Marinchip 9900 Word Processor

User Guide

by John Walker

Marinchip Systems 16 St. Jude Road
Mill Valley, CA 94941 ■ (415) 383-1545

Marinchip 9900 Word Processor User Guide

1. Introduction

The Marinchip 9900 Word Processor (WORD) is a comprehensive word processing program. It contains the functions required to perform all of the tasks normally done by word processors such as:

- Form letter preparation
- Automated documentation production
- Composition of text

The unique design of WORD allows well-formatted text to be produced with an absolute minimum of control information and commands. This permits text to be prepared using by users with little or no computer experience without the need of a long training period.

Although WORD is designed to operate with a minimum of commands, it includes a rich set of commands and user facilities to permit advanced users to produce camera-ready copy for printing. These features include:

- User control of all formatting parameters
- Macros with parameters
- Inclusion of library files
- User-defined strings for inclusion in text
- Automatic assignment of section numbers
- Automatic table of contents generation

This manual will first present the use of WORD in a tutorial suited to users with no experience with computer text formatting, then will discuss all the facilities of WORD in a reference manual style. Users familiar with text formatting programs on other computers may wish to skip the tutorial chapter.

1.1. Conventions

Several conventions have been adopted to make this manual more clear.

All examples of input and output from WORD will be indented and printed with a shorter line length that the main text of this manual. This does not indicate that the actual input and output would be indented, but merely serves to distinguish examples from the main text of the manual.

Within the examples, text enclosed in braces like {this} is commentary describing the example. It should not actually be placed in the input to WORD. Text enclosed in brackets like [this] represents optional items. The text explaining an example with optional items will describe under what circumstances the items may be omitted, and what is assumed when they are absent.

2. WORD Tutorial: Formatting text on the M9900

WORD is a text formatting program. It takes text prepared using the text editor (EDIT), or written by another program, and formats it into a readable form. The user is given control over the format used, so that the same text may be printed in many different ways without modification to the text itself. For information on how to enter and modify text files, refer to the manual "Marinchip 9900 Text Editor User Guide".

2.1. Calling WORD

Once the text has been prepared according to the specifications given in this manual, WORD is called to format it. WORD is called by typing the command:

```
    WORD <output file>=<input file>[,<driver>]
```

where <output file> is the name of the disc or device file where the output should be sent, <input file> is the name of the file in which the text has been placed, and <driver> is an optional name of a special output driver to process the text. If <driver> (and the comma before it) is omitted, the standard driver for monospace output will be used. Special drivers will be made available for various proportional spaced devices: contact Marinchip Systems for details.

For example, suppose you have edited the text to be formatted into the file KLUNKMANUAL, and wish to format the text and send the output to the system console, to see how it will look. Enter the command:

```
    WORD CONS.DEV=KLUNKMANUAL
```

and WORD will perform the desired action. To print a final copy on the printer, you would use:

```
    WORD PRINT.DEV=KLUNKMANUAL
```

To save the output from WORD in a file named KLUNK.PRT for examination with the editor, or for later printing, you would use:

```
    WORD KLUNK.PRT=KLUNKMANUAL
```

2.2. Preparing text

WORD is designed so that text simply typed into a file, using the computer just like a typewriter, will be formatted into good looking output. Little, if any, control information is required. The first line of a paragraph is normally indented, or set off from the last line of the preceding paragraph by a blank line. All subsequent lines in the paragraph should begin in column one. WORD will automatically move words from line to line to fill lines as completely as possible, and then will right justify the lines (to make the right margins line up). When a blank line is encountered, or a line is read that does not start in column one, the preceding line will not be justified, nor will words be moved onto it: this serves to delimit a paragraph. If the paragraph is indented in the input text, the indentation will be preserved in the formatted text. For output devices with proportional spacing, the line will be indented the width of the character "m" for each space placed before the first word in the line.

2.2.1. Text preparation example

Let us now look an example of text preparation and formatting. Comments regarding the example are enclosed in (braces).

```
.edit textfile=              ( User calls text editor )
Edit                         ( Editor signs on )
*                            ( Editor prompt.  User hits CR )
Input                        ( Editor enters Input mode)
Any user who has ever        ( User enters text )
tried computer text
formatting must agree
that it is the greatest
thing since the
invention of the pencil.
The ease of correction gained by  ( User waxes poetic )
the use of computer editing is as
large an advance over the technology
of the eraser and white-out paint
as the pencil and paper were over
stone tablets and chisels.
                                        ( Blank line )
Computers are an invaluable aid
to writers, editors, and all
humans who write text and make
mistakes.
                                        ( Null line )
Edit                                    ( Editor returns to edit mode )
*end                                    ( User leaves editor )
.word cons.dev=textfile      ( User calls WORD )
                             ( Formatted text appears )
Any user who has ever  tried  computer  text  formatting
must  agree  that  it  is  the  greatest thing since the
invention of the pencil.  The ease of correction  gained
by  the  use  of computer editing is as large an advance
over the technology of the eraser and white-out paint as
the  pencil  and  paper  were  over  stone  tablets  and
chisels.

Computers are an invaluable aid to writers, editors, and
all humans who write text and make mistakes.
```

In the above example, we wanted the paragraphs to appear as blocks on the page, separated by a blank line, but without indentation. If we wished, instead, to use indentation, we would have left out the blank line between the two paragraphs, and indented the word "Computers" in the second paragraph, say, three spaces. Of course, to be consistent, we would have also indented the first word of the first paragraph ("Any") the same amount. If we had done that, the output from WORD would have read:

```
   Any  user who has ever tried computer text formatting
must agree that it  is  the  greatest  thing  since  the
```

invention of the pencil. The ease of correction gained by the use of computer editing is as large an advance over the technology of the eraser and white-out paint as the pencil and paper were over stone tablets and chisels.
   Computers are an invaluable aid to writers, editors, and all humans who write text and make mistakes.

Note that the input text, messy as it was, has been formatted by WORD into professional-looking right-justified copy. Now let's consider a few of the implications of the way WORD decides where a paragraph starts. Since any line that starts in a column other than column 1 is a new paragraph, WORD will never move text from that line onto a previous line. It follows, then, that multiple lines which start in indented columns will not be merged into a paragraph; they will remain as typed in originally (unless they are too long to fit on the output line, in which case each will be continued onto a new line starting in column one). For example, to have WORD format a letter, the following input might be used.

```
                                  Hoople, North Dakota
                                  September 31, 1997

Dear Mr. Carson,

I regret to inform you that the paper you
found in your trash barrel is not a
genuine manuscript.  Our investigation
indicates that it is what it
originally appeared to be:  wrapping
paper for fish.

                                   Regards,

                                   Percival Slickly
```

WORD will format this letter, filling lines and right justifying the body of the letter (since all of its lines start in column 1), but will leave the address, salutation, and close alone, since they either start in indented columns, or are followed by blank lines. The formatted letter will look as follows:

```
                                  Hoople, North Dakota
                                  September 31, 1997

Dear Mr. Carson,

I regret to inform you that the paper you found in  your
trash   barrel   is   not   a  genuine  manuscript.  Our
investigation indicates that it is  what  it  originally
appeared to be:  wrapping paper for fish.

                                   Regards,

                                   Percival Slickly
```

At this point, you have seen the basics of the operation of WORD, and should be able to use it to produce simple formatted text. Before we undertake the detailed discussion of all the facilities of WORD, we will discuss a few features of WORD useful at the basic level.

2.2.2. Indented paragraphs

Frequently, it is desirable to have a paragraph of text which is indented on the page. If the text were simply entered indented, WORD would not normally fill lines and justify, because the text did not start in column 1, and hence WORD would assume that each line is a new paragraph. WORD can be told that the normal column for paragraph continuation is any column by the line:

```
    .COLUMN=<number>
```

where <number> is the column number to be used for paragraph continuations. The period on the command line must start in column 1 of the line. This is an example of a "command line" to WORD. Such lines control the operation of WORD, but are not directly printed in the output text. There are many other commands, which will be discussed in later sections of this manual.

Following the COLUMN command, text should be typed so that continued lines of paragraphs start in the column specified by <number> on the COLUMN command. As before, a blank line or a line whose first character starts in a column other than the COLUMN setting will form a new paragraph. Note that text to the left of the COLUMN setting will also form a new paragraph: this is called "hanging indentation" and is very useful in composing tables with associated text (see example below). The output text will be indented as many spaces (as wide as an "m") as the input text was indented.

For example, assume the following input were used:

```
    The major advantages of computer
    text formatting are:

    .column=11
      1.      Easy editing of copy using
              standard computer editing programs.
      2.      Instant, clean proof copies
              available at a moment's notice.
      3.      Ability to reformat text to new
              output formats without extensive
              retyping or modification of text.
    .column=1

    These advantages easily justify the use of a
    computer text editing system where any
    volume of text is produced.
```

When formatted by WORD, the output would appear:

```
    The major advantages of  computer  text  formatting
    are:
```

> 1. Easy editing of copy using standard computer editing programs.
> 2. Instant, clean proof copies available at a moment's notice.
> 3. Ability to reformat text to new output formats without extensive retyping or modification of text.
>
> These advantages easily justify the use of a computer text editing system where any volume of text is produced.

Note how the example switched back to column 1 before resuming non-indented text.

2.2.3. Short lines

Frequently, it is desirable to generate output with lines shorter than normal output lines. Examples are long quotations, and indented paragraphs such as those in the previous section. The command:

```
     .LINEWIDTH=-<number>
```

will reduce the length of the line by the number of "m" spaces given by <number>. When the line length is to be increased, the command:

```
     .LINEWIDTH=+<number>
```

should be used. It is important to note that the line width set by the LINEWIDTH command is the total length of the output lines, including any indentation. For a given setting of the line width, text will be right-justified to that width regardless of what is done to the left margin with the COLUMN command. LINEWIDTH is a lot more powerful than this brief discussion indicates: refer to the reference section of this manual for more complete information.

2.2.4. Centered text

To have text automatically centered on a line, use the command:

```
     .CENTER <text>
```

where <text> is simply the text to be centered. For example, the command:

```
     .center Section 4.  Advanced Craziness
```

would print as:

```
                              Section 4.  Advanced Craziness
```

Not only does use of the CENTER command eliminate the burden of counting characters and manually centering, but it automatically adapts the centering to the line width and the spacing characteristics of the output device being used.

2.2.5. Section numbers

WORD will automatically generate the section numbers (like 2.2.5), such as are used in this manual. A line of the form:

```
.<number> <text>
```

will generate a section number with <number> number of levels (between 1 and 9), with <text> as the section title. The numbers will be automatically incremented when subsequent sections with the same level are used, and cleared when a new higher level number is used. For example, the following input:

```
.1 Introduction
.2 Turning on power
.3 Plugging in cord
.3 Throwing "ON" switch
.2 Bringing up system
.3 System loading
.3 Initial setup
.1 Using the system
.2 Common commands
```

would generate the following output:

```
1.  Introduction
1.1.  Turning on power
1.1.1.  Plugging in cord
1.1.2   Throwing "ON" switch
1.2.  Bringing up system
1.2.1.  System loading
1.2.2.  Initial setup
2.  Using the system
2.1.  Common commands
```

Section titles generated in this manner will be preceded and followed by blank lines. Section titles also serve to separate paragraphs. A table of contents containing all section titles, headings, and page numbers where they appeared will automatically be printed at the end of the document if any section numbers were used.

3. Command and text input format

This section will describe, in full detail, the format of input to WORD. Subsequent sections will discuss the commands and variables available to the user to control the formatting capabilities of WORD.

3.1. Input syntax

Input to WORD consists of text and commands. Command lines must contain either a period (.) or an exclamation mark (!) in column one of the line. Following the command line sentinel ("." or "!"), is either a command word, or an assignment to a variable. Command words are simply a word followed by a space or an end of line. An assignment statement consists of a word followed by an equal sign (=) followed by a value to be assigned to that word (numeric or string).

3.1.1. Commands and assignments

Command words may invoke either commands predefined within WORD, or may call user-defined macros. User macros may redefine commands predefined within WORD. Macros may be passed parameters, which are expanded into the macro text. Command names, macro names, and variable names used in assignments are recognised regardless of case: "COLUMN" and "column" are the same variable.

Examples of WORD commands and assignment statements are:

```
     .EJECT                      A predefined command with no parameters.
     .CENTER Common Problems     A predefined command with parameters.
     .LINEWIDTH=70               An assignment of a numeric value.
     .LINEWIDTH=80,JUSTIFY=1     Multiple assignments on one line.
     .FIRSTNAME="Charley"        A string assignment.
```

The exact syntax of each command will be discussed in the specific command descriptions later in this manual.

3.1.1.1. Numbers and spacing parameters

Commands and assignment statements in WORD take parameters which, depending on the command, may either be a simple number or a representation of spacing. Numbers are simple unsigned integers. Spacings are represented internally as multiples of the internal spacing unit of 1/240 inch. Spacings are specified as either an integer or a decimal number with up to two decimal places, optionally followed by a scale factor. Scale factors are single characters defined as follows:

Character	Meaning	Value
i	Inch	240
c	Centimeter	240*(50/127)
P	Pica	240/6
m	Em	See below
n	En	See below
p	Point	240/72
u	Basic unit	1

v	Vert. line	See below

The Em (m), En (n), and Vertical line (v) spacings depend on the properties of the output device being used: they correspond to the physical properties of the characters printed on the device. If a spacing is specified with no scale appended, it will be taken as a multiple of Em spaces if the command refers to horizontal space, and a multiple of Vertical line spaces if the command refers to vertical space. The number specifying the value to be scaled may be a decimal number, but the final result is always kept as an integer multiple of basic units. In addition, numbers are rounded to the physical resolution of the output device being used (with the assurance that a nonzero value will always round to at least 1 space on the physical device).

Note that the case of the scale factor characters is significant. This is the only exception to the rule that WORD ignores the case of letters in non-text information. The following are examples of spacing parameters.

2i	Two inches
0.25c	1/4 centimeter
120u	120 basic units (1/2 inch)
7.5m	Width of 7.5 "m" characters

3.1.2. Text input format

Text is typed in without any special control information. There is a special column known to WORD as the COLUMN setting. This column, initially 1 (the leftmost column on the line), is the column where all text representing the second through last lines of a paragraph must be entered in the input file.

3.1.2.1. Line breaks

Any text line that begins in another column (below or above the COLUMN setting), will be taken as the start of a new paragraph. The start of a new paragraph causes what is referred to as a line break. The occurrence of a line break inhibits justification the last line of the preceding paragraph (if any), and prevents moving words from subsequent lines to previous lines to fill them. Line breaks are caused by:

1. New paragraph
2. Blank line
3. Command line beginning with "."
4. Certain special commands

Commands which begin with "!" will not cause a line break, except for commands which themselves generate output (such as CENTER). Output-generating commands will always cause a line break.

3.1.2.2. Embedded strings

WORD allows the input text to contain special sequences of characters that are expanded into numeric or string values when the text is actually processed. This mechanism allows passing of parameters to macros and files copied into the text, and also allows text to be parameterised. Additionally, it can be used to automatically generate references from one section of the text to another. An embedded string reference consists of the name of the string enclosed in corner brackets (<>).

For example, the string PAGE is predefined by WORD to be the current page number of the output text. The following line of input to WORD:

```
    This is page <page>.
```

will, if output on page 19 of the formatted text, print as:

```
    This is page 19.
```

All string references follow the basic syntax of a name enclosed in corner brackets: only the name of the string identifies it as a predefined string, user string, or macro parameter. Refer to the sections on these types of strings later in the manual for more information on referencing them.

If the name enclosed in corner brackets is not the name of a defined string, the string name, including corner brackets, will be printed unchanged. As a result, text which includes such constructions may be used, as long as none of the names used within corner brackets are defined by WORD. If two corner brackets are encountered in a row, WORD will consider that to represent one corner bracket and not look for a string name. This may be used to "force" string names into output text. For example:

```
    This is page <<page>.
```

will print as:

```
    This is page <page>.
```

in the output text.

4. Commands

This section will discuss the commands available within WORD. Commands generally cause an action to be taken, contrasted to assignments (described in the next chapter), which affect parameters controlling the formatting of text. All commands are invoked by a command character ("." or "!") in column one of an input line, followed by the name of the command (which may be in either upper or lower case). If the command line uses the "." character, it will cause a line break. Commands invoked with the "!" character will not cause a line break, unless they generate output. Commands which always cause a line break will be noted in the command description.

4.1. ABSPOS

```
    .ABSPOS <vertical spacing>
```

The ABSPOS command is a special-purpose command which may be used only within HEADING and FOOTING macros. It causes spacing to be done to position the next printable line at the specified absolute position on the page. If the <vertical spacing> is preceded by a minus sign, the spacing will be relative to the bottom of the physical page. ABSPOS may not be used in normal text. See the section on "Headings and Footings" below for examples of the use of ABSPOS.

4.2. CENTER

```
    .CENTER <text>
```

The text which follows the CENTER command will be centered within the current LINEWIDTH parameter and output as a line. If the text supplied is too long to fit on one line, multiple lines of centered text will be generated. The CENTER command always causes a line break.

Example:

```
    .center A Toolkit for the Quantum Mechanic
```

4.3. COPY

```
    .COPY <filename> <parameters>
```

The COPY command causes the contents of a file to be included in the text as if it were present in the original text in the place of the COPY command. Parameters may be specified which will be expanded as the COPY file is read, permitting the the "canned" COPY text to be filled with words specified in the calling text. The COPY command may be called with the "." sentinel to cause a line break before the file text is included, or with the "!" sentinel to allow the text from the file to be included in a paragraph being formed before the COPY command.

The most straightforward use of COPY is as a mechanism to include a macro library (see below for information about macros). Assume that a set of standard macros are in a file called MACLIB.WRD. They may be made available to WORD by simply including the command:

```
    .COPY MACLIB.WRD
```

in the text of the document before the first macro from that file is referenced (the COPY of the macro library would normally be the first line of the document).

The ability to pass parameters to the text included with COPY allows COPY to be used as a file-based macro facility. A common use of this facility is to produce form letters with inclusions of text. The parameters supplied on the COPY statement are simply strings of characters separated by spaces. Parameters containing spaces may be specified by enclosing them in quotes ("). If it is necessary to include a quote in such a parameter, two consecutive quotes will force a quote into the string. Within the text included with COPY, strings of the form <#n>, where "n" is the parameter number, will be replaced with the text of the corresponding parameter from the COPY command. That is, <#1> will be replaced by parameter 1, <#12> will be replaced by parameter 12, and so on.

Let us assume we have stored the following text in the file JUNKMAIL.

```
                                             Oceanview, Kansas
                                             February 30, 1989

    Dear Mr. <#2>:

    I am writing you this letter because I want you to
    share an opportunity that I recently became
    aware of.  As you know <#1>, opportunity only
    knocks once, and as I'm sure you and all
    the members of the <#2> family will agree,
    we all need every cent we can get to survive in
    today's world.  Yes <#1>, my idea and your
    money can make a lot of money for me.

                                             Sincerely,

                                             Marvin Mailfree
```

The statement:

```
    .COPY JUNKMAIL Tom Turkey
```

will result in the output:

```
                                             Oceanview, Kansas
                                             February 30, 1989

    Dear Mr. Turkey:

    I am writing you this letter because I want you to share
    an opportunity that I recently became aware of.  As you
    know  Tom, opportunity only knocks once, and as I'm sure
    you and all the members of the Turkey family will agree,
    we all need every cent we can get to survive in  today's
```

world. Yes Tom, my idea and your money can make a lot
of money for me.

Sincerely,

Marvin Mailfree

A COPY statement may appear within a file included with another COPY statement. The only limit to the depth of nesting of COPYs is the amount of available memory. One use of COPY that might not be immediately apparent is the statement:

```
.COPY CONS.DEV
```

which will cause text to be read from the system console. This may be used to allow the user to enter text for inclusion in the middle of a document, or to specify formatting parameters by typing commands and assignment statements. When a line is entered consisting of an EOT character (CONTROL D) in column 1, the COPY from the console will terminate and WORD will resume reading following the COPY statement in the file in which it occurred.

To use WORD as an interactive letter generator, it can be called with the command:

```
WORD PRINT.DEV=CONS.DEV
```

which will cause it to take its input from the console. The user at the console can then include standard text from various files by typing COPY statements on the console, specifying whatever parameters are required by the text included.

4.4. EJECT

```
.EJECT
```

The EJECT command causes WORD to immediately advance to the top of the next column of output. If the output format has only one column (the normal case), a new page of output will be started. The EJECT command always causes a line break.

4.5. MACRO

```
.MACRO <macroname>
    . . .
    . . .     Macro body
    . . .
.ENDMACRO
```

The MACRO command declares a macro (body of saved text), which can be later invoked by name. The MACRO facility saves the lines enclosed between the .MACRO command and the .ENDMACRO command. When a line is encountered with the <macroname> as a command word, the saved text (which may be either commands or text) will be included in the text in place of the calling line. Parameters may be specified following the <macroname> on the calling line. These parameters are either simple words delimited by spaces, or strings of characters which may include spaces, delimited by quote marks ("). If a quote is to be included in a quoted string, two consecutive quotes should be written. Within the MACRO text, sequences of characters of the form <#n>, where "n" is the number of a parameter on the macro invocation line, will be replaced by the parameter text.

As an example of MACRO usage, the form letter used as an example for the COPY command above may now be defined as a MACRO as follows:

```
    .macro formletter
                                                Oceanview, Kansas
                                                February 30, 1989

    Dear Mr. <#2>:

    I am writing you this letter because I want you to
    share an opportunity that I recently became
    aware of.  As you know <#1>, opportunity only
    knocks once, and as I'm sure you and all
    the members of the <#2> family will agree,
    we all need every cent we can get to survive in
    today's world.  Yes <#1>, my idea and your
    money can make a lot of money for me.

                                                Sincerely,

                                                Marvin Mailfree
    .endmacro
```

This macro may now be called to generate the actual letter. The call would be as follows:

```
    .formletter Tom Turkey
```

and the output would be identical to that given in the example for the COPY command.

MACROs are invoked exactly like built-in commands, and may be called in both line break (.) and non line break (!) modes. Built-in commands (such as CENTER) can be redefined as MACROs, if desired. MACRO invocations may be nested to any desired depth. MACROs may be defined and called within COPY files, and may themselves call COPY files: there are no restrictions.

Users tend to become confused by the COPY and MACRO facilities because they are so similar and can be used for many of the same purposes. The MACRO facility is the primary tool for storing text to be expanded at multiple locations within one document. MACROs are kept in memory, and are accessed much faster than COPY files. COPY files are intended to include text, commands, and macro definitions from "canned" libraries kept in files. Since the COPY file is not kept in memory, much larger volumes of text may be included from COPY files.

4.6. NEED

```
    .NEED <vertical spacing>
```

The NEED command takes a vertical spacing as an argument. It compares the spacing specified with the space left in the current column, and does an EJECT if less space is left in the column than is requested on the NEED command. For example, if a table is to be included in the text, and you wish to prevent it from being split across two pages, and you know the table is 10 lines long, you would place the command:

```
    .NEED 10
```

before the first line of the table. If less than 10 lines remain in the column, WORD will eject to a new column before printing the table. The <vertical spacing> given on the NEED command is a general spacing, and may be specified in terms of inches, centimeters, picas, etc., by appending a scale factor (See section 3.1.1.1, "Numbers and spacing parameters" above). If no scale factor is supplied, the number will be assumed to mean a number of vertical line spaces.

4.7. NEWPAGE

```
    .NEWPAGE
```

The NEWPAGE command will eject to a new physical page of paper. If the output is single column, the NEWPAGE command is identical in action to the EJECT command. If the output is multiple column, NEWPAGE will always go to a new page, while EJECT will go to the top of the next column on the page.

4.8. SPACE

```
    .SPACE <vertical spacing>
```

The SPACE command will skip the vertical space indicated by the <vertical spacing> parameter. That parameter is a fully general spacing parameter. If no scale factor is specified, normal vertical spaces will be assumed. If the spacing requested exceeds the length left in the column, the SPACE command will be equivalent to an EJECT command (note that white space will not be skipped at the start of the next column). For example, suppose we wish to include an illustration in the finished text. We know that the picture to be pasted into the output from WORD is 3.5 inches tall. We might skip the space for it with the command:

```
    .SPACE 3.5i
```

However, if this command happened to be processed within 3.5 inches of the bottom of a column, WORD would simply eject to a new column, leaving insufficient space in the old column. For this reason, the sequence:

```
    .NEED 3.5i
    .SPACE 3.5i
```

should be used. A blank line in the input file is exactly equivalent to the command:

```
    .SPACE 1
```

5. Assignment statements

Assignment statements are used to set the values of predefined variables, which are numeric variables controlling text formatting, and to set the values of String variables, which are user-defined variables used to specify strings later invoked by name and included in the formatted text. All assignments follow the general syntax:

```
.<variable>=<value>
```

where <variable> is the name of the variable (predefined or string) to be set, and <value> is the value to be stored in the variable. The <value> may be a string, in which case it should be enclosed in quotes ("). A quote may be included in the string by writing two consecutive quotes. Number <value>s may be either simple integers, or spacings representing either horizontal or vertical space. What kind of value is expected is determined from the variable being set by the command. In the discussion of predefined variables below, the expected parameter type will be indicated as follows:

<number>	A simple integer.
<horiz spacing>	Horizontal spacing. If no scale factor is specified, the number will be taken as a number of "m" spaces.
<vert spacing>	Vertical spacing. If no scale factor is specified, the number will be taken as a number of normal vertical spaces.

Assignment statements may begin with either a period (.), which forces a line break, or with an exclamation mark (!), which inhibits a line break. Multiple assignments may appear on one line, separated by commas. For example:

```
.COLUMNCOUNT=2,COLUMNWIDTH=3.5i
```

5.1. Predefined variables

Predefined variables all take numeric values. They control the formatting applied to the output text. Assignments to predefined variables may be either absolute or relative. If the equal sign in the assignment is followed by a number, the assignment is absolute and the variable is set equal to the number. If the equal sign is followed by a plus or minus sign, the number following the sign will be added to or subtracted from, respectively, the current value of the variable. This is referred to as a relative assignment. The three types of assignments are illustrated below:

```
.LINEWIDTH=7i          Absolute assignment
.LINEWIDTH=+0.5i       Positive relative assignment
.LINEWIDTH=-2i         Negative relative assignment
```

5.1.1. COLUMN

```
.COLUMN=<number>
```

The COLUMN variable controls which column in the input text is to be considered the column for continuation of paragraph text. Any line whose first nonblank character is in a column other than that specified by the COLUMN variable will cause a line break. COLUMN is initially set to 1, but may be set to any value to allow indented paragraphs. Note that text appearing to the left of the COLUMN setting will cause a line break, and that only text to the right of the COLUMN setting will be expanded to achieve right justification. This permits "hanging indentation" with no special commands or requirements.

5.1.2. COLUMNCOUNT

```
.COLUMNCOUNT=<number>
```

The COLUMNCOUNT variable specifies the number of columns of text to be placed on the page. The default value is 1 column. The columns controlled by COLUMNCOUNT are newspaper-type parallel columns of text, not character columns. If COLUMNCOUNT is set to 2 or more, make sure the settings of COLUMNWIDTH and LINEWIDTH (described below) are appropriate.

5.1.3. COLUMNHEIGHT

```
.COLUMNHEIGHT=<vert spacing>
```

The COLUMNHEIGHT variable specifies the height of the columns to be placed on the page. When titles and footings are used, COLUMNHEIGHT should be set keeping in mind the amount of space that will be occupied by the title and footing text. COLUMNHEIGHT is initially set to fill the page with text.

5.1.4. COLUMNWIDTH

```
.COLUMNWIDTH=<horiz spacing>
```

The COLUMNWIDTH variable controls the total width of a column of text. This variable is relevant only when multiple columns of text are being placed on the page (COLUMNCOUNT=2 or more), and serves to specify the spacing between columns. A column must be at least as wide as LINEWIDTH (see below), but may be as much wider as desired. The spacing between columns will be equal to COLUMNWIDTH-LINEWIDTH. For example, to construct a page with three columns of text, each two inches wide, with 1/2 inch spacing between the columns, one would use:

```
.COLUMNCOUNT=3
.COLUMNWIDTH=2.5i
.LINEWIDTH=2i
```

5.1.5. FILL

The FILL variable controls whether WORD will move words from line to line (in the absence of line breaks) in order to fill lines with as many words as possible. This variable is initially set to 1, and causes WORD to fill lines. If set to zero, WORD will assume a line break at the end of every input line, regardless of what column the next line starts in. This will cause WORD to transcribe the input text exactly to the output (unless the input line is too long, in which case it will be continued onto a new line with indentation specified by the COLUMN variable). For example, to include a table in the text without worrying about where lines start, the following might be used.

```
    in the following table.
    .fill=0
    Quantity        Item          Price
    18000           O-ring         .02
       14           Washer         .18
    .fill=1
    As you can see from the above table...
```

5.1.6. JUSTIFY

```
    .JUSTIFY=<number>
```

The JUSTIFY variable controls the action taken upon filling a line of text. The initial value is 1, which causes the text to be left and right justified, causing the right margin to be aligned (any indentation is preserved, of course). If set to 0, the alignment of the right margin will not be done, and the text will be left "ragged right".

5.1.7. LINESPACE

```
    .LINESPACE=<vert spacing>
```

The LINESPACE variable controls the normal spacing done between lines of text. The initial value is 1v, which generates single spacing. The spacing between lines can be set to any value desired.

5.1.8. LINEWIDTH

```
    .LINEWIDTH=<horiz spacing>
```

The LINEWIDTH variable specifies the width of text lines. This value is initially provided by the device driver selected, as appropriate to the device being used. The line width may be changed by the user at any time, and is normally used to produce right indentation of a paragraph of text. Since the normal LINEWIDTH used is a function of the selected output device, users are encouraged to use relative specifications when setting the line width, so that text will appear properly formatted regardless of the output device. For example, to indent the right margin of a paragraph:

```
    .LINEWIDTH=-0.5i
           .    .
           .    .    Text
           .    .
    .LINEWIDTH=+0.5i
```

5.1.9. PAGE

```
    .PAGE=<number>
```

The page number counter is set to the specified number. Changes to

PAGE made anywhere within a page will change the value edited by the internal string <PAGE> within the HEADING and FOOTING macros for that page.

5.1.10. PAGEINDENT

    .PAGEINDENT=<horiz spacing>

The PAGEINDENT variable specifies the width of the left margin to be placed before any output text. PAGEINDENT is initially set to zero, causing formatted text to start in the first printable column of the output device. It may be set to any desired value. This is normally used to align the output on the physical page so that the output will avoid perforations, etc. The COLUMN mechanism and simple indentation should be used for indentation of text.

5.1.11. PAGEPAUSE

    .PAGEPAUSE=<number>

The variable PAGEPAUSE is initially set to zero. If set to 1, a printer pause character (ENQ) will be generated at the end of each page printed. The receipt of this character will cause the printer to suspend itself until continued from the system console. PAGEPAUSE should be set to 1 when printing text on individual pieces of paper, such as letterhead stock, where manual intervention is required at the end of each page.

5.1.12. PAGEROMAN

    .PAGEROMAN=<number>

The PAGEROMAN variable controls whether page numbers generated by the <PAGE> internal string (see below) are edited in Arabic or Roman numerals. The initial value is zero, and causes Arabic numerals to be generated. If set to 1, upper case Roman numerals will be edited (e.g., MCMXLVI). If set to 2, lower case Roman numerals will be edited (e.g., mcmxlvi).

5.1.13. TOCMAX

    .TOCMAX=<number>

Normally, all section titles will be placed in the table of contents generated after printing the document. The TOCMAX variable allows control over which titles are included in the table of contents. If TOCMAX is set to a number between 1 and 9, inclusive, only section titles with a level less than or equal to the value assigned to TOCMAX will be placed in the table of contents. If TOCMAX is set to 0, no section titles will be included in the table of contents, and if TOCMAX is set to 10 or higher, all section titles will be included.

5.2. String variables

The string variable feature of WORD allows it to expand named strings appearing within the input text. This feature is used to implement macro parameters, but is also used to make internal values of WORD available to the text being formatted, as well as permitting the user to define strings to be called by name within the text.

Any string variable, regardless of type, is invoked in the text by enclosing the name of the string in corner brackets ("<" and ">"). The corner brackets and string name will be deleted from the input text and the value of the string will be inserted. String names may appear anywhere in the input text, either as words by themselves, or embedded in other words. Strings are also expanded on command lines, so that string values may be used to specify parameters to commands.

If the name enclosed in corner brackets is not a string known to WORD, the name, corner brackets and all, will be left in the text unchanged. The appearance of two corner brackets in a row will prevent WORD from expanding a string name. Hence, to include an item in corner brackets and be sure that WORD will not expand it, write it like:

```
    <<thingie>
```

5.2.1. Macro parameter strings

Macro parameters are discussed above in section 4.5 which describes the MACRO command. Macro parameter strings have names of the form:

```
    <#n>
```

where "n" is the number of the macro parameter. Hence, the first parameter will be <#1>, the second will be <#2>, etc. Macro parameter strings behave exactly like any other string when used within the macro that defines them, but are deleted upon reaching the end of the macro. Since macro calls may be nested, there may be multiple definitions for the macro parameter strings. The user should be aware that only the most recently defined value is available within a macro. Upon termination of that macro, the new value will be deleted, and the next outer value will become available. Since strings may be used within macro parameters, parameters may be passed in to macros called within other macros, solving the problem of parameter referencing. The following is an example of a macro that passes one of its parameters to another.

```
    .macro letter
    Dear <#1>,

    .letterbody <#1>

    Sincerely,
    .endmacro
```

5.2.2. Internal strings

Certain special string names are defined by WORD. These string names provide the ability to insert information regarding the formatted text, such as the current page number, into the text itself. These predefined strings may be redefined by the user (see "User-defined strings" below). Such redefinition makes the predefined value inaccessable throughout the rest of the text. Each internal string is described below.

5.2.2.1. PAGE

The PAGE string represents the current page number of the output text. For example:

    Here we are on page number <page>.

Whether the page number is edited in Arabic, upper case Roman, or lower case Roman numerals is controlled by the PAGEROMAN variable, described in section 5.1.12 above.

5.2.2.2. SECTION

The SECTION string represents the current section number, edited as on the last section number command. For example, the string <section> would be replaced by "5.2.2.2" if placed within this paragraph.

5.2.3. User-defined strings

The occurrence of an assignment statement with a quoted string on the right of the equal sign will create a user-defined string with the value of the specified string. For example:

```
.name="Ichabod"
```

A quote may be included in the string by writing two quotes in a row, and multiple string assignments may be written on one line, possibly intermixed with assignments to numeric variables. For example:

```
.lastname="Crane",height="5' 11"" or so",LINEWIDTH=7i
```

If an assignment is made to a previously-defined string, the old value will be changed to the new value. User-defined strings exist regardless of macro level, and are never deleted automatically by WORD.

5.2.4. Notes on use of strings

The following are useful things to know about how WORD uses strings. This information is intended for the advanced or very inquisitive user of WORD.

5.2.4.1. Strings and the input line length

Like most other software on the M9900 system, WORD limits the input line length to 132 characters: any information beyond that is discarded. This limit applies only to input read from the standard input and from COPY files, it does not apply to lines expanded by the expansion of strings. An input line may be expanded to any length by string expansion, and no information will be lost.

5.2.4.2. String recursion

When a string name is encountered in input text, it is replaced by the value of that string. That value may, in turn, contain other string references, which will be expanded. The only limit to the nesting of string references is the amount of available memory.

5.2.4.3. Using strings for references

Since string expansion occurs in command and assignment statements as well as in normal text, a combination of internal and user-defined strings may be used for automatic generation of references within the text. Such references are currently limited to backward references only. For example, suppose you have a paragraph of text that you wish to refer to later in the text.

```
    .ref1="<page>"
    The frammis dohickey is the software equivalent of the
    blowtorch.  It finds many applications...
```

When WORD processes this statement, it will expand the reference to <page> into the page number on which that statement (and hence the paragraph that follows) occurred. Then, later in the text, a line could say:

```
    Refer to page <ref1> for more information on
    the frammis dohickey.
```

The correct page number saved by the assignment to REF1 will be inserted in the text.

5.2.4.4. Saving macro parameters

Since user-defined strings are all at the same level (unlike macro parameters), they can be created within macros and later used outside. This allows the construction of macros to simplify the specification of strings. Consider the following macro:

```
    .macro person
    .firstname="<#1>",lastname="<#2>"
    .address1="<#3>",address2="<#4>"
    .endmacro
```

This macro might be called as follows:

```
    .person Vlad Dracula "Dracula Castle" Transylvania
```

This would save the user from having to type in (or know) the string names, which could then be used later in the text.

6. Headings and Footings

WORD permits page headings and footings to be generated through the use of special macros. These macros are defined like any other macro, but are automatically called by WORD at the top and bottom of each page. A special command, ABSPOS, is available to these macros to give them exact control over page formatting.

6.1. The HEADING macro

The page heading is defined by the HEADING macro. Whenever WORD reaches the top of a page, it will test whether the user has defined a macro with the name HEADING. If so, that macro will be called (with no parameters). The following is an example of a simple HEADING macro which places a centered document title at the top of each page.

```
     .macro heading
     .center Microprocessor Emulation of the Chainsaw
     .space 1
     .endmacro
```

6.2. The FOOTING macro

The FOOTING macro is defined exactly like the HEADING. When the page has been filled with the number of columns specified by COLUMNCOUNT, each of height specified by COLUMNHEIGHT, the columns will be printed and the FOOTING macro will be invoked. Because the page may not have been completely filled, the FOOTING macro normally uses the ABSPOS command to position to the start of the footing area on the page.

6.3. Using the ABSPOS command

```
     .ABSPOS [-]<vert spacing>
```

The ABSPOS command causes the next printable line to appear at the specified <vert spacing> from the top of the page if no minus sign appears before the <vert spacing>, and from the bottom if a minus sign is supplied. If the position on the page is already below the specified absolute position, the ABSPOS command will be ignored. A typical use of ABSPOS within a FOOTING macro is:

```
     .macro footing
     .abspos -1 { position to last printable line }
     .center -<page>-
     .endmacro
```

6.4. Headings, Footings, and COLUMNHEIGHT

The initial setting of the COLUMNHEIGHT variable is such that the entire page will be filled with text. If headings and footings are to be used, COLUMNHEIGHT must be adjusted by the user to leave room for the heading and footing text. Normally this is done by using a negative relative assignment to COLUMNHEIGHT at the point the HEADING and/or FOOTING macros are declared. For example, if the HEADING and FOOTING occupy a total of 4 lines, one would place the assignment:

```
     .columnheight=-4
```

immediately before the macro declarations.

6.5. Changing headings and footings

The heading or footing may be changed at any time simply by redeclaring the appropriate macro. Redeclaring the macro as the null macro, e.g.,

```
     .macro heading
     .endmacro
```

will turn off the heading or footing. Of course, when the heading and/or footing are changed, COLUMNHEIGHT should be adjusted to compensate for any changes in the height of the heading and footing.

6.6. Formatting variables in headings and footings

Heading and footing macros have their own copy of the formatting variables to prevent interference with the main text. At the time a heading or footing is declared, the formatting variables in effect in the main text at that time will be saved. When a heading or footing macro is invoked, that set of variables will be used, regardless of the variables in effect in the main text at the time of the call. Changes to formatting variables by assignment statements within the heading and footing macros will affect only the heading and footing: the main text will be unchanged. The reason for this somewhat nonobvious mechanism is to prevent interaction between formatting commands in the text and the format of headings and footings. If headings and footings did not have their own copy of the formatting parameters, a change in LINEWIDTH, for example, would change the format of the page title if a page boundary were crossed when LINEWIDTH was set to the nonstandard value.