number of occurrences,

>R and <R can never fail, as the repeat count is indefinite. The terminal bell will never ring, even if there are no occurrences of the target string in the file at all.

Ludwig stores both the target string and the replacement string which you last entered. You need only press Return in response to the "Replace:" prompt in order to replace the same string again. The "By:" prompt will not appear at all in this case. If you are part way through typing either string and decide that you do not want to do the replacement, pressing (Control/C will cancel the command and redraw the screen.

2.12 Marking Text

It is useful to be able to explicitly mark a place in the text, for later reference. This can serve as a bookmark, saving the trouble of scanning the text to find a particular place again. It can also be used to group text together, as will be explained in Section 2.12.1.

The Ludwig command for marking text is Mark **(M).** Entering this command leaves a mark at the current position of the cursor, no matter where this is in the file. It may be on a character in the text, somewhere out in the blank space to the right of the text, or even on the <End of File> line. This mark is not visible on the screen, nor does it alter the text in any way.

The mark is attached to a character (which may be a blank), and moves with it. If the whole line on which the mark is situated is deleted, the mark moves to the first character position of the next line.

A positive numerical leading parameter can be used to specify the number of the mark. You are allowed to have up to 9 marks, numbered 1 to 9. So nM marks a position with mark n. If another place is marked with the same mark, the old position is forgotten. Of course, 1M is the same as M. To explicitly remove mark **n,** execute the command **-nM.** The cursor does not have to be in any particular place to do this.

2.12.1 Spans and How to Define Them

In Ludwig, characters and lines of text may be grouped together into a unit known as a "span". A span has a beginning and an end, and a name, by which it can be referenced. In order to make a piece of text into a span, you must supply these three things:

Move the cursor to the first character that you would like to be included in the span, and mark it, using \M. Then move the cursor to the character *following* the last character you wish to be included, and enter the Span Define command (SD). The prompt

Span :

will appear. Type the name by which you wish the span to be known. A span name may contain blanks. Only the first 31 characters given are used, but it is best if the name is fairly short in order to minimize the amount of typing you have to do each time you refer to the span later. When you have typed the name, press Return. You have now defined a span.

You do not have to use just M. You can use any of the nine possible marks, but you must then remember to use the same number as a leading parameter to the SD command. For example, if you entered \3M to mark the first character, you would then enter \3SD to define the span.

Since it is impossible to tell where a span is just by looking at the text, there is a command to show an index of all the spans currently defined. The Span Index command **(SI)** will cause the text on the screen to be replaced by a list of span names. Next to each name, the characters which follow the beginning of that span are shown. Only one line is given, even if the span extends over more than one line. If the span is longer than 31 characters, only the first 31 are shown. If the span contains less than 31 characters, the entire span is shown. Following the list of spans, you will also see another list, headed Frames. This is not relevant yet, and will be explained in Chapter 3. Pressing Return, as indicated, will return to your text.

If you edit the text inside the boundaries of a span, the contents of the span are altered accordingly. Once a span has been defined, it is not dependent on the mark, nor on its boundary characters. If either of the boundary characters are deleted, the span simply contracts.

If a new span is defined using the name of an existing span, the old span becomes undefined. To explicitly undefine a particular span, enter \-SD and reply to the "Span:" prompt with the name of the span to be undefined.

2.12.2 Referring to Marked Text

At the end of Section 2.6.1 (on leading parameters), there was a brief note to the effect that "@" could also be used as a leading parameter with some commands. Using "@n", where n can be any integer from 1 to 9, as a leading parameter signifies that the command applies to mark n. The number may be omitted if referring to mark number 1.

You have already seen nSD used to define a span between the current position of the cursor and mark n. The "@" may be omitted in this case.

Other commands which you have already seen which take "@n" as a leading parameter are A, D, J and K. You can see that the "@" must be present with these commands, as numerical leading parameters are normally used to indicate that these commands should be executed a certain number of times.

Here is the action of each of these commands when a leading parameter of "@n" is given :

Command	Action
A	Advance to the first character position of the line containing mark n. This may be in either direction.
D	Delete all characters from the current cursor position to mark n. This may extend over more than one line, in either direction.
J	Jump to the character on which mark n is located. This does not have to be on the current line.
K	Kill all lines from the current line to the line containing mark n. This may be in either direction.

The symbol "=" can be used as a leading parameter whenever "@n" can be, and specifies that the previous position of the cursor, rather than mark n, should be used as a reference point for the command.

2.12.3 The Difference Between the Cursor and Dot

The cursor is a useful visual reference, however Ludwig uses an invisible mark, which is always positioned between two characters, and to the left of the character occupying the column which Dot is in. In this way Ludwig commands such as Get (see Section 2. 10) will always find a character to the right of Dot (the character which the cursor is on) if that character is the one being searched for. It also allows sections of text, such as paragraphs, to be defined in terms of inter-character positions, rather than in terms of beginning and end characters. Most Ludwig commands can be explained in terms of either Dot, or the current character. Some Ludwig commands can only be described in terms of Dot and therefore most discussion of commands will be in terms of Dot.

2.13 Moving Text

It is often the case that you wish to move a piece of text from one place to another. This piece of text may be a word, part of a line, a whole line or several lines. Deleting the piece of text and retyping it elsewhere is time-consuming, tedious and prone to error.

2.13.1 Transferring a Span of Text

The piece of text which you wish to move should first be defined as a span, using the method described in Section 2.12.1. Once this has been done, the whole piece of text may be picked up and moved as a single entity. To transfer a span from one place to another, first move the cursor to the position to which you wish the span to be moved. Enter \ST (Span Transfer). The prompt

Span :

will appear. Enter the name of the Span and press Return. The Span will be transferred from its original position to the new position, starting at the current position of the cursor. The text at the current position always moves aside to make room for the Span.

If one Span is transferred inside another, it becomes part of the other Span. If you try to transfer a Span to a position located inside the Span itself, the Span will not move.

When you define the span, it is sometimes better to have the cursor on the beginning of the line following the last line you wish to be included in the Span, so as to include the end of line in the Span.

2.13.2 Transferring a Single Line of Text

Transferring a single line of text may be thought of in terms of swapping that line with the line above or below it. With the cursor positioned anywhere along the line to be moved, entering \SW (SWap line) will swap this line with the line below it. Entering \-SW will swap the current line with the line above it.

Positive and negative numeric leading parameters may be used with the SW command to increase the distance that the current line may be moved. You can think of +nSW as swapping the current line with the line below it n times. The result of this is to move the current line n lines down. The lines between the old and new positions of the current line move up one line, but their relative order remains the same. Similarly, -nSW swaps the current line with the line above it n times, and so moves the current line n lines up.

SW is another command which may take "@n" as a leading parameter (see Section 2.12.2). Entering \@nSW moves the current line to a position above the line containing mark n.

2.14 Copying Text

Particularly when writing programs, you may find yourself typing the same, or very nearly the same, piece of text over and over. This is time-consuming, tedious and prone to error.

2.14.1 Copying a Span of Text

The procedure for copying a span of text is almost identical to that for transferring it. First define the piece of text as a span, as described in Section 2.12.1. Then move the cursor to the position from which you wish the copy to start. Enter \SC (Span Copy). The "Span:" prompt will appear, to which you respond with the name of the span, followed by Return.

The text will always move aside to make room for the copy. The original span remains where it is. Once the span has been defined, it may be copied to as many places in the text as you require. Note that if you edit the original span, any copies made thereafter will include the alterations, but the previously made copies will remain unchanged.

A copy of a span may be placed inside the span itself. As is the case with any other alteration made to the original span, the copy is then also part of the span.

The SC command may take a positive numeric leading parameter. Entering \\nSC will produce n copies of the span one after the other, starting from the current cursor position.

2.14.2 Copying Single Characters of Text

To copy a single character from the line above, move the cursor to the position directly underneath the character to be copied, and enter \". To copy a single character from the line below, move the cursor to the position directly above the character to be copied, and enter \'. These are called Ditto commands.

A leading parameter of "+n" may be used with either of these commands to copy n characters to the right, including the current one. Similarly, a leading parameter of "-n" may be used to copy n characters to the left, not including the current one. A leading parameter of ">" will copy all the characters from the current one to the end of the line. A leading parameter of "<" will copy all the characters from the character to the left of the current position, to the beginning of the line.

However, once you have entered \" or \', the command remains in effect. You can then go on entering just " or ' to continue to copy as many characters from either line as necessary. This remains in effect until you press a different key.

2.15 Changing Case

Sometimes you may want to change one or more characters from lower to upper case, or vice versa. For example, you may be writing a program in a language which requires that all reserved words be in upper case letters, and have forgotten this as you were entering the text.

There is a Ludwig command to change the case of alphabetic characters. Firstly, enter * Now you must specify whether you want to change the character to lower or upper case. If you are changing to upper case, enter a U and if lower case, an L. Only when you have entered this, is the command complete. If the character on which the cursor is positioned is a letter ih the opposite case from that specified in the command, it will be converted appropriately and the cursor will then be positioned one character to the right. If it is already in the case specified, the command will have no effect except to move the cursor. The latter is also the case if the character is not alphabetic.

The * command accepts the same leading parameters as the " and ' commands introduced in Section 2.14.2, and their behaviour is analogous. This also means that once you have entered *, you need only enter U or L in order to change the case of successive letters, and this remains in effect until you press a different key.

In addition to U and L, there is a third type of case change you may specify --- E. This stands for "edit case". In edit case, the first letter of every word is in upper case, and the remaining letters are in lower case. Specifying this case change is most useful in conjunction wit* __ numerical or indefinite leading parameter. For example, with the cursor positioned on the following line as shown,

the university of adelaide

entering \>*E produces

The University Of Adelaide

2.16 How to Find Out More

Whenever you feel locked into a situation where you must do lots of mechanical operations, particularly those which require visual judgment, first see if a Ludwig command exists which will accomplish the same task.

One way to do this is to use the on-line Help facility which was introduced in Section 1.3. Entering \H and then responding to the "Topic:" prompt with the number 3, followed by Return, will give you a complete list of Ludwig commands with a one-line description of each. Once you have found a command which you think may do the job, you can look up the full entry on that command by responding to the "Command or Section or <retum> to exit:" prompt with the command name.

3. Environment Settings

Ludwig allows the capability to modify the editing environment to suit individual tastes. For example some people prefer text to be inserted into the current document when it is typed while others prefer typed text to overwrite existing text. Other environment features are discussed below.

The Edit Parameters command (EP) allows default settings to be changed. Edit Parameters, or environmental settings, are specific to each Frame (see Section 4), with the exception of the C and K parameters, explained below, which always apply **to** every Frame. When a Frame is created it will have the default settings. Default settings are global, but may be modified and new defaults imposed for new Frames.

The EP command will prompt for additional information with "Param : See examples below.

3.1 Keyboard Modes

There are two modes of text entry via the keyboard: Keyboard Insert and Keyboard Overtype. Keyboard Insert will enter typed text into the Frame and move existing text to the right as characters are inserted. This is the default mode and is the common method of entering text in word processors. Keyboard Overtype will overtype, or overwrite, any existing text as new text is entered.

The command EP with parameter K=O will set the mode to overtype, while
EP'K=I' will set the mode to insert.

Because it is convenient to change between the two modes frequently when editing These two EP parameters have been mapped to the keyboard commands O and I respectively. Thus EP'K=O' is equivalent to \\O and E'K=I' is equivalent to \I.

The command EP'K=C' will enable direct command entry,
EP'K=-C' returns to text insert or overtype.

This feature is sometimes useful for testing command procedures.

3.1.2 Changing the Command Introducer

It is sometimes convenient to be able to change the command introducer \\ to another character. This can be done with the C option within the EP command. For example to change the command introducer from the default, \, to *, use

EP'C=*'

The command introducer may also be set to be a function key with the command

EP'C=keyname'

where the key fame is taken from the terminal description file (see Section 9).

3.1.3 Changing the Size of the Window

Ludwig allows the user to change the size of the window (the amount of text that will be displayed). By default Ludwig uses a window size of the terminal in use, usually 80 characters wide by 24 lines - the standard size for a visual display terminal. When using low speed modems, for example, it might be desirable to limit the size of the display to increase the effective throughput, especially when whole screens are being transmitted. Both horizontal and vertical limits can be set to modify the size of the window.

EP'H=height'	Sets the number of lines to be displayed. Default is the height of the user's screen, usually 24.
EP'W=width'	Sets the number of characters displayed. Default is the width of the user's screen, usually 80.
Example: \EP'h=4'	would set the window size to 4 lines without modifying the width.

3.1.4 Changing Horizontal and Vertical Margins

The horizontal and vertical margins determine the scrolling region for the screen as well as the position for wrapping text (if wrap is turned on - see Section 3. 1.6).

The top vertical margin (default is normally 4, but depends on the size of the terminal screen) determines how close to the top of the window the cursor may move to before more lines are scrolled down. With the default value of 4, the cursor will not move higher than 4 lines from the top of the screen (except at the top of the file). This feature allows the surrounding context to be always on die screen when editing a particular line. Forcing the vertical margin to zero would mean that the cursor could be positioned at the top of screen and editing take place, without the previous line(s) being visible. The bottom vertical margin performs a similar function.

The left horizontal margin determines where the cursor will return to on pressing Return or where words will be positioned, if word wrap is on, see Section 3.1.6). The right horizontal margin determines the point at which text may no longer be entered. If such an attempt is made the word will wrap to the next line (if turned on), or the bell will sound and entered text will be discarded. Text may be inserted past the margins, but the cursor must be moved past the margin (left or right) explicitly first

Note that arrow keys, **Tab** and **Backtab**, and **Backspace** will move past the horizontal margins, but **Return** will not. The vertical margins can only be passed at the top of file or end of file (or page),

The default settings for margins are -

left margin	1
right margin	width of screen (usually 80)
top margin	4
bottom margin	4

These may be changed as with the M and V options (for horizontal and vertical margin settings) as follows -

EP'M=(10,70)'	To set the left margin to 10 and the right margin to 70.
EP'V=(1,4)'	To set top margin to 1 and bottom margin to 4.
EP'M=(10)'	To set left margin to 10 and leave right margin unchanged.
EP'M=(,50)'	To set right margin to 50 and leave left margin unchanged.
EP'M=(.)'	To set left margin to the current cursor (Dot) position.

The left brace (**{**) may also be used to set the position of the left margin. This is extremely useful during editing sessions where paragraph indenting is required. The command { will change the left margin to be the current position of the cursor. This is equivalent to the command EP'm=(.)'. The right brace (}) may be used to set the right margin at the current cursor position.

3.1.5 Changing Tab Settings

The default tab settings are 9,17,25,33, ..., with left margin at 1 and right margin at 80. Tab settings may be changed in several ways:

EP'T=(c1,c2,...cn)'	will set tab stops at the column position indicated. example: EP'T=(5,l0,30)'
EP'T=D'	will reset the tabs stops at the default values.
EP'T=S'	will set a tab stop at the current position of Dot.
EP'T=C'	will clear a tab stop at the current position of Dot.
EP'T=I'	will insert a line into the current Frame showing the tab stops.
EP'T=T'	will set tab stops using the current line as a Template.

The tab options I and T may be used together to modify existing tab stops. Thus a line would be inserted into the current text showing existing tab stops. This line can be modified and used as a template and then deleted from the text.

3.1.6 Changing Default Values

Any Frame that is created will take the current global default environment parameters. The defaults may be modified by including a $ before the new parameter settings. For example, to change the default window height the command

EP'$H=10'

would be used to set the default height to 10 lines. This setting would then apply to all new Frames which are created.

3.1.7 Word Wrap, Indentation Tracking and Newline Insert.

The Editor Options sub-command allows -

EP'O=W'	Enables word wrap onto the next line when typing. O-W will tumthe option off. This feature is appropriate for document preparation.
EP'O=I'	set the indentation tracker on. This feature allows for the indentation of paragraphs by using the the current paragraph indentation as left margin, and is commonly used for program indentation.

EP'O=N' Forces a newline when lRet1m1m is pressed when keyboard insert is used. This may be undone by Delete at column 1, or at left margin if there is no text between column 1 and left margin

EP'O=S' Shows the current Options selected.

3.1.8 Changing Memory Limits

The default memory limit for each Frame is 500,000 characters. This may be changed with the S option, for example

EP'S=1000000'

will set the memory limit to 1,000,000 characters.

3.2 Splitting and Joining Lines

There is a Ludwig command to split a line into two lines: the Split Line **(SL)** command. Move the cursor to the character which you wish to be the first character of the second line, and enter \SL (Split Line). The character on which the cursor was positioned will move to the first character position of the next line, dragging all those characters which were to the right of it with it, and the cursor will still be positioned on it. If there was already text below the original line, this will move down to make room for the new line.

There are several ways to do the reverse, that is to join two adjacent lines together into one line, depending on which Edit Parameters have been set (see Section 3).
With the Edit Parameters set so that pressing Return gives a new line, then pressing Delete re-join the lines (see Section 3).

With default settings move the cursor to the character position following the last character of the first line of the pair, and then press Return. Now, enter \=D. The second line will move up, and its first character, on which the cursor remains, will be immediately to the right of the last character of the first line.

There is one point you must remember. In order for this to work as just described, you must not enter any other command, or even press a key on the terminal which will cause the cursor to move, between pressing Return and entering \=D. The "=" is a leading parameter, like those you saw in Section 2.6. 1. Rather than indicating that the command should be done a specific number of times, it means "execute the command from where the cursor is now to where it was before". So, =D in this situation deletes every character between the current position of the cursor (on the first character of the second line) and the previous position of the cursor (following the last character of the first line). Thus, all the blank space at the end of the first line is deleted, and so the two lines become one. But, **if** you **move** the cursor again before entering this command, the characters between this new cursor position and the first character of the second line are deleted, so be careful. A more complete explanation of "=", and further uses, is given in Section 2.12.2.

4. Frames and Multiple Environments

4.1 The Concept of a Frame

It is often useful to be able to edit parts of the same document separately, but within the same editing session. It is also useful, sometimes, to have more than one file open for editing at the same time (and within the same editing session) and to allow the transfer of text between these environments. Within Ludwig, Frames allow for such multiple editing environments.

For example, it would be useful, while typing in a new program to be able to insert a section of code that already exists in another program. This may be achieved by creating another Frame (while editing the program), opening and reading the old program file and copying the relevant code from that Frame to the appropriate position in the new program in its Frame.

Alternatively one may wish to edit a particular section of a file in a completely separate environment. Copying, or transferring the section, or Span, of the file to a different Frame allows for such a function. It is then possible to write the Frame (or part of it) to a file or to copy or transfer it to another Frame.

A Frame may also be regarded as a window onto a tile, multiple Flames allow windows onto multiple files and the ability to transfer text between them. Text transfer between Frames achieved (with the SC and ST commands) in the same manner as text transfer within a Frame.

A Frame is a separate editing environment, within a single editing session, which may have input and/or output files associated.

4.2 Creating Frames and Moving Between Frames

The Edit command (ED) will prompt for the name of the Frame which you wish to edit. Frame names may be up to 31 characters long consisting of any of the printable characters. Frame names are displayed next to the <End of File> marker. The ED command will change the frame currently being edited to the Frame specified. If no Frame of that name exists one will be created.

When Ludwig is invoked the default Frame, called LUDWIG, is associated with the files specified at the operating system level. Frames that are created with the ED command have no default files associated them. It is possible to associate input files and output files with other Frames (see Section 5.1) and to edit these files and transfer data between Frames.

The Edit Return command (ER) will return to the last used Frame, while the command ED with no Frame name specified will return to die default Frame (LUDWIG).

4.3 Moving Text Between Frames

Conceptually a Frame is a superset of the concept of a Span. Thus an entire Frame may be transferred or copied into another Frame. In Sections 2.14 and 2.15 the methods for transferring data within a Frame were discussed. The method of transferring data between Frames is similar. The Span must first be defined, by marking and defining the Span, then the appropriate Frame chosen and the position within Frame identified (with the cursor). Then the data may be tranferred with the command SC (Span Copy) or ST (Span Transfer). Because Frames are supersets of Spans it is possible to copy or transfer an entire Frame using the Frame name as a Span name.

4.4 Deleting Unwanted Frames

The Edit Kill command (EK) will remove a Frame from the current editing environment. Although there is no limit to the number of Frames allowed this command may be useful for tidying up unwanted Frames.

4.5 Listing Frame and Span Names

The Span Index command (SI) will display a list, or index, of all current Spans and Frames and files associated with each Span or Frame.

4.6 Pre-defined Frame Names

There are four Frames that Ludwig uses for special purposes.

When Ludwig is started up the Frame used to start editing the file specified on the operating system command line is called LUDWIG. This Frame will have input and output files associated with it, depending on what options were used at the operating system level. For example, if the Read Only switch was used there will be an input tile but no output tile.

The Frame COMMAND (see Chapter 5) is used by Ludwig to store command strings for execution and should not therefore be used for editing files because any data in the Frame may be deleted.

The Frame OOPS is reserved by Ludwig to retain deleted lines of text. This Frame is useful as a means of recovering deleted text and therefore should not be used as a user Frame. Text deleted from OOPS is irretrievably lost.

The Frame HEAP is reserved by Ludwig for use by the SA and UK commands and should also not be used as a user Frame.

5. Using Files

5.1 Files Attached to Frames

Associating files with frames gives the ability to read new text in, to write text out in a separate editing environment, which may then be transferred to other Frames.

The File Input command (FI) will open an input file and associate it with the current Frame. It will also automatically load the file into the Frame. The command -FI will close an associated input tile.

The File Output command (FO) will open a file, for output, and associate it with the current frame. The command -FO will close an output file, writing out the contents of the Frame first.

The File Edit command (FE) will associate an input and an output file, both of the same name, with the Frame in a similar manner to the default frame.

The File Back command (FB) will rewind the input file attached to the current Frame.

5.2 Globally Accessible Files

In addition to files which are specifically attached to particular Frames there is the facility to have an input and an output file which are accessible from any Frame. These are called Global Files and operations on them are similar to files associated with Frames. The relevant commands are

FGI Open a Global Input File (-FGI to close)
FGO Open a Global Output File (-FGO to close)
FGB Rewinds the Global Input File
n FGR Reads n lines from the Global Input File.
n FGW Writes n lines to the Global Output File.

```
Example 1.
ED'new_frame'          ! make a new frame
FI'file1'              ! open the file "file" and read it
- FI                   ! close the input file
WE                     ! Put DOT at the end of the frame
FGI'file2'             ! open file "file2" as a Global file
9FGR                   ! read 9 lines from file2 and append
                       ! to end of frame "new_frame"
                       !
                       ! other editing
                       !
WT                     ! put DOT at the top
FGO'file3'             ! open file3 for output as a Global
> FGW                  ! write the contents of the frame to
                       ! global file "file3"
Example 2.
ED'eric'               ! make a new frame called eric
FE'new'                ! open input and output files "new"
                       !
                       !
```

Note that files attached to Frames are read or written in their entirety, while global files allow parts of files to be read and written. Global files are accessible from every Frame, while files attached to Frames are not.

5.3 Using Very Large Files

Ludwig allocates a default limit of memory (currently 500,000 characters) for each Frame so that very large files may not fit into a single Frame, but the memory limit may be increased with the EP command, or the command line qualifier (/SPACE under VMS and -s under Unix).

Ludwig allows for a part of the file to be paged in and edited when the whole tile is larger than the memory Space limit. The File Page command (FP) allows for the previous contents of the Frame to be written out and a new page read in. For such large files, page boundaries are shown at the end of the page with the symbol <Page Boundary> instead of <End of File>.

Under VMS the maximum memory space limit is 1,000,000 characters. Under Unix there is no limit imposed.

6. Elementary Text Processing

Ludwig provides the ability to perform a variety of text processing based on the concept of a word being delimited by a space, comma or period. This definition allows for such functions as justifying and filling. Such functions are usually performed on the document as a whole or on a paragraph basis, although the commands themselves are word oriented. Because of this, these commands are commonly used in command procedures to build word processing functions.

6.1 Word Oriented Commands

YA Advance Dot by one word
YC Centres words on the current line between the margins
YD Deletes the next word.
YF Places as many words as possible on the current line, getting extra words from the next line if possible (Fill Line).
YJ Expands the current line with spaces to fit exactly between the margins (Justily Line).
YL Forces a word to start at the Left margin.
YR Forces a word to end at the Right margin.
YS Removes extra spaces from the current line (Squeeze Line).

Examples:

```
>YF =J >YC          'This procedure will centre an entire paragraph.
>YS =J >YF =J >YJ   This procedure will fully left and right justify a paragraph.
```

The Editor Options commands (EP, O=W and I) for wrapping words at the right margin and setting the indentation tracker are also useful word processing options.

6.2 Tests

Tests for conditions relating to the current position of dot in relation to word, column and mark positions are available -

EOL Tests that dot is at End of Line, which is defined to be the position immediately to the right of the last non-space character on the line.
    >EOL tests that dot is at or beyond the End of Line.
    <EOL tests that dot is at or to the left of End of Line
EOP Tests if dot is on the null line at the End of Page, which is displayed as <Page Boundary>.
EOF Tests if dot is on the null line at the End of File, which is displayed as <End of Fi1e>.
EQC Tests for the column position of Dot. Column numbers may be in the range 1 to 400, which is the maximum length of a line.

Examples:

```
  EQC'9'        succeeds if dot is 'm column position 9
- EQC'9'        fails if dot is in column position 9
> EQC'9'        succeeds if dot is at or to the right of column 9
< EQC'9'        succeeds if dot is at or to the left of column 9
```

EQM Tests for the dot being on the position of the mark number specified.

Examples:

EQM'3'	succeeds if dot is at mark 3
- EQM'3'	fails if dot is at mark 3
> EQM'3'	succeeds if dot is at or to the right of mark 3
<EQM'3'	succeeds if dot is at or to the left of mark

EQS Tests for a string to the right of Dot, which will remain at its original position. By default inexact case matching is performed, but the rules of case matching apply, and any valid delimiter may be used (see 7.4)

Examples :

EQS'cat'	succeeds if Dot is at the start of the string eat
- EQS'cat'	will fail if the string car starts at Dot
> EQS'cat'	will succeed if the string starting at Dot, and extending for three characters, is lexically greater than cat
< EQS'cat'	will succeed if the string starting at Dot, and extending for three characters, is lexically less than cat

These conditionals are most commonly used within command procedures, which are discussed in the next Section.

6.3 Character Set Searches and Bridgeing

In addition to the Get command (G), which searches for a specified string of characters the Next command (N) will search for the next occurrence of any of a set of specified characters. The command will fail if there are no occurrences remaining in the Frame.

N'a'	search for the flrst occurrence of the character "a"
-3 N'a'	search backwards for the 3rd occurrence of "a"
N'abc'	search for the first occurrence of "a", "b" or "c"
N'a..z'	search for any lower case alphabetic character.

The Bridge command (B R) can be regarded as the inverse of the Next command in that it will skip over n occurrences of any of a set of specified characters. If the character at Dot is not in the set specified, Dot will not move and the command will not fail. Exact case matching is used.

BR'a'	skip occurrences of 'a' to the right
- BR'a'	skip occurrences of 'a' to the left.

As an example the Bridge and Next commands may be used to extract text and assign the selected text to a Span -

(N'0..9' M BR'0..9' SD'a' ED'temp' ST'a' ER)

This procedure will
- find the next numeric,
- define Mark 1,
- Bridge across to the next non-numeric character,
- Define a Span called 'a',
- Move to a Frame called 'temp', creating in if necessary,
- Transfer the Span 'a' into the Frame 'temp', and
- Return to the previous Frame.

The effect is to transfer the first occurrence of a string of numeric characters to the Frame 'temp'. The above procedure is functionally equivalent to

(N'0..9' BR'0..9' =SD'a' ED'temp' ST'a' ER)

but in this case numbered Marks are not used.

When Bridging, Dot is positioned (as always) to the left of the character

7. Ludwig Command Procedures

Ludwig provides the capability to compose commands together into command procedures, as a Span (or Frame), which are to be executed sequentially. Such command procedures, may be as simple or as complex as required. Command procedures may be stored and recalled or executed immediately.

7.1 The Execute String Command

In its simplest form this method for entering a sequence of commands is achieved through the A command, which allows interactive entry of commands, and in its most general form by the EX command which executes commands which exist in a Span or Frame. The command ^ (caret) will prompt (with Command:) for you to enter a command procedure. The string of commands that are entered are inserted into the Frame COMMAND and will replace any existing text in that Frame. On pressing Return the command procedure that has been entered will be compiled and executed

If any errors are encountered during compilation the Frame COMMAND will be displayed (this will contain the text of the command procedure) and a pointer (!) will indicate which command is in error. An error message will also be displayed. The command string may be edited to correct the error. It is not necessary to delete any error messages before re-executing the command string, because the pointer (!) is also the comment introducer.

To re-execute the corrected procedure, first return to the appropriate Frame (ER will return to the last used Frame) and then Execute the command procedure with the command EX. Note that by default Cow1t1mIIIG is mapped to be equivalent to "EXecute the Frame COMMAND" .

Example: \^ Command: >(10j 3c 5j d a)

This example would move 10 places to the right, insert 3 spaces, move 5 places to the right, delete one character and advance to the next line. This process would repeat until the end of file, or end of page.

The Execute Suing command is useful in many circumstances in that it allows quite complex procedures to be entered and executed as well as having the procedure displayed so that errors in typing, or construct, can be seen and corrected.

7.2 Execute Immediate

The Execute Immediate facility allows for procedures to be entered directly from the keyboard, without any echoing, by enclosing the procedure in brackets. The left bracket (is used to introduce the procedure which is terminated by a matching right bracket). Thus the above procedure

>(10j 3c sj d a)

could be entered directly via the keyboard, with no prompting and therefore no editing of visible entry. The command will be executed immediately the Return key is pressed. See \(and -\ in Help

7.3 Nesting and Repeat Counts

As seen above, brackets can be used to enclose a given sequence of commands which may be given a repeat count: the repeat count may be a positive or negative integer, or <, or >, which have meanings as described previously.

Nesting of sets of procedures is also possible, to an arbitrary number of levels. This is achieved by enclosing inner procedure sets within an outer pair of brackets. The procedure

4(5(a 10d) Sa)

is an example of a procedure with two levels of nesting. This procedure will satisfy the inner nest by advancing one line and deleting 10 characters

This will be repeated for the next five lines, then the rest of the outer nest will execute, by advancing five lines, and

The whole process will repeat four times.

7.4 Argument Delimiters

For commands which require a string (of characters) as an argument, any balanced pair of non-alphanumeric characters will serve. For example in search strings such as are associated with the Get command the following examples are all valid and will identify the string being searched for -

g/cat/
g.cat.
g'cat'

Some characters have special meaning as string argument delimiters as follows

" Indicates that exact case matching is to take place. That is, upper and lower case are significant. Normal string matching is independent of case.

& The enclosed string is to be used as a prompt for interactive argument entry, and the reply to the prompt is used as the search string.

$ The enclosed string is taken to be a Span, or Frame, name and the contents of the Span, or Frame, are used as the string.

` Indicates that a pattern matching template is to be used (see Section 8)

$$ This, double dollar, delimiter is used to indicate that the contents of the named Span, or Frame, contain the name of another Span, or Frame, the contents of which are to be used as the string. This is also known as de-referencing.

Examples: G&What do you want& will prompt "What do you want “
instead of the default "Get : "

R& &With& will use the default first prompt "Replace : " and will use "With " instead of "By : " as the second prompt.
Where && is used with a null parameter no string the default prompt appropriate to the command will be used.

V&Insert here ?& [i/*Added Text*/] will insert the text "*Added Text*" if the reponse to the prompt "Insert here ?" is a Y or y.

g$$fred$$ will use the contents of the span "fred" as the name of the span whose contents will be used as the search string.

7.5 Comments Within Command Procedures

For command procedures that are complex or often used it may be desirable to include comments within the procedure to explain its actions. The exclamation mark (!) is used as a comment introducer and any text remaining on the line will be ignored by the command parser. As explained previously, when procedures fail the Frame, or Span, that contains the command procedure is displayed with an explanation mark pointing to the failure point. This allows for the command line itself to be corrected without deleting the comment line.

7.6 Cursor Movement Within Command Procedures

Cursor movement within command procedures can be achieved with the commands -

ZB Move Dot left to the previous tab position
ZC Carriage Return, advance Dot to the beginning of the next line
ZD Move Dot down one line
ZH Move Dot to the beginning of the Frame
ZL Move Dot left one character
ZR Move Dot right one character
ZT Move Dot right to the next tab stop
ZZ Delete one character
ZU Move Dot up one line

7.7 Executing Commands in a Span

The capability of Ludwig to allow extensive and complex command procedures to be stored, recalled, compiled and executed gives the editor great power. Commonly used, or particularly complex, command procedures may be written and the file read into a Frame which may then be compiled and executed. The Execute command (EX),which will prompt for the name of the Span to be executed, will compile and execute the named Span, or Frame

The Execute Norecompile command (EN) will execute but not recompile and is useful in situations where a Span of commands will be executed frequently and the time taken to repeatedly compile can be avoided. The command Span Recompile (SR) may be used to compile, or re-compile, a Span or Frame containing Ludwig commands.

The Span Assign command (SA) assigns typed text to the named Span. If the named Span already exists in a Frame the text will be inserted in that Frame.

Example: Assume that the file SWAP_PARA.LUD contains the command string

g'xyz(' n',' =sd'wally' n')' st'wally'

This procedure will
- get the string 'xyz(',
- position dot on the first cornea,
- position dot on the second comma,
- define a span, called wally ,between the first and second commas,
- position dot on the next ')', and then
- transfer the defined span.

The effect of the procedure is two swap the second and third parameters in the procedure xyz.

To make use of this procedure within Ludwig it would be necessary to create a Frame, open the file, read it into the frame and then execute it. This could be achieved with the following commands:

```
ED'fred'                ! create the frame fred
FI'swap_para.lud'       ! open the file
ER                      ! return to the original frame
EX'fred'                ! execute fred.
```

The above sequence of commands is functionally equivalent to the single File Execute command (FX), eg. FX'swap_para.lud'

7.8 Handling Success or Failure Exit Conditions

When an attempt is made to perform any command there are two possible outcomes: success and failure. The exit condition (success or failure) of each command is set on exit from that command. The exit handler is provided so that, depending on the outcome of any given command (whether the exit condition is success or fail), particular actions may be taken.

7.9 Default Action on Command Completion (Exit)

The exit handler takes default action, which may be modified as shown below. The default action is:
on a successful completion, control is passed to the next command, with success set,
on failure of the command, control is passed to the next level of nesting, with failure set.

The simplest example of this default action is a complex command string where after every successful command executed, control is passed to the next sequential command. When one command fails to execute control is returned to the keyboard with failure set, which will sound the bell.

In interactive mode a command typed at the keyboard will -

- on successful completion, wait for the next command
- on failure, sound the terminal bell (indicating failure) and wait for the next command.

In a command procedure with only one level of nesting each command that completes successfully will pass control to the next command, but where one command fails, control will pass to the next outer level of nesting, which in this case would be the keyboard.

For example, the procedure >(g'cat' =d i'dog') will exit with failure when no more occurrences of the string cat can be found, control will be returned to the keyboard and the bell will sound.

7.10 The Exit Handler

The exit handler allows command procedures to take specified actions depending on the completion status of commands. This gives users fine control over command procedures. Exit handlers are only used within command procedures. An exit handler may follow any command and takes the form -

[*success actions : failure actions*]

where success actions and failure actions are any Ludwig procedures. The exit handler must be bounded by square brackets, [ ], and the success and failure branches separated by a colon. Either, or both the branches are optional, and where omitted default actions will be taken.

To understand the actions of exit handlers the examples below will prove helpful. But first the Exit Commands, XS, XF and XA need to be explained. These commands provide the ability to exit the current level of nesting and pass an exit condition to the nominated level of nesting.

XS will exit the current level of nesting, set the exit condition to true (for success), and pass it to the exit handler for the outer level.

XF will exit the current level of nesting, set the exit condition to false (for failure), and pass it to the exit handler for the outer level.

XA will exit back to the keyboard for the next command independently of how many levels of nesting there are. In non-interactive mode XA will abort and quit fromLudwig.

7.11 Examples of Exit Handling

Example 1: <j >(eol [i'<Blank line>'] a)

In this example of a one level procedure Dot is forced to the beginning of the current line (with <j).

Dot is then tested for being at the End-of-line with (eol). The exit handler associated with the eol command has only a success branch so that if Dot is at the beginning of line as well as the end of line the command i'<Blank line>' takes effect and the text <Blank line> will be inserted. The fail branch of the exit handler is unspecified and provides for no action.

Dot is then advanced to the next line with the a command and the loop repeated (from the outer >).

When the Advance command fails, since there is no exit handler, the default action takes place whereby an exit is taken by one level, with a fail condition which in this case will return command to the keyboard with failure (and sound the bell) because only one level of nesting is present.

This example is equivalent to the following procedure where all the default condition handling is made explicit -

<j[:xf] >(eol[i'<Blank line>':] a[:xf]) [:xf]

Example 2:

4(5(a 10d) Sa)
this command procedure is equivalent to

```
4( 5(a[ :xf] 10d[ :xf])  5a[ :xf])
     (------------------)              nesting level 1
  (---------------------------------)  nesting level 2
```

The function of this procedure is to delete the Fust 10 characters from each of the next 5 lines, then skip 5 lines and then repeat for a total of 4 times.

If, at any stage, within nesting level 1, there are not 10 characters on the line to delete the exit handler [:xf] will force an exit to the next level (level 2) of nesting with failure set. Because the failure condition is passed to level 2, it also will exit with failure set. This will cause control to be returned to the keyboard, with failure set which will sound the bell.

The above procedure could be modified so that in the event of a failure to delete 10 characters, it would continue. Thus

4(5(a 10d[:]) 5a)

has a 'no action' as the failure action and so it would continue until completion, or until there were no more lines to advance. Note that "no action" never fails.

8. Pattern Matching

8.1 Rationale for the Pattern Matcher

Ludwig provides three commands that depend upon an ability to match text. These commands test, find, or replace text and are EQS, G, and R, (EQual String, Get, and Replace). In normal use are invoked by specifying the text to be found as a single string (the target string). Often the need arises for a more flexible way of specifying the target string and so Ludwig provides a pattern specification and matching facility. The pattern matcher uses a regular expression language to express the structure of the target pattern.

8.2 Usage of the Pattern Matcher

The reserved delimiter ` (the backquote) is used to indicate a pattern being used instead of a text string target in EQS, G and R.

For example EQS`*pattern specification* `
or R / `*pattern specification* `/ *replacement text* /

8.3 Basic elements of patterns.

The basic element of a pattern is a character class. Ludwig provides a number of useful pre- defined character classes. These are:

a	The alphabetic characters.
l	The lower case alphabetics
u	The upper case alphabetics
n	The numerals '0' through '9'
p	Punctuation characters ' " , . ( ) ! ? -
s	The space character
c	All printable characters

For example

G`sul`	will End a space followed by an upper case letter followed by a lower case letter, such as may occur at the beginning of a sentence.
G`sulllls`	will find a five character word starting with a capital letter.

In much the same way as Ludwig commands may be grouped and structured the pattern matcher provides structuring of pattern elements. The most basic is a repeat count, which is placed immediately before the element to be repeated.

G`su4ls`	is equivalent to the example above.

Two special operators provide indefinite repetition. The Kleene Star, written as "*" indicates that an arbitrary number of occurrences are allowed, including none at all. Whilst the Kleene Plus, written as "+" requires that at least one occurrence to be present.

`s+as`	will match any word of one letter in length or larger.
`su*1s`	the same, but starting with a capital

Patterns may be grouped to form sub-patterns allowing repetitions to apply to the entire construct.

`4(s+a)s`	will match four words separated by a single space.

In addition to the predefined character classes it is possible to define arbitrary character classes. This is done with the D construct, which takes a trailing parameter exactly as for the Next and Bridge commands. Briefly any character between the delimiters is included in the set, and any two characters separated by two periods is interpreted as including all those characters between them in the ASCII character set. A period by itself (or a pair at the beginning or end of the definition) in interpreted as including the period itself in the set. The definition is case specific, so it is necessary to include both the upper and lower case letters needed explicitly.

D'a..z'	the set of characters from "a" to "z" is identical to the predefined set L
D'0..9a..fA..F'	is the set of allowed characters in a hexadecimal number
D'abc'	the set {a,b,c }

It is also possible to invert the contents of a character class. The negation operator – will cause the matcher to look for any character except those in the class.

-a	is all non-alphabetics
-s	is all non-spaces
-D'a..zA..Z0..9'	is all characters not allowed in identifiers.

When used in conjunction with a repeat count it is necessary to enclose the character class specifier in parenthesis. The negation operator can only be used on a single character class, it cannot be used on a compound pattern.

+(-s)	is one or more non-spaces
-(an)	is illegal

8.4 Character Strings.

Strings may be defined as pattern components using the same notation as used by Ludwig trailing parameter in the EQS, G and R commands, eg.

"Hello"	looks for the string *Hello* with case significant.
'Hello'	will match *Hello , hELLO, hello* and *HeLlO*.

The quoted string is simply shorthand for a sequence of character class specifications.

“a”	is identical to D/a/
'Hello'	is identical to D/hH/D/eE/2D/lLl/D/oO/

Thus the class negation operator "-" cannot be used with quoted strings.

When it is necessary to specify a set of alternatives the operator "|" is used.

'hello'|'goodbye'	will match either the string *hello* or the string *goodbye*

.

When used with more complex pattern specifications it is a good idea to use extra sets of parenthesis to be sure of correctly conveying the intention of the pattern.

```
'procedure'*s'('|'function'*s'('
```

will not find the intended pattern (the beginning of either a procedure or function declaration), the alternate will only apply to the '(' and 'function' whereas

```
('procedure'*s'(')|('function'*s'(')
```

will correctly specify the intention.

8.5 Contexts

When using a command procedure in Ludwig, the position of the marks dot and equals is usually important. When a pattern specification is used as a target, the wanted piece of text may be a subset of the entire pattern, but the whole pattern is necessary to uniquely describe the target. The pattern matcher makes it possible to have a leading and trailing context for a pattern that are outside the text selected after a match has completed.

s+as matches a word delimited by spaces but will include the spaces in the match.

A full pattern specification consists of three patterns separated by commas:

left-context , target , right-context

When a match succeeds *equals* (=) will be placed in the position corresponding to the first comma in the pattern and dot in the position corresponding to the second. (When the search direction is reversed, with either a -R or -G, the positions are reversed.)

s,+a,s will find a word in text but will not include the surrounding spaces.

A pattern specification need not contain all three parts. If no context markers are in the pattern, the pattern is interpreted as a middle part. If only one marker is present the pattern is a left then middle context:

left , middle
left , middle , right
,middle
, middle , right
left , , right

Each context must be a pattern specification in its own right. A context separating comma may not appear in the middle of a compound pattern. The syntax (see section xxx) makes this clear, however a simple Me is that a comma may not be placed within parenthesis.

(),((()()),()() is legal
()(,()()),()() is not

8.6 Position markers.

In addition to the character classes the pattern matcher recognises positions within the text.

{ is the left margin
} is the right margin
< is column one (left edge of page)
> is the end of the line
^ is the column that Dot is currently in

Examples:
<'program' will find the string *program* only if it starts in column one.

^'end' if Dot is placed at particular level of indenting this pattern will find *end* at the same level of indent.

Of lesser utility are position specifiers for the normal Ludwig marks.

@n	is one of the marks 1 .. 9
=	the special mark *equals*
%	is the special mark for last change

8.7 Range Specifiers.

In addition to the repeat counts described earlier there is a general range specification.

[*from , to*]

This makes it possible to specify a range of repetitions for a pattern.

[2 ,4] (+ s + a) will match between two and four words.

If the *from* field is left out it defaults to zero repetitions. If the *to* field is left out it defaults to an indefinite number of repetitions.

[,]	is identical to the Kleene star *
[1,]	is identical to the Keene plus +
[2,2]	is identical with a normal repetition count of two.

8.8 Span De-references.

All of the normal de-referencing mechanisms are available within pattern specifications. However a piece of dereferenced text must be a complete sub-pattem. It is not possible to supply an arbitrary fragment of a pattern with the de-referencing mechanism. The normal delimiters are used to specify de-referencing.

$*span-name*$	Uses the contents of the named span
&*prompt-string*&	Prompts with prompt-string for the user to type in the pattern
&&	uses a default prompt

Span de-references may be used for the following parts of pattern definitions.

Any complex pattern, but not including context markers. (Again this is more clear from the syntax description, but any part of a pattern that can be enclosed in parenthesis may be de-referenced.)

$*span-name* $,4a*(s$*span-name-2* $)s

De-referenced spans are treated as if they are sub-patterns enclosed in parenthesis. A range specification may be directly applied to a span de-refererence.

[2,6]$*span-name* $

A character class definition may use the contents of a span.

D$*span-name* $	will define a character class of all the characters in the span (also using the .. range notation).

Character strings may be de-referenced in two ways. First they may be in a span like normal pattern segment. This requires that the span includes the surrounding quotes. This will be inconvenient. Therefore quoted strings can contain a de-reference.

'$*span-name* $'	will take the contents of the named span and treat it as a quoted string within the pattern interpreting the quotes as usual.
'$text$moretext'	will not cause a de-reference since it does not both start and finish with a $.

The span de-reference must occupy the entire string. If it is necessary to match a string that both starts and ends with de-referencing delimiters ($, &) the string must be broken into two parts so that each does not both start and end with the delimiter.

'$not-a-span-name$' becomes '$not-a-span' 'name$'

9. Mapping Commands to the Keyboard

Many Ludwig commands have control key or other keyboard equivalents as default settings. For example the Window Forward command (WF) can be invoked either by pressing the key sequence \WF, by pressing Control/F, or by pressing Next Screen, if available. This mapping of commands onto keyboard keys makes use of commonly used commands much easier.

The **UK** command allows users to set their own keyboard mappings, which may over-rule the defaults as well as adding more. Keyboard mappings may be simple Ludwig commands or command procedures. Normal keyboard keys may not be mapped. Keyboard mapping is restricted to the keys shown below.

The UK command needs two parameters: the key name on the keyboard and the command procedure.

9.1 Key Names

Ludwig uses a standard set of key names for commonly used keys which are available for mapping. These are -

Control keys:	control-a, control-b, ..., control-@, control-_, etc.
Standard keys:	return, line-feed, tab, back-space, delete
Cursor keys:	up-arrow, down-arrow, left-arrow, right-arrow, home, back-tab
the PFn keys:	pfl, pf2, pf3, pf4
Keypad keys:	keypad-0, … keypad-9, keypad-minus, keypad-plus, keypad-enter, and keypad-period
Common keys:	insert-char, delete-char, help, do, …

Function keys:	function-1, ..., function-n shift-function-1,... contro1~function-1, … contro1-shift-function-1 , ... meta-funtion-1, … meta-shift-function-1 , … meta-control-function-1 … meta-control-shift-function-1 , …

Some control characters are interpreted by the operating system and should not be mapped. These characters (such as (Control/Y under VAX/VMS and Control/Z under Unix) must not be used as the command introducer because the Operating System will interpret them and they are not passed to Ludwig.

When mapping keys with a single insert or overtype command, the command must be enclosed in brackets, eg. UK/.../(i'...')/

9.2 Examples of Key Mapping

uk'control-b'wb'	Maps the key <Control/B to the command WB
uk'up-arrow'zu'	Maps the up-arrow key to the command Cursor Up

9.3 Default Keyboard Mapping

Ludwig provides a standard set of keyboard mappings as follows -

Key	command	key	command
control-b	wb	up-arrow	zu
control-d	d	down-arrow	zd
control-e	we	left-arrow	zl
control-f	wf	right-arrow	zr
control-g	ex/command/	home	zh
control-h	zl	back-tab	zb
control-I	zt	insert-char	c
control-j	zd	delete-char	d
control-k	k	insert-line	I
control-l	l	delete-line	k
control-m	zc	help	h&&
control-n	wn	find	g&&
control-p	uc	next-screen	wf
contro1-r	zr	prev-screen	wb
control-t	wt		
control-u	zu		
control-w	ya		
control-^	c		

9.4 Terminal Definitions

The functional aspects of supported terminals are defined in the terminal description file for each supported terminal.

9.4.1 VAX/VMS

Under VAX/VMS, these files are located in the directory with the logical name TERMDESC. Ludwig uses the logical name TRMHND to interpret the definition file for your terminal. This can be changed from the system default value with the DCL command

```
$ define trmhnd <terminal name>
```

If the logical name TERMDESC translates to a file, that file is used; otherwise TERMDESC must translate to a directory and the file name is the same as the name of the terminal handler defined by the logical name TRMHND. The default file type for a terminal description file is .BIN.

9.4.2 Unix

Under Unix, if the environment variable TERMDESC contains a file path, that file is used; otherwise the file named by the environment variable TERM is accessed in the TERMDESC directory, or in "/usrAoca1/lib/termdesc" if there is no environment variable TERMDESC.

9.4.3 Example Terminal Description File

```
! VT100 Keyboard Definition
!
terminal = vt100
ans-terminal = true

! Control Keys
key-tab = ht
key-return = Cr
key-delete = del

! Arrow Keys (both ANSI and application mode)
key-up-arrow            = csi+'A' I ss3+'A'
key-down-arrow          = csi+'B' I ss3+'B
key-left-arrow          = csi+'D' I ss3+'D'
key-right-arrow         = csi+'C' I ss3+'C'

! PF Keys
key-pf1, key-insert-char = ss3+'P'
key-pf2, key-delete-char = ss3+'Q'
key-pf3, key-insert-line = ss3+'R'
key-pf4, key-delete-line = ss3+'S'
```

```
! Keypad
key-keypad-0 = ss3+'p'
key-keypad-1 = ss3+'q'
key-keypad-2 = ss3+'r'
key-keypad-3 = ss3+'s'
key-keypad-4 = ss3+'t'
key-keypad-5 = ss3+'u'
key-keypad-6 = ss3+'v'
key-keypad-7 = ss3+'w'
key-keypad-8 = ss3+'x'
key-keypad-9 = ss3+'y'
key-keypad-minus = ss3+'m'
key-keypad-comma = ss3+'l'
key-keypad-period = ss3+'n'
key-enter = ss3+'M'
```

10. Processes and Sub-processes

Commands to attach to process and subprocess are allowed only in interactive sessions. These features a useful for maintaining the Ludwig editing environment while executing another task, or set of tasks. For example you may wish to read mail that has just been notified to you, but still retain you current editing environment. **US** will allow this, by creating a sub-process in which you can execute the operating system mail command. When you exit from the sub-process you will be returned to the previous Ludwig environment as it was before you executed US

10.1 VAX/VMS

The command Subprocess (US) will attach the terminal to a sub-process of the process running Ludwig. If no sub-process exists, US will create a new one. On subsequent executions US will attach to the same process, if it exists.

The command Parent process (UP) will attach the terminal to the parent process of the process running Ludwig (if there is one). The Ludwig session will be resumed when the terminal is attached to the process running it.

10.2 Unix

The Subprocess command (US) creates a subprocess running the shell specified by the environment variable SHELL, or "/bin/sh" if the environment variable is not defined.

The Parent process command (UP) attaches the terminal to the parent process of the one running Ludwig, if there is one. The Ludwig session will be resumed when the terminal is attached to the process running Ludwig.

11. Operating System Commands

11.1 VAX/VMS

The DCL command which calls Ludwig has the following form:

Ludwig[qualifier ...] [input_file] [output_file]

All parameters and qualifiers are optional.

If Ludwig is invoked with no file specifications, Ludwig will either -

1. Use the last used filename as the input_fi1e and output_file (Ludwig memory stores the last used filename during the current terminal session), or
2. Prompt for the name of an input_file, which will be used as the name of the output_file.

When only one filename is specified it is taken to be an input_file and an output_file of the same name will be created on quitting from Ludwig, unless either the Read_only or Create qualifiers are present.

When both filenames are specified, text is read from the input_file and written to the output_fl11e on quitting.

The following command line qualifiers are optional, the first four characters of each qualifier are significant.

/ATTRIBUTE=attribute

default is A1*1RIBUTE=SAME

This qualifier specifies the record attributes of the output file. The possible values are -'same', 'carriage_return', 'list', and 'fortran'.

'same' copies the record attributes from the input file, and defaults to 'carriage_return' when there is no input file, or the input 81e has an unsupported attribute. Attributes 'none' and 'PRN' are unsupported. 'list' is a synonym for 'carriage_return'.

/CREATE

default is not present

The file is to be created, the specified file must not already exist. Only one file parameter may be given when /CREATE is specified. This qualifier may not be used with /READ_ONLY.

/[N0] INTTIALIZE [=file] default is /INTTIALIZE=LUD_INIT

The initialization procedure is executed after the files specified on the command line have been attached to the default frame and the input file, if any, loaded. The initialization file is executed as a Ludwig command procedure by the Ludwig File Execute command (FX). Note: Defining the logical name LUD_INIT is a convenient way to specify an initialization procedure without having to redefine the ludwig command symbol (see Section 12)

/[NO]MEMORY[=logical name] default is /MEMQRY=LUD_MEMORY

The logical name is used as the input file if no input file is specified. When the output tile is closed its file specification (without the version number) is assigned to the logical name.

/READ_ONLY default is not present
Ludwig is started with only an input file. An output file can be created during the editing session, and all normal editing behaviour can occur. Only one file parameter may be given when /READ_ONLY is specified. This qualifier may not be used with /CREATE.

/SPACE=value default is /SPACE=500000
The number of characters of text allowed in a frame is initialized by this qualifier. The maximum value is 1000000.

/[NOTAB default is /NOTAB
If an output file has carriage_return attribute replace each lot of eight leading spaces in the Line with a tab character.

/TYPE=type default is TYPE=SAME
This qualifier specifies the record type of the output file. The possible values are `same', `variable', `stream_lf', and `numbered'. `same' copies the record type from the input file, and defaults to `variable' when there is no input file, or the input file has type `fixed'. `stream' and 'stream_cr' files are written as type ` stream_lf' .

11.2 Unix

The Unix command which calls Ludwig has the following form:

ludwig [-option ...] [input_file] [output_file]

All parameters and options are optional.

If Ludwig is invoked with no file specifications, Ludwig will either -

1. Use the last used filename as the input_file and output_file (Ludwig memory stores the last used filename), or

2. Prompt for the name of an input_file, which will be used as the name of the output_file.

When only one filename is specified it is taken to be an input_file and an output_file of the same name will be created on quitting from Ludwig, unless either the -r or -c options are present.

When both filenames are specified, text is read from the input_file and written to the output_file on quitting.

The following command line options are optional (Unix)

-b value	Default is -b 1 Create backup copies of the output tiles, such that if more than value versions already exist the oldest version is deleted. Backup files have ~number appended to them. Thus the first backup version of the file eric would be named eric~1, the second eric~2 and so on.
-B	Default is not present. Do not make any backup copies or delete any backup versions.
-c	Default is not present. Create an output file, checking that it does not exist already.
-r	Default is not present. Read only. Do not open an output tile.
-i file	Default is -i ~/.ludwigrc Specify a file to be executed, with FX, after the default frame LUDWIG has been loaded (see Section 12).
-I	Default not present Do not perform any initialisation
-s value	Default is -s 500000 Set the number of characters of text allowed in the default frame.
-m file	Default is -m ~/.lud_memory. Use the first line in the named file as the name of the file to edit. Ludwig will update this file with the path of the current output file.
-M	Default is not present. Do not use or set the Ludwig filename memory.
-t	Default is not present Entab all output files.
-T	Default is -T Disable entabbing of output files.

12. Initialising an Environment

The default environment values that are provided are often not appropriate for individual users. You can specify an environment when Ludwig is invoked, at the operating system level.

Under VAX/VMS this is done with the /INITIALIZE=file qualifier, thus the DCL command

$ ludwig/INIT=your_init_file.lud input_file output_file

Under Unix, use the form

% ludwig –i your_init_file input_file output_file

An example initialisation file follows which will

- Set word wrap on and indent tracking to be the default
- Increase the default size of frames
- Set up a keymap, and
- Define a command procedure to be used

```
! Example Ludwig initialisation file
!
ep'$o=(w,i), $=750000'                  ! Set EP Options
!
uk'keypad-enter'>ys =j >yf =j >yj'      ! Justify the current paragraph from

uk'keypad-p1us'en/swap_letters/'            ! Maps the swap_letters command procedure
sa'swap_letters'-j =sd/temp/ 2j st/temp/'   ! assigns the command procedure to the
                                            ! span "swap_1etters" for later use by the
                                            ! Keypad-plus key
```

Appendix A: Complete Command Summary

A	Advance	Moves forward or backward n lines
BR	Bridge	Bridges any of a set of characters
C	Character insert	Inserts n spaces
D	Delete	Deletes n characters
ED	Edit frame	Changes frames, possibly creating a new one
EK	Kill Edit frame	Destroys a frame
EX	Execute	Executes commands in a span
EN	Execute Norecompile	Executes commands in a span, no recompile
EOL	End of Line	Tests for end of line
EOP	End Of Page	Tests for end of page
EOF	End of File	Tests for end of file
EP	Edit Parameters	Set editor parameters, e.g., margins, options
EQC	Equal Column	Tests for column position of dot
EQM	Equal Mark	Tests for position of mark n
EQS	Equal String	String match at dot
ER	Edit Return	Returns to frame which called current frame
FE	File Edit	Open an Input and Output file (-FE to close)
FGB	Global File Back	Rewinds the global input file
FGI	Global Input File	Opens the global input file (-FGI to close)
FGK	Global File Kill	Closes and deletes the global output file
FGO	Global Output File	Opens the global output fle (-FGO to close)
FGR	File Read	Reads n lines from the global input tile
FGW	File Write	Writes n lines to the global output file
FI	File Input	Opens an input tile (-FI to close)
FK	File Kill	Closes and deletes an output file
FO	File Output	Opens an output file (-FO to close)
FP	File Page	Writes to the output file, reads from input
FT	File Table	Displays a table of current file attachments
FX	File Execute	Read a tile into frame C and execute it
G	Get	Gets the nth occurrence of a string
H	Help	The Ludwig Help system
I	Insert	Insert option --- typed text is inserted
J	Jump	Moves dot left or right
K	Kill	Deletes n lines
L	Insert Line	Inserts n blank lines above dot
M	Mark	Defines a mark**,** nM defines mark 1--9
N	Next Character	Gets an occurrence of any of a set of characters
O	Overtype	Overtype option --- typed text overwrites existing
Q	Quit	Exits from editor
R	Replace	Replaces one string with another
SA	Span Assign	Assigns text to a span
SC	Span Copy	Copies a previously defined span
SD	Span Define	Defines and names a span
SI	Span Index	Lists all spans and frames
SJ	Span Jump	Jumps to the beginning or end of a span
SL	Split Line	Splits a line in two
SR	Span Recompile	Recompiles a span
ST	Span Transfer	Moves a previously defined span (not a copy)
SW	Swap Line	Swaps a pair of lines
UC	Insert CLI	Insert Command Introducer
UK	User Define	Associates a command string with a key
UP	Transfer Control	To parent process
US	Transfer Control	To sub-process
V	Verify	Command procedure interactive verify

WB	Window Back	Moves the window back over the frame
WE	Window End	Moves the window to the end of the frame
WF	Window Forward	Moves the window forward over the frame
WH	Window Height	Sets the height of the window
WL	Window Left	Shifts the window left
WM	Window Middle	Centres the window on dot
WN	New Window	Redisplays the current window
WR	Window Right	Shifts the window right
WS	Window Scroll	Enables scrolling with arrow keys
WT	Window Top	Moves the window to the top of the frame
WU	Window Update	Updates the window without a full re-draw
YA	Word Advance	Advances n words
YC	Centre Line	Centres line between margins
YD	Delete Word	Delete n words
YF	Word Fill	Places as many words as possible on a line
YJ	Line Justify	Expands line to fit exactly between margins
YL	Align Left	Places start of line at left margin
YR	Align Right	Places end of line at right margin
YS	Word Squeeze	Removes extra spaces from line
XA	Exit Abort	Aborts Command Procedure
XS	Exit success	Command Procedure exit with success
EF	Exit Failure	Command Procedure e>dt with failure
ZB	Backtab	Same as BACKTAB key
ZC	Carriage Return	Same as RETURN key
ZD	Cursor Down	Same as DOWN key
ZH	Cursor Home	Same as HOME key
ZL	Cursor Left	Same as LEFT key
ZR	Cursor Right	Same as RIGHT key
ZT	Tab	Same as TAB key
ZU	Cursor Up	Same as UP key
ZZ	Delete	Same as DELETE key
-\	Command	Invoke Command Mode
^	Execute String	Prompts for and executes a Command Procedure
(	Direct Entry	An unprompted version of Execute String
"	Ditto	Copies characters from line above
‘	Ditto from below	Copies characters from line below
*	Case Change	Changes case to upper, lower or edit case
|	Margin	Resets the left margin
?	Invisible Insert	Insert characters invisibly

Appendix B: Complete Command Descriptions

A ADVANCE
= =======
Moves Dot forwards or backwards through the text by lines, each time re-setting Dot to column one. The command will fail if there are insufficient lines below or above the line containing Dot; however >A and <A will never fail. If the command fails, Dot does not move.

EXAMPLES:

```
 9A   move forward 9 lines
-9A   move backward 9 lines
 0A   move to the beginning of the current line
 >A   move to the end of file marker <End of File>
 @A   move to the line containing mark 1
 =A   move to where Dot was previously
```

LEADING PARAMETER: [none, + , - , +n , -n , > , < , @] A

BR BRIDGE CHARACTER
== ================
Skips over occurrences of any of a set of specified characters. If the character at Dot is not in the set specified, Dot does not move, and the command does not fail. Exact case matching is used.

EXAMPLES:

```
  BR'a'    skip occurrences of the character "a" to the right
 -BR'a'    skip occurrences of the character "a" to the left
  BR'abc'  skip occurrences of any of "a", "b" or "c"
  BR'aBc'  skip occurrences of any of "a", "B" or "c"
  BR'a..z' skip occurrences of any lower case alphabetic character
  BR'0..9' skip occurrences of any numeric string
```

LEADING PARAMETER: [none, + , - , , , , ,] BR

C INSERT CHARACTER
= ================
Inserts n spaces to the left of Dot. With a positive leading parameter, Dot does not move relative to the beginning of the line, while with a negative leading parameter, Dot stays on the character it was originally on. The command will fail to insert any spaces if the requested insertion would cause the line length to exceed the implementation line length limit (400 characters).

LEADING PARAMETER: [none, + , - , +n , -n , , ,] C

D DELETE CHARACTER
= ================
Deletes characters at and to the right of Dot for positive parameters, and to the left for negative parameters. The @ parameter specifies the deletion of all characters between the current position and the specified mark. Note that the mark can be to the left or the right of Dot. @D also places all the deleted text into frame OOPS.

EXAMPLES:

```
 9D   delete 9 characters at and to the right of Dot
 >D   delete all characters from the current position to the end of line
 -D   delete a character to the left of Dot
@5D   delete all characters from Dot to mark 5
 =D   delete all characters from Dot to the previous Dot position
```

LEADING PARAMETER: [none, + , - , +n , -n , > , < , @] D

PREFIX E COMMANDS Commands beginning with "E" fall into 4 categories.
==========

1. Most E* commands pertain to Frames. A Frame is a local editing
   environment, containing its own: marks, input and output files, default
   strings for EQuals-String, Get and Replace, and text. All Frames are
   named, the default frame has the name LUDWIG.
   Other frames always created by Ludwig are :
    COMMAND the command frame. FX and ^ place commands here and compile
          and execute them.
    OOPS   all text deleted from any other frame is placed here.
    HEAP   a special scratch frame. Used especially by SA and UK.

ED	Edit frame	Changes frames, possibly creating a new one
EK	Kill Edit frame	Destroys a frame and its attributes
EP	Edit Parameters	Show and/or change editor parameters.
ER	Edit Return	Returns to frame which called current frame

---

2. EX and EN execute the contents of spans as Ludwig command code.

EX	Execute	Compiles and executes commands in a span
EN	Execute Norecompile	Executes commands in a span; no recompilation

---

3. EQ* The EQUALS commands (see EQ)
4. EO* The END OF commands (see EO)

ED EDIT FRAME
== ==========

Change editing frames to the frame with the specified name. If a frame
with that name does not exist, a new (empty) one is created with that name.
Note that the default frame has the name LUDWIG, and that the frame which
holds the current Command Procedure has the name COMMAND. Frame OOPS
contains all text deleted by the K, @D, SA, FX, and ^ commands. Frame HEAP
is used by the SA command to hold any spans it creates, and by the UK
command.

LEADING PARAMETER: [none, , , , , , ,] ED

EK EDIT KILL
== =========

Deletes a frame and its text. The command will fail if there are files
attached to the frame, the frame is currently being edited, or the
frame is a special frame (COMMAND, OOPS, HEAP). The default frame can be
deleted.

LEADING PARAMETER: [none, , , , , , ,] EK

EN EXECUTE NORECOMPILE
== ===================

Executes the compiled code for a span; if there is no compiled code for
the span, the span is compiled first. Thus a span may be modified, or even
deleted, yet its old code retained and executed. Compilation can be forced
by the Span Recompile command SR. The Execute command EX is equivalent to
SR followed by EN.

LEADING PARAMETER: [none, + , , +n , , > , ,] EN

PREFIX EO COMMANDS
==========

Commands beginning with EO are used in Command Procedures to test whether
Dot is at the END OF a construct.

EOL	End Of Line	Tests for end of line
EOP	End Of Page	Tests for end of page
EOF	End Of File	Tests for end of file

EOL END OF LINE
=== ===========

Tests that Dot is at the end of line, which is defined to be the position
immediately to the right of the last non-space character in the line. >EOL
tests that Dot is at or beyond the end of line. <EOL tests that Dot is at
or to the left of the end of line.

LEADING PARAMETER: [none, + , - , , , > , < ,] EOL

EOP END OF PAGE
=== ==========
Tests if Dot is on the null line at the end of page, which is displayed as
<Page Boundary>. Note that EOP succeeds at end of file as well.

LEADING PARAMETER: [none, + , - , , , , ,] EOP

EOF END OF FILE
=== ==========
Test if Dot is on the null line at the end of file, which is displayed as
<End of File>.

LEADING PARAMETER: [none, + , - , , , , ,] EOF

EP EDITOR PARAMETERS
== =================
Each editing frame has a set of parameters, initialized to default values
at frame creation; the parameter values for the current frame may be changed
at any time by the Editor Parameters command. A "$" prefixed to a parameter
assignment refers to the global default value, and the new value will be
used in new frames. An empty trailing parameter (<RETURN> after the prompt)
produces a display of the current parameters in the following order:

  K keyboard :
    =I insert from keyboard (default)
    =O overtype from keyboard
    =C commands executed from keyboard without command introducer

  C interactive command introducer (default is "\")
    The introducer can be a single ASCII characater not in the set
    [' ','A'..'Z','a'..'z'], or any key name supported by the keyboard
    interface. This parameter can only be defined in screen mode.

  S memory space limit (default 500000 characters)

  H screen height

  W screen width

  O editor options: (all off by default)
    =S Show current options
    =W Wrap at right margin
    =I Indentation tracker, <RETURN> to current indentation, not
        margin
    =N Newline when <RETURN> is pressed in insert mode

  M left and right margin settings (default is M=(1,terminal_width))
      The character "." represents the column containing Dot.

  V top and bottom margin settings (default depends on terminal height)

  T set and clear tabs:
    =(c1,c2,...cn) for tabs at columns c1,c2,...cn
    =D return to default tab setting
    =S set a tab at Dot
    =C clear a tab at Dot
    =T set tabs using the current line as a template
    =I insert a line in the text showing tab stops and margins
    =R set tabs and margins using the current line as a ruler--
        as inserted by T=I

EXAMPLES:
  EP'H=15' set the screen height to 15 lines
  EP'O=I' turn on the indentation tracker
  EP'O=-I' turn off the indentation tracker
  EP'M=(12,70)' set left margin to column 12, right margin to column 70
  EP'M=(20)' set left margin to column 20
  EP'M=(,50)' set right margin to column 50
  EP'M=(,.)' set right margin to current Dot position
  EP'$S=300000' set space for any new frame to 300000 characters
  EP'$O=(w,i,n)' set W,I,N options as global defaults

LEADING PARAMETER: [none, , , , , , ,] EP

PREFIX EQ COMMANDS
==========

Commands beginning with EQ are used in Command Procedures to test whether Dot is at a particular place, or at a particular piece of text.

EQC	Equal Column	Tests for column position of Dot
EQM	Equal Mark	Tests for position of mark n
EQS	Equal String	String match at Dot

EQC EQUAL COLUMN
===

Tests the relationship between Dot and the specified column.
Column numbers range from 1 to 400 (an implementation limit).

EXAMPLES:

EQC'9' succeeds if Dot is in column 9
-EQC'9' fails if Dot is in column 9
>EQC'9' succeeds if Dot is at, or to the right of column 9
<EQC'9' succeeds if Dot is at, or to the left of column 9

LEADING PARAMETER: [none, + , - , , , > , < ,] EQC

EQM EQUAL MARK
===

Tests the relationship between Dot and the specified mark.
The Equals and Modified marks are tested for using trailing parameters of "=" and "%".

EXAMPLES:

EQM'9' succeeds if Dot is at mark 9
-EQM'9' fails if Dot is at mark 9
>EQM'9' succeeds if Dot is at, or to the right of mark 9
<EQM'9' succeeds if Dot is at, or to the left of mark 9

LEADING PARAMETER: [none, + , - , , , > , < ,] EQM

EQS EQUAL STRING
===

Matches a string with the text at and to the right of Dot using inexact case matching for alphabetic characters. Exact case matching, pattern matching etc. are available and use the same pattern matching conventions as the commands G and R. Dot moves during the matching process, but returns to its original position after the match.

EXAMPLES:

EQS'cat' succeeds if Dot is at the start of string cat
-EQS'cat' fails if Dot is at the start of string cat
>EQS'cat' succeeds if the text string is equal to or lexically greater than cat
<EQS'cat' succeeds if the text string is equal to or lexically less than cat

LEADING PARAMETER: [none, + , - , , , > , < ,] EQS

ER EDIT RETURN
==

The Edit Frame command ED changes the editing frame, and in doing so leaves a back reference to itself in the new frame. The Edit Return command ER uses the back reference to return to that frame. For example, if ED/B/ were executed in frame A, then the editor would change frames from A to B. If ER were then executed in frame B, the editor would return to frame A, as if ED/A/ were executed. A chain of returns is possible (as are cycles!). Note that ER itself does not leave a back reference.

LEADING PARAMETER: [none, + , , +n , , , ,] ER

EX EXECUTE
== =======

Executes the contents of a span as a Command Procedure. If the procedure has syntax errors, it will not be executed, and the procedure will be displayed in its frame, with error messages. Error messages take the form of comments, the comment marker (!) pointing to the error. The erroneous procedure may be edited; the error messages need not be deleted. After returning to the appropriate editing frame (usually by the Edit Return command ER), the procedure can be re-compiled and executed.

LEADING PARAMETER: [none, + , , +n , , > , ,] EX

PREFIX F COMMANDS
==========

Commands beginning with F pertain to files. A frame may have one input file, and one output file.

Commands beginning with FG pertain to Global Files.
Global files are not attached to frames. (See help entry FG)

FB	File Back	Rewinds the input file of the current frame
FE	File Edit	Opens input and output files for current frame
FI	File Input	Opens input file for current frame (- to close)
FK	File Kill	Closes and deletes an output file
FO	File Output	Opens output file for current frame (- to close)
FP	File Page	Writes to the output file, reads from input file
FS	File Save	Saves contents of current frame
FT	File Table	Displays a table of currently open files
FX	File Execute	Read file into frame COMMAND, compile & execute

FB FILE BACK
== =========

Rewinds the input file attached to the current frame, and loads the file into the frame.

LEADING PARAMETER: [none, , , , , , ,] FB

FE FILE EDIT
== =========

Opens both an input and output file for the current frame. The file specification argument is used for the input file. The output file uses the same file specification with the version number field removed. If not given, the default device and directory are used. This is the normal command for opening a file in a new frame for editing.
The command -FE writes the contents of the frame to the output file, copies any lines left in the input file to the output file, and closes both the input and output files. No output file is created if the frame has not been modified.

LEADING PARAMETER: [none, + , - , , , , ,] FE

PREFIX FG COMMANDS
==========

Commands beginning with FG pertain to Global Files. Ludwig may have one Global Input file, and one Global Output file. Input may be read into any frame, and output written from any frame.

FGB	Global File Back	Rewinds the global input file
FGI	Global Input File	Opens the global input file (- to close)
FGK	Global File Kill	Closes and deletes the global output file
FGO	Global Output File	Opens the global output file (- to close)
FGR	File Read	Reads n lines from the global input file
FGW	File Write	Writes n lines to the global output file

FGB FILE BACK (global)
=== ==========

Rewinds the global input file.

LEADING PARAMETER: [none, , , , , , ,] FGB

FGI FILE INPUT (global)
=== ==========

Opens an input file which is not attached to any particular frame, but which is globally accessible via the Global File Read command FGR, which reads from the global input file into the current frame. The command -FGI closes the global input file. There may only be one global input file.

LEADING PARAMETER: [none, + , - , , , , ,] FGI

FGK FILE KILL (global)
=== ==========

Closes and deletes the global output file.

LEADING PARAMETER: [none, , , , , , ,] FK

FGO FILE OUTPUT (global)
=== ==========

Opens an output file which is not attached to any particular frame, but which is globally accessible via the Global File Write command FGW, which writes to the global output file from the current frame. The command -FGO closes the global output file. There may only be one global output file.

LEADING PARAMETER: [none, + , - , , , , ,] FGO

FGR FILE READ (global)
=== ==========

Reads n lines from the global input file into the current frame, above the current line.

LEADING PARAMETER: [none, + , , +n , , > , ,] FGR

FGW FILE WRITE (global)
=== ==========

Writes lines from the current frame to the global output file, beginning with the current line.

LEADING PARAMETER: [none, + , - , +n , -n , > , < , @] FGW

FI FILE INPUT
== ==========

Opens an input file for the current frame, and loads the file into the frame. The command -FI closes the input file attached to the current frame.

LEADING PARAMETER: [none, + , - , , , , ,] FI

FK FILE KILL
== ==========

Closes and deletes the output file attached to the current frame.

LEADING PARAMETER: [none, , , , , , ,] FK

FO FILE OUTPUT
== ==========

Opens an output file for the current frame. If no trailing parameter is given, and an input file is attached to the current frame, the output file specification defaults to that of the input file, with the version number field removed.

The command -FO writes the contents of the frame to the output file, copies any lines left in the input file (if any) to the output file, and closes the output file. No output file is created if the frame has not been modified.

LEADING PARAMETER: [none, + , - , , , , ,] FO

FP FILE PAGE
== ==========

Moves all text above the line containing Dot to the output file attached to the current frame, if any, and reads more text from the input file, if any. The current line is not written out. Once text is paged to an output file, it may only be edited again by closing the output file re-opening it as an input file.

LEADING PARAMETER: [none, , , , , , ,] FP

FS FILE SAVE
== =========
The current frame is saved: the contents of the current frame are written to the output file; the remainder of the input file (if any) is copied to the output file; the output file is renamed from its temporary name to its final name; backup copies are created; and a new output file is opened.
The editing environment is not altered.

LEADING PARAMETER: [none, , , , , , ,] FS

FT FILE TABLE
== ==========
Displays the current files in a table, giving their status (viz. FI, FO, FGI, or FGO), read status (EOF if an input file has no further input), an indication whether the frame to which they are attached has been modified, the name of the frame to which they are attached, and the file specification.

LEADING PARAMETER: [none, , , , , , ,] FT

FX FILE EXECUTE
== =============
Opens and reads a file into frame COMMAND, closes the file, and then executes frame COMMAND as a Command Procedure. The previous contents of frame COMMAND (if any) are placed in frame OOPS.

LEADING PARAMETER: [none, + , , +n , , > , ,] FX

G GET
= ===

Interactive Use:

Performs a pattern directed string search. The pattern may be a simple string, or an expression in a general pattern matching language. For patterns which are simple character strings, inexact case matching is the default. Exact case matching is performed if the search string is given in double quotes, e.g. "Cat". If a null pattern is specified, the Get command will use the pattern last used in the current frame, if any. The result of a search is displayed, and the user is asked to verify the result. After a successful search, Dot is at the character after the matched string, and the Equals mark (=) is the first character of the string. The position of Dot and Equals are reversed after a backward search.

Command Procedure Use:

The Get command does not seek verification in Command Procedures, so an explicit Verify command can be useful, e.g. G/.../ V&& Note that this Verify command is not part of the Get command.

EXAMPLES:

```
  G'cat'     search for the string cat, Cat, CAT etc.
 3G'cat'     search for the third occurrence of cat, Cat, CAT etc.
-G'cat'      search backwards through the text for the pattern
  G"Cat"     search uses exact case match
  G'cat'=J   move to the start of the found string (i.e. the previous
             position of Dot)
  G'cat'=D   delete the found string
```

LEADING PARAMETER: [none, + , - , +n , -n , , ,] G

H HELP
= ====

To get help on a particular command, enter the command name, and press <RETURN>.
You may ask for the display of any of the sections listed below by typing the section number, then pressing <RETURN>.
<RETURN> by itself, returns you back to Ludwig.
On long help items, <SPACE> will get more help on that item.

Pressing space will eventually get you to Topic 0.

0. Help for Ludwig users
1. Ludwig Basic Command Summary
2. Control Code Commands
3. Complete Command Summary
4. Trailing Parameters
5. Use of the Pattern Matcher
6. Key Map Numbering
7. Ludwig Notices (for latest information on Ludwig)

This command is not allowed in non-interactive mode.

LEADING PARAMETER: [none, , , , , , ,] H

I INSERT
= ======

Interactive Use:

Sets the mode of keyboard entry so that typed characters are inserted into the text rather than overtyping existing text. Note that insertion may cause a line to become longer than the screen width. Insertion will fail when the line has reached its maximum length (400 characters), and at the right margin when the EP wrap option is turned off. If the EP newline option has been turned on, <RETURN> will split the line. (See help on EP.) The O Overtype command places the editor into the overtyping mode of keyboard entry.

Command Procedure Use:

The command takes a trailing parameter in Command Procedures; see the examples below.
n is the number of copies to be inserted (Command Procedures only)

EXAMPLES:

```
  I/cat and dog/       insert the string "cat and dog" at Dot
 9I/cat and dog/       insert the string "cat and dog" at Dot 9 times
  I&&                  prompts for the text to be inserted
  I&Enter name:&       user defined prompt for the text to be inserted
```

LEADING PARAMETER: [none, + , , +n , , , ,] I

J JUMP
= ====

Moves the Dot n characters to the right or left in a line. If the command cannot move the Dot as far as the leading parameter specifies (within the implementation line limit of 400 characters), the command fails, and the Dot is not moved.

EXAMPLES:

```
 9J   advance the Dot 9 characters to the right, moving the window to the
      right if necessary.
-9J   advance the Dot 9 characters to the left, moving the window to the
      left if necessary
 >J   advance to the position after the last non-space character in a
      line, i.e. jump to the end-of-line. If Dot is past the end-of-line,
      the command >J fails
 <J   move the Dot to column 1
 =J   jump to the previous Dot position (not necessarily in the same line)
@9J   jump to mark 9 (not necessarily in the same line)
```

LEADING PARAMETER: [none, + , - , +n , -n , > , < , @] J

K KILL LINE
= =========

Deletes n lines and then positions Dot on the next line in the same column. The command -K deletes the line above , but does not move the Dot. All deleted text is placed in frame OOPS.

EXAMPLES:

```
  9K   delete 9 lines at and below the current line
 -9K   delete 9 lines above the current line
  >K   delete from the current line to the end-of-page
  <K   delete above the current line to the beginning of the page
  @K   delete lines from the current line to the line containing mark 1
  =K   delete lines from the current line to the previous Dot position

LEADING PARAMETER: [none, + , - , +n , -n , > , < , @ ] K

L       INSERT LINE
=       ===========
Inserts n empty lines above the current line, and positions Dot on the
topmost inserted line, in the same column.  The command -L inserts above the
current line, but does not move the Dot.

LEADING PARAMETER: [none, + , - , +n , -n ,   ,   ,   ] L

M       MARK
=       ====
Defines a mark in the text at the Dot.  If text containing a mark is
deleted, the mark becomes attached to the first character to the right of
the deleted text.  The commands which use marks are { A D FGW J K SD SW };
for example, @9J moves the Dot to the position of mark number 9.

EXAMPLES:

    M    define mark number 1
   9M    define mark number 9.  Mark numbers range 1..9
  -9M    undefine mark number 9

LEADING PARAMETER: [none, + , - , +n , -n ,   ,   ,   ] M

N       NEXT CHARACTER
=       ==============
Searches for the nth occurrence of any of a set of specified characters.
The command fails if there are less than n occurrences remaining in the
frame.  Exact case matching is used.

EXAMPLES:

    N'a'        search for the first occurrence of the character "a" forwards
   -N'a'        search for the first occurrence of the character "a" backwards
   9N'a'        search for the ninth occurrence of the character "a" forwards
    N'abc'      search for the first occurrence of any of "a", "b" or "c"
    N'a..z'     search for the first occurrence of any lower case alphabetic

LEADING PARAMETER: [none, + , - , +n , -n ,   ,   ,   ] N

O       OVERTYPE
=       ========

Interactive Use:
----------------
  Sets the mode of keyboard entry so that typed characters are typed over
existing text rather than inserted into the text.  Overtyping will fail at
the right margin when the EP wrap option is turned off.  (See help on EP.)

Command Procedure Use:
----------------------
  The command takes a trailing parameter in Command Procedures; see the
examples below.
n is the number of copies to be overtyped (Command Procedures only)

EXAMPLES:

    O/cat and dog/          overtype the string "cat and dog" at Dot
   9O/cat and dog/          overtype the string "cat and dog" at Dot 9 times
    O&&                     prompts for the text to be overtyped
    O&Enter name:&          user defined prompt for the text to be overtyped

LEADING PARAMETER: [none, + ,   , +n ,    ,   ,   ,   ] O
```

Q QUIT
= ====

Quits the editor altogether. All lines from frames with attached output files are written to their corresponding output files. For each frame with both an input and an output file attached, any remaining lines in the input file are copied to the output file. All files are then closed.

If a frame has an input file, no output file, and has been modified, then Ludwig will display the frame at the point of its last modification, and ask if the user wishes to continue the quit operation. The following responses are possible:

A will cause Ludwig to disregard all frames in this condition
Y will disregard this frame but continue checking for others
N or Q will return to editing in the displayed frame

LEADING PARAMETER: [none, , , , , , ,] Q

R REPLACE
= =======

Interactive Use:

Performs n replacements of one string by another. Each occurrence is verified before replacement. Replying "N" will ignore an occurrence of the target pattern and search for the next, "Q" will abort the entire command, "Y" will replace this occurrence and search for the next if more than one replacement is to be performed, while "A" will perform all remaining replacements without further verification. Replying "M" will display more lines of context aroung this occurrence, in order for the user to verify replacement. The command prompts for both the target pattern and the replacement string unless a <RETURN> is given for the first parameter, in which case they will be the same as used by the previous Replace command in that frame. Pattern matching is the same as for the Get command G.

Command Procedure Use:

Replacement is not verified (see the Get command).

EXAMPLES:

R/cat/mouse/ replace the next occurrence of cat, Cat or CAT etc. by mouse
R/"cat"/mouse/ replace the next occurrence of cat by mouse (exact case match)
R"cat"mouse" an abbreviation for R/"cat"/mouse/
>R/cat/mouse/ replace all occurrences of cat by mouse, from Dot to the end-of-page
9R/X/Y/ replace the next 9 occurrences of x or X by Y
R&&& prompt interactively from a Command Procedure
R&this&that& user defined prompt from a Command Procedure

LEADING PARAMETER: [none, + , - , +n , -n , > , < ,] R

PREFIX S COMMANDS
==========

Except for SW and SL commands beginning with S pertain to Spans. Spans are named pieces of text delimited by two marks. Spans are the basic method of text transport inside Ludwig. Spans may contain Ludwig command code, which is executed with EX or EN. Spans may also be associated with compiled Ludwig commands (created by SR or EX, and used by EN and EX).

SA	Span Assign	Assigns text to a span
SC	Span Copy	Copies a previously defined span
SD	Span Define	Defines and names a span
SI	Span Index	Lists all spans and frames
SJ	Span Jump	Jumps to the beginning or end of a span
SR	Span Recompile	Recompiles a span
ST	Span Transfer	Moves a previously defined span (not a copy)

The SW and SL commands are useful for fast manipulation of text lines.

SL	Split Line	Splits a line in two
SW	Swap Line	Swaps a pair of lines

SA SPAN ASSIGN
== ==========

The Span Assign command SA replaces the contents of a span with a new piece of text. If the span does not exist it is created at the bottom of the frame HEAP. The old contents of the span (if any) are placed in frame OOPS.

EXAMPLES:

```
SA/name/Hello/    places the text "Hello" in the span called NAME.
SA/name/$name2$/  places a copy of the text in the span called NAME2 in
                  the span called NAME.
V$Flag$[ SA/Flag/N/ : SA/Flag/Y/ ]
   replaces the contents of Flag by N if it is currently Y, and Y if N.
```

LEADING PARAMETER: [none, , , , , , ,] SA

SC SPAN COPY
== =========

The Span Define command SD, defines and names spans of text; the Span Copy command SC inserts a named span into the text, immediately to the left of Dot. The command does not delete the original span.

LEADING PARAMETER: [none, + , , +n , , , ,] SC

SD SPAN DEFINE
== ==========

Defines and names a span between the character at Dot and the character at a specified mark. If the mark was before the Dot, then the mark is at the first character of the span and the Dot is at the character after the end of the span. If the mark was after the Dot, the opposite holds. Once a span has been defined, it is not dependent on the the mark, nor on its boundary characters; if either of these are deleted, the span simply contracts. Span definition replaces any previous use of that span name, but names of existing frames cannot be used as span names. Span names are global to all frames. Any non-blank string up to 31 characters in length is a valid name, but leading and trailing spaces are removed, and alphabetic characters are converted to upper case.

EXAMPLES:

```
  SD'cat'     defines a span named cat between the Dot and mark 1
 9SD'cat'     as above, but uses mark 9
 =SD'cat'     as above, but uses the previous position of Dot for the mark
```

LEADING PARAMETER: [none, + , - , +n , , , , @] SD

SI SPAN INDEX
== ==========

Displays all span names and the first 31 characters of each span. All frames and their attached files are also displayed.

LEADING PARAMETER: [none, , , , , , ,] SI

SJ SPAN JUMP
== ==========

Moves Dot to the end of a span (the character to the right of the last character of the span). The command -SJ moves Dot to the first character of the span.

LEADING PARAMETER: [none, + , - , , , , ,] SJ

SL SPLIT LINE
== ==========

Splits a line in two at the current position. The characters in the current line to the left of the current position remain in place, and the rest of the line is moved to a new line, with the current character positioned where a <RETURN> or ZC command would have placed the cursor. This will be the left margin, unless the EP indentation tracker option is turned on. (See help on EP.) Dot stays attached to the first character of the right-hand segment of the line.

LEADING PARAMETER: [none, , , , , , ,] SL

SR SPAN RECOMPILE
== ==========

Recompiles a span for execution. The Span Execute command EX automatically recompiles its span before executing it; the Execute Norecompile command EN does not do so if compiled code for the span exists.

LEADING PARAMETER: [none, , , , , , ,] SR

ST SPAN TRANSFER
== ==========

The Span Define command SD, defines and names spans of text; the Span Transfer command ST inserts a named span into the text, immediately to the left of Dot. Note that the span is moved, not copied.

LEADING PARAMETER: [none, , , , , , ,] ST

SW SWAP LINE
== =========

Transfers the current line to a position after the nth line below it. With a negative leading parameter, the line is moved to a position before the nth line above it. The Dot stays with the line as it moves.

EXAMPLES:

SW	swap the current line with the line below it
3SW	move the current line 3 lines down
-SW	swap the current line with the line above it
-3SW	move the current line 3 lines up
@SW	move the current line to a position before line with the mark

LEADING PARAMETER: [none, + , - , +n , -n , > , < , @] SW

PREFIX U COMMANDS
==========

Commands beginning with U are a miscellaneous group with various functions.

UC	Command Introducer	Types the command introducer into the text
UK	Key Mapping	Maps a command string onto a keyboard key
UP	Parent Process	Attaches the terminal to the parent process
US	Subprocess	Attaches the terminal to a subprocess

UC COMMAND INTRODUCER
== ==========

This command types the command introducer character into the text, inserting or overtyping it according to the current keyboard mode. The command fails if the introducer is not an ASCII character. This command is allowed only in screen mode.

LEADING PARAMETER: [none, , , , , , ,] UC

UK KEY MAPPING
== ==========

This command maps a Command Procedure onto a keyboard key. It takes two parameters:

UK/key name/command procedure/

The key names are defined by the keyboard interface (see Section 6). The command procedure may contain multiple lines of text.

The command will fail if the key name is not supported by the user's terminal. If the compilation of the command procedure fails, the command aborts and errors are reported in the same way as by the Execute EX command. This command is allowed only in screen mode.

If the command procedure consists of a single I or O command, it is necessary to put the command in ( ). For example

uk/keypad-2/(i"2")/

LEADING PARAMETER: [none, , , , , , ,] UK

The following definitions are pre-loaded by Ludwig:

uk/control-b/wb/	uk/up-arrow/zu/
uk/control-d/d/	uk/down-arrow/zd/
uk/control-e/we/	uk/left-arrow/zl/
uk/control-f/wf/	uk/right-arrow/zr/
uk/control-g/ex'command'/	uk/home/zh/
uk/control-h/zl/	uk/back-tab/zb/
uk/control-i/zt/	uk/insert-char/c/
uk/control-j/zd/	uk/delete-char/d/
uk/control-k/k/	uk/insert-line/l/
uk/control-l/l/	uk/delete-line/k/
uk/control-m/zc/	uk/help/h&&/
uk/control-n/wn/	uk/find/g&&/
uk/control-p/uc/	uk/next-screen/wf/
uk/control-r/zr/	uk/prev-screen/wb/
uk/control-t/wt/	
uk/control-u/zu/	
uk/control-w/ya/	
uk/control-^/c/	

V VERIFY
= ======

Verify looks at the first character of its trailing parameter, and if it is either a 'y', a 'Y' or a space then the V command succeeds. If the first character is an 'a' or 'A' then the V command will not prompt again, and will always return success. Entering 'q' or 'Q' causes an ABORT (see XA). Any other response is taken as 'n' or 'N' and V fails.

Example:

V&Shall I Go On?&	prompts "Shall I Go On?"
V/Shall I Go On?/	uses "S" as the answer (watch out for this!)
V&&	prompts "Verify?"

LEADING PARAMETER: [none, , , , , , ,] V

PREFIX W COMMANDS
=================

Commands beginning with W pertain to the management of the visible window (the screen). Normally the window tracks the Dot. The Dot is restricted to be within the EP Vertical Margins, and never off the sides of the screen. In a Command Procedure the screen image is kept up to date, until the Dot moves off the screen, in which case no further updating is attempted unless a WU command forces a new screen image to be made, or the Command Procedure terminates.

WB	Window Back	Moves the window back over the frame
WE	Window End	Moves the window to the end of the frame
WF	Window Forward	Moves the window forward over the frame
WH	Window Height	Sets the height of the window
WL	Window Left	Shifts the window left
WM	Window Middle	Centres the window on Dot
WN	New Window	Redisplays the current window
WR	Window Right	Shifts the window right
WS	Window Scroll	Enables scrolling with arrow keys
WT	Window Top	Moves the window to the top of the frame
WU	Window Update	Updates the window without a full re-draw

WB WINDOW BACK
== ===========

Moves the window back over the frame by n window heights.

LEADING PARAMETER: [none, + , , +n , , , ,] WB

WE WINDOW END
== ==========

Moves the window to the end of the current frame.

LEADING PARAMETER: [none, , , , , , ,] WE

WF WINDOW FORWARD
== ==========

Moves the window forward over the frame by n window heights.

LEADING PARAMETER: [none, + , , +n , , , ,] WF

WH WINDOW HEIGHT
== ==========

Sets the height of the window to n lines. When operating over slow communications lines it may be convenient to reduce the window height. When the leading parameter is omitted, the height of the window is set to the height of the terminal.

LEADING PARAMETER: [none, + , , +n , , , ,] WH

WL WINDOW LEFT
== ==========

Moves the window n characters to the left, if possible. When no leading parameter is given, the window is moved 40 characters left, if possible.

LEADING PARAMETER: [none, + , , +n , , , ,] WL

WM WINDOW MIDDLE
== ==========

Scrolls the window so that Dot is centred between the top and bottom margins.

LEADING PARAMETER: [none, , , , , , ,] WM

WN NEW WINDOW
== ==========

Redisplays the current window. This command is useful when the current display has become corrupted, e.g. by a transmission error, or by a system message.

LEADING PARAMETER: [none, , , , , , ,] WN

WR WINDOW RIGHT
== ==========

Moves the window n characters to the right. When no leading parameter is given, the window is moved 40 characters right. Note that the maximum line length is 400 characters, and the right margin of the screen cannot move past column 400.

LEADING PARAMETER: [none, + , , +n , , , ,] WR

WS WINDOW SCROLL
== ==========

Scrolls the screen n lines with Dot fixed to the text. When no leading parameter is given, the arrow keys scroll until some other key is pressed.

LEADING PARAMETER: [none, + , - , +n , -n , > , < ,] WS

WT WINDOW TOP
== ==========

Moves the window to the top of the current frame.

LEADING PARAMETER: [none, , , , , , ,] WT

WU WINDOW UPDATE
== ==========

Ludwig attempts to maintain the display while a Command Procedure is being executed. If the Dot moves off the screen, no further updating is attempted until the procedure terminates. The Window Update command WU can be used within a Command Procedure to force an update at any time.

LEADING PARAMETER: [none, , , , , , ,] WU

PREFIX X COMMANDS
==========

Commands beginning with X are the EXIT commands. These commands are used within Command Procedures for flow control and error handling. Any command in Ludwig can exit with one of three statuses, Success, Failure, or Abort. Also in a Command Procedure a group of commands (commands within parentheses) may exit with any of the three statuses. If a command or a group of commands is followed by an exit handler the success and failure statuses are used to select action within the handler. An Abort exit status causes the Command Procedure to terminate. The EXIT XA,XS,XF commands are used to explicitly cause a group of commands to exit. The leading parameter indicates the number of levels of parentheses (textual) that should be exited.

Exit handlers have the form
  <command> "[" code executed on success ":" code executed on failure "]"
If the colon is omitted the code (if any) is assumed to be a success handler.

XA	Exit Abort	Aborts Command Procedure
XS	Exit success	Command Procedure exit with success
XF	Exit Failure	Command Procedure exit with failure

XA EXIT ABORT
== ==========

Command Procedure Use:

Aborts all Command Procedure execution, no matter how deeply nested (textually or dynamically), and returns control to interactive keyboard entry; signals failure on return.

LEADING PARAMETER: [none, , , , , , ,] XA

XF EXIT FAIL
== =========

Command Procedure Use:

Transfers control out of a Command Procedure by the specified number of nesting levels; "nesting level" is textual, based on a count of parentheses only, not dynamic. Usually used in exit handlers (see Command Procedures). It signals failure on return.

EXAMPLE:

```
>( 5( J[:2XF] ) A[:XS] ) O/*****/
```

Failure of the command J causes execution of its fail handler, which in this case is the command 2XF. Control transfers out by two levels, i.e. out of both loops. Because XF signals a failure on exit from the loop, the entire procedure fails at that point so O/*****/ is not executed.

LEADING PARAMETER: [none, + , , +n , , > , ,] XF

XS EXIT SUCCESS
== ============

Command Procedure Use:

Transfers control out of a Command Procedure by the specified number of nesting levels; "nesting level" is textual, based on a count of parentheses only, not dynamic. Usually used in exit handlers (see Command Procedures). It signals success on return.

EXAMPLE:

```
>( 5( J[:2XF] ) A[:XS] ) O/*****/
```

Failure of the command A causes execution of its fail handler, which in this case is the command XS. Control transfers out by one level, i.e. out of the outermost loop. Because XS sets the execution status to "success", O/*****/ is executed.

LEADING PARAMETER: [none, + , , +n , , > , ,] XS

PREFIX Y COMMANDS
==========

Commands beginning with Y are the word processor commands. These commands are used to construct Command Procedures, or may be used in sequence, to format text according to normal text layout conventions.

YA	Word Advance	Advances n words
YC	Centre Line	Centres line between margins
YD	Word Delete	Deletes n words
YF	Word Fill	Places as many words as possible on a line
YJ	Line Justify	Expands line to fit exactly between margins
YL	Align Left	Places start of line at left margin
YR	Align Right	Places end of line at right margin
YS	Word Squeeze	Removes extra spaces from line

YA WORD ADVANCE
== ==========

Word Advance moves to the beginning of the nth word, where a word is a contiguous block of non-blank characters. –YA moves to the previous word. 0YA moves to the beginning of the current word. >YA moves to the beginning of the next paragraph, and <YA to the beginning of the current paragraph, where a paragraph is a contiguous block of non-blank lines.

LEADING PARAMETER: [none, + , - , +n , -n , > , < ,] YA

YC CENTRE LINE
== ==========

Centre Line places the current line between the left and right margins, with an equal number of blanks at each end. YC fails if the line is too long to fit between the margins. Dot is placed at the left margin of the next line. >YA centres a paragraph beginning at the current line, where a paragraph is a contiguous block of non-blank lines.

Suggested use is
>YF=J>YC

LEADING PARAMETER: [none, + , , +n , , > , ,] YC

YD WORD DELETE
== ==========

Word Delete deletes n words, where a word is a contiguous block of non-blank characters. >YD deletes to the beginning of the next paragraph, and <YD to the beginning of the current paragraph, where a paragraph is a contiguous block of non-blank lines.

LEADING PARAMETER: [none, + , - , +n , -n , > , < ,] YD

YF LINE FILL
== =========

Line Fill places as many words on the current line (between the margins) as possible, placing words onto the next line, or removing them from the next line, if necessary. A word is a contiguous block of non-blank characters. If text protrudes to the left of the left margin it is not moved. >YF will format an entire paragraph, where a paragraph is a contiguous block of non-blank lines.

Suggested use to format a paragraph is as follows :
Place Dot on the first character of the first word (indented to its intended position).

Ragged Right Format	>YS=J>YF
Ragged Left Format	>YS=J>YF=J>YR
Left and Right Justified	>YS=J>YF=J>YJ

LEADING PARAMETER: [none, + , , +n , , > , ,] YF

YJ LINE JUSTIFY
== ============

Line Justify expands the current line by inserting spaces between words so that the line fits between the margins. If text protrudes to the left of the left margin the protruding text is not modified. The line is not modified if the next line is blank. If the line protrudes past the right margin the command fails. >YJ justifies an entire paragraph from Dot onward, but will not expand the last line of the paragraph. 3YJ justifies the next 2 lines treating the third line as the last line of the paragraph. Dot is placed on the left margin of the next line.

Suggested use to justify a paragraph is as follows :

Place Dot on the first letter of the first word of the paragraph (indented to its intended position).
Then

>YS=J>YF=J>YJ

LEADING PARAMETER: [none, + , , +n , , > , ,] YJ

YL ALIGN LEFT
== ==========

Align Left places the first character of the first word of the current line on the left margin. If the line protrudes to the left of the left margin the command fails. The Dot is placed on the left margin of the next line. >YL aligns an entire paragraph from Dot onward, where a paragraph is a contiguous block of non-blank lines.

LEADING PARAMETER: [none, + , , +n , , > , ,] YL

YR ALIGN RIGHT
== ===========

Align Right places the last character of the last word of the current line on the right margin. If the line protrudes to the right of the right margin the text is not modified. The Dot is placed on the left margin of he next line. >YR aligns an entire paragraph from Dot onward, where a paragraph is a contiguous block of non-blank lines.

LEADING PARAMETER: [none, + , , +n , , > , ,] YR

YS WORD SQUEEZE
== ============

Word Squeeze removes all extra spaces from the current line. Any series of multiple spaces is replaced by a single space. If all the text is to the right of the left margin then the leading spaces are retained. Dot is placed on the left margin of the next line.

Suggested use to format a paragraph is as follows :
Place Dot on the first character of the first word (indented to its intended position).

Ragged Right Format	>YS=J>YF
Ragged Left Format	>YS=J>YF=J>YR
Left and Right Justified	>YS=J>YF=J>YJ

LEADING PARAMETER: [none, + , , +n , , > , ,] YS

PREFIX Z COMMANDS
==========

Commands beginning with Z are the commands that implement the basic movement functions in Ludwig. They are useful within Command Procedures, since the appropriate key cannot be pressed, and they provide some functions not available elsewhere in Ludwig.

The keyboard keys only invoke the associated command by default. It is possible to map a command string onto any of these keys.

ZB	Backtab	Same as <BACKTAB> key
ZC	Carriage Return	Same as <RETURN> key
ZD	Cursor Down	Same as down arrow key
ZH	Cursor Home	Same as <HOME> key
ZL	Cursor Left	Same as left arrow key

ZR	Cursor Right	Same as right arrow key
ZT	Tab	Same as <TAB> key
ZU	Cursor Up	Same as up arrow key
ZZ	Delete	Same as <DELETE> key

ZB BACKTAB
== =======
Moves to the previous tab position on the line, if any. Tabs are set by default on columns 1, 9, 17,... etc. See command EP for information on setting tabs.

LEADING PARAMETER: [none, + , , +n , , , ,] ZB

ZC CARRIAGE RETURN
== ===============
Advances Dot n lines; useful in Command Procedures. If the EP indentation tracker option is turned off, Dot moves either to the left margin or to column one if it was to the left of the left margin already. If the EP indentation tracker option is turned on, Dot moves to the position of the first non-blank character of the line which it was on when the ZC command was executed. (The behaviour of the indentation tracker is actually more complex than this, but is best understood by experimentation.) When working in insert mode, if the EP newline option is turned on, ZC or <RETURN> behaves as Split Line SL.

LEADING PARAMETER: [none, + , , +n , , , ,] ZC

ZD CURSOR DOWN
== ===========
Moves Dot vertically down; useful in Command Procedures.

LEADING PARAMETER: [none, + , , +n , , > , ,] ZD

ZH CURSOR HOME
== ===========

Moves Dot home; useful in Command Procedures. Home is the position in column 1 of the top line of the current screen. However, in order for Dot to stay within the upper vertical margin, the screen may scroll downwards while Dot remains on the character in the home position. The number of lines which the screen must scroll depends on the upper margin setting. (See help on EP.)

LEADING PARAMETER: [none, , , , , , ,] ZH

ZL CURSOR LEFT
== ===========
Moves Dot left; useful in Command Procedures. The command >ZL moves Dot to the current left margin, unless it is to the left of the left margin, in which case it fails.

LEADING PARAMETER: [none, + , , +n , , > , ,] ZL

ZR CURSOR RIGHT
== ============
Moves Dot right; useful in Command Procedures. The command >ZR moves Dot to the current right margin, unless it is to the right of the right margin, in which case it fails.

LEADING PARAMETER: [none, + , , +n , , > , ,] ZR

ZT TAB
== ===
Moves to the next tab position on the line, if any. Tabs are set by default on columns 1, 9, 17,... etc. See command EP for information on setting tabs.

LEADING PARAMETER: [none, + , , +n , , , ,] ZT

ZU CURSOR UP
== =========
Moves Dot up; useful in Command Procedures.

LEADING PARAMETER: [none, + , , +n , , > , ,] ZU

ZZ DELETE
== ======

Equivalent to the keyboard DEL key; useful in Command Procedures.
Equivalent to the command sequence -J O/ / -J when in overtype mode
and -J D when in insert mode.

LEADING PARAMETER: [none, + , , +n , , > , ,] ZZ

\ COMMAND
= =======

Normally, typing at the keyboard causes characters to be entered into the
text being edited, either by overtyping or insertion. The character "\" is
the default command introducer, and when it is typed, it is not entered into
the text, but enables the keyboard to be used to enter the name of a single
editing command. The command will be executed if it is valid.

Normally, the command introducer is needed for every command entry. The
command -\ enables a sequence of editing commands to be entered
interactively without the need for a command introducer. This facility is
especially useful for hand simulating Command Procedures, but should be used
with care. The command \ re-establishes normal text entry from the keyboard
(as do the commands O and I). \ returns Ludwig to the editing mode stored
by the most recent -\ command. I or O put it into Insert or Overtype modes
respectively.

LEADING PARAMETER: [none, + , - , , , , ,] \

^ EXECUTE STRING
= ==============

Provides a line for the interactive specification of a Command Procedure.
Command Procedures entered this way may not be longer than one line on the
screen. On pressing <RETURN>, the command string is placed in frame
COMMAND, the previous contents of which (if any) are placed in frame OOPS.
Frame COMMAND is then compiled, and if there are no syntax errors, executed.

If any syntax errors are detected, frame COMMAND is displayed with the
Command Procedure and error messages. Error messages take the form of
comments, the comment marker (!) pointing to the error. The erroneous
procedure may be edited in frame COMMAND; the error messages need not be
deleted.

After returning to the appropriate editing frame (usually by the
Edit Return command ER), the procedure in COMMAND can be re-compiled and
executed by the Execute command EX. Unless remapped (see the UK command),
CTRL/G is a convenient synonym for EX/COMMAND/.

LEADING PARAMETER: [none, + , , +n , , > , ,] ^

(DIRECT ENTRY EXECUTE STRING
= ============================

The Direct Entry Execute String command "(" is used in exactly the same
way as the Execute String command "^", but it does not provide a line to
type on, nor does it echo the command string to the terminal. The command
string is submitted for execution by a balancing ")", but the command string
is not copied into frame COMMAND. Syntax errors are reported by ringing the
terminal bell at the time the error is typed. Any valid Command Procedure
may be entered this way. This method of command entry is most useful for
Command Procedures stored in terminal function keys.

LEADING PARAMETER [none, + , , +n , , > , ,] (

" DITTO
= =====

Copies characters from the line immediately above the Dot, inserting or
overtyping them at the Dot, and moving the Dot to the right. The command
remains in effect until a key other than " or ' is depressed; thus copying
can be controlled conveniently with repeated entry of the " character. With
a negative leading parameter, copying moves from right to left. The command
' copies from the line below. -" and -' are not allowed in insert mode.

EXAMPLES:

  " copy a character from above the Dot

```
  '          copy a character from below the Dot
 9"          copy nine characters from above and to the right of the Dot
-9"          copy nine characters from above and to the left of the Dot
 >"          copy all characters from above the Dot to the end of line
 <"          copy all characters from above the Dot to column one
 >'          copy all characters from below the Dot to the end of line
 <'          copy all characters from below the Dot to column one
```

LEADING PARAMETER: [none, + , - , +n , -n , > , < ,] "

' DITTO FROM BELOW
= ==========

Copies characters from the line immediately below the Dot, inserting or overtyping them at the Dot, and moving the Dot to the right. The command remains in effect until a key other than " or ' is depressed; thus copying can be controlled conveniently with repeated entry of the " character. With a negative leading parameter, copying moves from right to left. The command ' copies from the line below. -' and -" are not allowed in insert mode.

EXAMPLES:

```
  '          copy a character from below the Dot
  "          copy a character from above the Dot
 9'          copy nine characters from below and to the right of the Dot
-9'          copy nine characters from below and to the left of the Dot
 >'          copy all characters from below the Dot to the end of line
 <'          copy all characters from below the Dot to column one
 >"          copy all characters from above the Dot to the end of line
 <"          copy all characters from above the Dot to column one
```

LEADING PARAMETER: [none, + , - , +n , -n , > , < ,] '

* CASE CHANGE
= ==========

Changes the case of alphabetic characters. The command takes a trailing parameter, either U, L or E (u, l or e) to specify upper, lower or edit case. The command moves the Dot to the right, or to the left with a negative leading parameter. The command remains in effect until a character other than U, L or E is typed, or the end-of-line is reached. No trailing parameter delimiters are required when the Case Change command is used in Command Procedures.

EXAMPLES:

```
 9*U         change nine characters to upper case, from the Dot to the right
-9*U         change nine characters to upper case, from the Dot to the left
 >*L         change all characters to lower case, from the Dot to the end of line
 <*L         change all characters to lower case, from the Dot to column one
 >*E         Changes The Rest Of The Line To Edit Case (This Is Edit Case)
```

LEADING PARAMETER: [none, + , - , +n , -n , > , < ,] *

{ LEFT MARGIN
= ==========

The command { sets the left margin at the current column and -{ sets it to the default left margin for the frame. The left margin is significant for commands such as <RETURN> and Split Line. The left and right margins can also be set by the Editor Parameters command EP.

LEADING PARAMETER: [none, + , - , , , , ,] {

} RIGHT MARGIN
= ==========

The command } sets the right margin at the current column and -{ sets it to the default right margin for the frame. The right margin is significant for interactive text entry. The left and right margins can also be set by the Editor Parameters command EP.

LEADING PARAMETER: [none, + , - , , , , ,] }

? INVISIBLE INSERT
= ==========

The command allows n characters to be entered interactively, and inserted at the Dot in the current frame. However, no prompt is issued, character entry is not echoed, and the insertion does not occur until entry is complete. The leading parameter > allows text to be accepted invisibly until <RETURN> is entered. Invisible insert is useful in Command Procedures which have to prompt for input from the terminal.

LEADING PARAMETER: [none, + , , +n , , > , ,] ?