

WordPerfect

5.1 Macro Manual

Extracted by WPDOS.org

Appendix I: Macros and Merge, Comparison

Macros and Merge are both complex features of WordPerfect that have many similarities, but important differences. Since these features can perform many of the same tasks, this section is provided to help you better understand each feature and distinguish between them.

Before reading this section, we recommend that you familiarize yourself with the Macros and Merge features, as documented in the following sections:

- Workbook Lessons 23, 24, and 25
- Macros
- Macros, Define
- Macros, Execute
- Macros, Macro Editor
- Macros, Message Display
- Merge
- Appendix J: Macros and Merge, Expressions
- Appendix K: Macros and Merge, Programming Commands
- Appendix L: Macros and Merge, Variables

Similarities

Commands

Both Macros and Merge use a similar programming command language. Many of the major programming structures (IF structures, CASE structures, FOR and WHILE loops, global variables, subroutine branching structures) are identical, whether in a macro or merge file. See also *Commands* under *Differences* below.

Macro-to-Merge Interface

Either feature can initiate execution of the other, i.e., a macro can execute a merge, and a merge can execute a macro (by nesting or chaining). This allows you to take advantage of the abilities of both features to accomplish any given task.

Overlap in Abilities

Often the same task can be performed by either a macro or a merge. Depending on the nature of the task, one feature may be more efficient than the other in accomplishing the task.

Repetition

Both Macros and Merge control repetitive processes, and greatly enhance the flexibility of WordPerfect and its ability to assist you in creating the documents you need.

Differences

Command Insertion

Although you can define a macro at the normal editing screen when in macro define mode, macro programming commands must be inserted in the macro via

the Macro Editor. In Merge, the codes are inserted in the document at the normal editing screen. There is no “Merge Editor.”

In Macros, when you select a command from the command access box, only the command itself is entered in the macro. You must then enter the arguments with their appropriate tildes (~) yourself. In Merge, if you select a command that requires arguments, you are prompted for each argument, and WordPerfect enters the tildes for you.

Commands

Although there are many common programming commands between Macros and Merge, many are very different. Sometimes even the same command has important differences in function depending on whether it is in a macro or merge file.

For example, the User Interface commands ({CHAR}, {INPUT}, {PROMPT}, {TEXT}) in Macros allow cursor positioning codes to position the message of the command anywhere on the screen. In Merge, cursor positioning is not allowed. You must use a scrolling technique, or nest a macro to display large messages (see *Message Display* in *Appendix K: Macros and Merge, Programming Commands*).

In Merge, the {LEN} and {MID} commands can take full expressions (see *Appendix J: Macros and Merge, Expressions and Command Syntax* in *Appendix K: Macros and Merge, Programming Commands*) as arguments. In Macros, these commands can take only values as arguments. The differences between Macros and Merge for each command are outlined in *Appendix K: Macros and Merge, Programming Commands*.

In Macros, there are both programming commands and keystroke commands. Keystroke commands (such as {Search}, {Left}, {Home}, {Mark Text}, etc.) cannot be entered in a merge file. See *Keystroke Commands in Variables* in *Appendix L: Macros and Merge, Variables* for more information.

Execution

You execute a macro by using the **Macro** (Alt-F10) key. You execute a merge by using the **Merge/Sort** (Ctrl-F9) key.

In both Macros and Merge, you then enter the name of the file that will control the process. In the case of Macros, you enter the name of a macro file. In the case of Merge, you enter the name of a primary merge file (see *Merge* in *Reference*).

In Macros, execution display defaults to off, but you can turn display on to see every keystroke as it is executed.

In Merge, execution is never displayed. You can use the {REWRITE} command to display on the screen the state of the merged document at that point, but this is only a checking device. You cannot display execution of a merge.

However, in both Macros and Merge, you can use the {STEP ON} command to check each step of execution.

File Format

Macro programming commands are stored in a file in a special format (.WPM format). All macro files must have the extension .WPM (added automatically when you define a macro) which allows WordPerfect to recognize it as a macro file.

Merge programming commands are stored in regular WordPerfect documents. These documents may use any extension you can use for a WordPerfect file.

Formatting of Commands

When editing a macro in the Macro Editor, pressing **Tab** or **Enter** formats the commands, but does not place a code in the macro. The tabs and hard returns are completely ignored when the macro executes.

In Merge, because the non-command part of the file(s) is part of the resulting merged document, the formatting of the commands must be very carefully done. You can still use tabs and hard returns to make the commands more readable, but you must be careful to enclose them in a {COMMENT} command if you want to prevent them from appearing in the merged document (see the description of the {COMMENT} command in *Appendix K: Macros and Merge, Programming Commands*).

General Process

A macro is a recorded series of *keystrokes*. It is executed as if you were performing the same keystrokes at the console. The macro programming commands simply determine *which keystrokes* are executed in *which order*. WordPerfect will perform only those commands that are in the macro.

A merge is the process of combining *text* from multiple sources into a single document. The merge programming commands give you control over the merge, affecting *what text* is merged in *what order* into the resultant document.

It is important, however, to know what parts of the merge WordPerfect handles for you, without a specific command in the merge file. For example, WordPerfect executes the primary file once for each record in a secondary file, unless you insert commands forcing it to do otherwise. WordPerfect also automatically moves the record pointer (see *Record Pointer* at the end of *Appendix K: Macros and Merge, Programming Commands*) in the secondary file from one record to the next when an iteration of the primary file is complete. If you were not aware of how WordPerfect handles the records, you might be tempted to put a {NEXT RECORD} command at the end of the primary file, causing the merge to skip every other record in the secondary file and insert a hard page between each iteration of the resulting document.

Repetition

Macros are most useful for repetitive *keystroke* activity. For example, you might create an *Alt-letter* macro to save and print a document (see *Macros, Define* in *Reference*). The *Alt-letter* macro would replace the following keystrokes with a single keystroke: F10,Enter,y,Shift-F7,f.

Merge is most useful for repetitive *text* activity. For example, you might create a merge document that contains the text of a letter you frequently send to clients. The text of the letter would always be the same, but the client information would vary for each letter. When you perform the merge, the information for each client would then be substituted into each copy of the letter. If you had certain paragraphs that frequently but not always needed to be included in the letter, you could create a more complex merge document that would insert the paragraphs only when certain criteria were met.

Searching

The concept of *searching* is very different between Macros and Merge. Searching in Macros is very similar to searching in a document. You use the ♦Search (F2) or ♦Search (Shift-F2) key (represented by the {Search} and {Search Left} commands in a macro) to search forward or backward in a document for a string of text. Macro execution then depends on whether or not the text is found. You can use the {ON NOT FOUND} command to specify what should occur if the text is not found.

In Merge, *searching* is carried out in the context of *fields* and *records*. You can search a given field in all records of a secondary file, or you can check whether the contents of a given field in the current record are the same as the contents of a variable. But you cannot insert a {Search} or {ON NOT FOUND} command in a merge file to search for a specific string of text and branch accordingly, as with macros.

Records in a secondary file are searched in the order they occur in the secondary file, from top to bottom. You cannot search “backwards” through a secondary file. This order of procedure through records is governed by WordPerfect and cannot be altered using programming commands, except by nesting, substituting, or chaining secondary files. (See *Chaining, Nesting, and Substituting* in *Appendix K: Macros and Merge, Programming Commands* for more information on the {CHAIN ...}, {NEST ...}, and {SUBST ...} commands available in merge. See also *General Process* above.)

Variables

Macros use only global variables. Merge can use the same global variables as Macros, but Merge can also use local variables.

You can also pass information between macro variables and Shell variables. See {SHELL ASSIGN} and {SHELL VARIABLE} in *Appendix K: Macros and Merge, Programming Commands* for more information.

See *Appendix L: Macros and Merge, Variables* and *{VARIABLE}* in *Appendix K: Macros and Merge, Programming Commands* for more information on variables in *Macros and Merge*.

See Also: Lessons 23 through 25; *Macros*; *Macros, Define*; *Macros, Execute*; *Macros, Macro Editor*; *Macros, Message Display*; *Merge*; *Appendix J*; *Appendix K*; *Appendix L*

Appendix J: Macros and Merge, Expressions

Expressions are used to determine values in the {ASSIGN}, {CASE}, {CASE CALL}, {FOR}, {FOR EACH}, {IF}, {SHELL ASSIGN}, and {WHILE} commands in both Macros and Merge, and additionally in the {LEN}, {LOCAL}, {MID}, and {NTOC} commands in Merge (see *Appendix K: Macros and Merge, Programming Commands*). You can also use expressions to determine values for Alt-number variables. Expressions can perform operations on either numbers or strings of text.

An expression can contain up to 129 keystrokes. A keystroke can be a character, an extended character, a keystroke command (in Macros), or a programming command.

Numeric Expressions

The following is a list of numeric expressions. The values must contain only integers (or variables which contain integers).

The highest positive number you should use is 2,147,483,647. Numbers higher than 2,147,483,647 are considered to be negative by WordPerfect (see *Negative Numbers* below). You can use signed numbers in expressions.

When performing multiplication or division, only one number may exceed $\pm 65,535$. For example, $65535*65536$ is legal; $65536*65536$ is not legal.

In the table, the terms n1 and n2 represent number 1 and number 2. Although only a single operator is illustrated in each example below, you can use several operators as well as parentheses in expressions. For definitions of the operations used in this table, see *Expression Terms* below.

| Expression | Operation |
|------------|---|
| !n1 | Returns the logical NOT (bitwise) of the number n1 (see <i>Expression Terms</i> below). Example: !0 is -1. |
| -n1 | Returns the negative of the number n1 (see <i>Negative Numbers</i> below). Example: If variable 1 holds 5, $-\{\text{VARIABLE}\}1$ is -5. |
| n1+n2 | Returns the sum of n1 and n2. Example: 5+4 is 9. |
| n1-n2 | Returns the difference of n1 and n2. Example: 10-1 is 9. |
| n1*n2 | Returns the product of n1 and n2. Example: 6*5 is 30. |
| n1/n2 | Returns the integer quotient of n1 and n2. Examples: 20/5 is 4. 5/2 is 2. |
| n1%n2 | Returns the <i>remainder</i> of the quotient of n1 and n2. Examples: 20%5 is 0. 5%2 is 1. |

| Expression | Operation |
|------------------------|--|
| <code>n1&n2</code> | Returns the logical AND (bitwise) of n1 and n2 (see <i>Expression Terms</i> below). Examples: <code>7&4</code> is 4. <code>3&4</code> is 0. |
| <code>n1 n2</code> | Returns the logical OR (bitwise) of n1 and n2 (see <i>Expression Terms</i> below). Examples: <code>7 4</code> is 7. <code>3 4</code> is 7. |
| <code>n1=n2</code> | Returns a true value (-1) if n1 and n2 are equal; otherwise, returns a false value (0). Example: If variable 1 holds 5, then <code>{VARIABLE}1=5</code> is true and <code>{VARIABLE}1=3</code> is false. |
| <code>n1!=n2</code> | Returns a true value (-1) if n1 and n2 are not equal; otherwise, returns a false value (0). Example: If variable 1 holds 5, then <code>{VARIABLE}1!=3</code> is true and <code>{VARIABLE}1!=5</code> is false. |
| <code>n1>n2</code> | Returns a true value (-1) if n1 is greater than n2; otherwise, returns a false value (0). Examples: <code>6>4</code> is true. <code>4>6</code> is false. |
| <code>n1<n2</code> | Returns a true value (-1) if n1 is less than n2; otherwise, returns a false value (0). Examples: <code>2<10</code> is true. <code>10<2</code> is false. |

If you try to use an invalid numeric expression (e.g., incorrect use of operators, characters other than numbers and valid operators), the expression is simply treated as a text string.

String Expressions

A string is a name for any sequence of one or more characters, including spaces. For example, "Apple", "245", "QB12", "Z", and "Personal Computer" are strings. Keyboard commands (e.g., {Enter}, {HPg}) should be enclosed in string delimiters (" or ') when they are part of an expression (see *String Delimiters* below).

String delimiters must also be used whenever you compare strings. If you are comparing the string contents of two variables, both variable commands must be enclosed in string delimiters. For example, `"{VARIABLE}x"="{VARIABLE}y"`.

The expressions outlined below are used to compare strings. The terms s1 and s2 represent string 1 and string 2.

| Expression | Operation |
|------------|---|
| "s1"="s2" | Returns a true value (-1) if string 1 is identical (including case) to string 2; otherwise, returns a false value (0). Examples: "true"="true" is true. "true"="TRUE" is false. |
| "s1"!="s2" | Returns a true value (-1) if string 1 is not identical (including case) to string 2; otherwise, returns a false value (0). Examples: If variable 1 holds the string "string", then "{VARIABLE}1"!="rope" is true. "{VARIABLE}1"!="string" is false. |
| "s1">"s2" | Returns a true value (-1) if string 1 is greater than* string 2; otherwise, returns a false value (0). Examples: "abcd">"abcd" is true. "a">"A" is true. |
| "s1"<"s2" | Returns a true value (-1) if string 1 is less than* string 2; otherwise, returns a false value (0). Examples: "abcd"<"abcd" is true. "A"<"a" is true. |

*In a string comparison, the WordPerfect character set values are compared. See WordPerfect Character Set Values below for details.

If you do not use the delimiters correctly on s1, the expression is simply treated as a text string. If you do not use the delimiters correctly on s2, the expression evaluates as false.

Expression Evaluation

An expression must be written according to the rules in this appendix so WordPerfect can evaluate it correctly. The following information will help you create and use expressions.

When an expression is encountered in a command, the expression is evaluated first, and the result of the expression is used to complete the command. For example, in the statement {ASSIGN}1~{VARIABLE}1~+1~, the expression is "{VARIABLE}1~+1". When the expression is evaluated, the contents of variable 1 are incremented by one. The assignment is then performed, replacing the old contents of variable 1 with the result of the expression.

In several of the expressions, the result of the operation is either true (-1) or false (0). WordPerfect assigns a numeric value to true and false. These values were chosen because they are opposites (numeric complements) of each other (see *NOT* under *Expression Terms* below).

In the following example, the first assignment statement assigns false (0) to variable 1, the second assignment statement assigns true (-1), the complement of false, to variable 1, and the third assignment statement assigns true (-1) to variable 1.

```
{ASSIGN}1^5=4^
{ASSIGN}1^{VARIABLE}1^
{ASSIGN}1^5!=4^
```

The order in which the various operators are applied in an expression is not simply the order in which they occur. WordPerfect uses an order of precedence that determines which operators are used first, second, etc. For example, in the expression $4+7*8$, there is a different result depending on whether the addition is performed before or after the multiplication. Operator precedence is described below.

Operator Precedence

WordPerfect supports expressions with several operators. Consequently, some order of evaluation must be followed. The operator precedence used by WordPerfect is similar to the accepted precedence for mathematical operators in arithmetic. The following table shows the order that operators are applied:

- 1 - (unary minus), + (unary plus), ! (NOT)
- 2 * (multiply), / (divide), % (mod)
- 3 - (subtract), + (add)
- 4 < (less than), > (greater than), = (equal), != (not equal) (Relational operators also work on strings.)
- 5 & (AND), | (OR)

You can override the operator precedence by placing parentheses around those elements that you want to be evaluated first. Elements inside of parentheses are always evaluated before the elements outside. If parentheses are nested, the innermost parentheses are evaluated first.

In the expression $4+7*8$, the multiplication ($7*8$) is performed first, followed by the addition. This is because multiplication has a higher precedence than addition. The result is 60. If you wanted the addition to be performed first, you would type $(4+7)*8$. In this case, the result is 88.

Notes

Expression Terms

The following are technical terms referenced in the discussion of expressions above. An understanding of these terms is not an essential part of creating macros or performing merges. These definitions are provided for those who are somewhat familiar with programming.

AND (&)

A bitwise AND operation compares the bits of two numbers. When both numbers have a 1 bit in the same position (for example, there is a 1 bit in column 1 (right-most column) of the first number *AND* the second number), a 1 is placed in that position in the result. For example, the expression $21&47$ is evaluated as follows:

| Value | Bits |
|-------|------------------------|
| 21 | 000000000010101 |
| 47 | <u>000000000101111</u> |
| 21&47 | 00000000000101 |

The resulting bits represent the number 5. So, 21&47=5.

Bitwise Operation

In the computer's memory, numbers are represented as a series of sixteen 1s and 0s. Each of the 0s and 1s represents a bit. The pattern for each number is unique. The following table outlines some values and their corresponding bits:

| Value | Bits |
|-------|------------------|
| 0 | 0000000000000000 |
| -1 | 1111111111111111 |
| -3 | 1111111111111101 |
| -21 | 1111111111101011 |
| 3 | 000000000000011 |
| 4 | 000000000000100 |
| 7 | 000000000000111 |
| 21 | 00000000010101 |
| 47 | 000000000101111 |

A bitwise operation works on one column at a time, using a single bit from each number. The operation is done 16 times so each bit of each number is operated on.

Evaluate the Expression

Perform the operation(s) on the expression.

NOT (!)

A bitwise NOT operation takes the bits of the number and complements them. For example, if the expression is !0 (0 is 0000000000000000), the resulting value is -1 (-1 is 1111111111111111).

OR (|)

A bitwise OR operation compares the bits of both numbers. When either number has a 1 bit in the same position (for example, there is a 1 bit in column 1 (right-most column) of the first number *OR* the second number), a 1 is placed in that position in the result. For example, the expression 21|47 is evaluated as follows:

| Value | Bits |
|-------|------------------------|
| 21 | 000000000010101 |
| 47 | <u>000000000101111</u> |
| 21 47 | 000000000111111 |

The resulting bits represent the number 63. So, 21|47=63.

Negative Numbers

In WordPerfect, negative numbers are represented as large positive numbers, from 2,147,483,648 to 4,294,967,295. 4,294,967,295 is -1, 4,294,967,294 is -2, and so on. To determine the number WordPerfect uses to represent any given negative number from -1 to -2,147,483,647, use the following formula:

$$4,294,967,296 - |x|$$

where x is the negative number whose equivalent you are trying to find. For example, to find the WordPerfect equivalent of -3,

$$4,294,967,296 - 3 = 4,294,967,293$$

To find the negative number represented by a given WordPerfect equivalent, use this formula:

$$x - 4,294,967,296$$

where x is the WordPerfect equivalent. For example, to find the negative number represented by 4,294,967,293,

$$4,294,967,293 - 4,294,967,296 = -3$$

You can assign variables to be negative numbers by using the minus (-) operator (see *Numeric Expressions* above), or by using the WordPerfect equivalent. Do not use commas or other punctuation in the WordPerfect equivalent. For example,

```
{ASSIGN}number^-1~
```

is the same as

```
{ASSIGN}number^4294967295~
```

String Delimiters

A string delimiter is a character which marks the beginning or end of a string. In string operations, the " and ' characters serve as string delimiters. Delimiters must be paired correctly. For example, the delimiters in "string" and 'string' are correctly paired, but in "string' they are not. However, one string *can* use the " character while the other uses the ' character (e.g., "string"='string').

Whenever you compare any two items that are not numbers, you must use string delimiters around both strings.

WordPerfect Character Set Values

WordPerfect assigns a unique value to each character in each WordPerfect character set (see the descriptions for the {KTON} and {NTOK} commands in *Appendix K: Macros and Merge, Programming Commands*). This is called the WordPerfect character set value. In a string comparison, the character set values are compared.



For characters in the same character set, one character is considered “less than” another character if the first character comes before the second character. For example, in character set 0, “3” is less than “4” and “A” is less than “a”.

For characters in different character sets, the character from the character set with the lower numerical value is considered “less than” the character from the higher numerical character set. For example, any character from character set 2 is less than any character from character set 3.

See Also: Macros; Macros, Define; Macros, Execute; Macros, Macro Editor; Merge; Appendix I; Appendix K; Appendix L

Appendix K: Macros and Merge, Programming Commands

The WordPerfect Programming Language commands let you control how macros and merges function. Those familiar with programming will recognize many commands as similar to those in other programming languages.



















Some of the commands described in this section can be used only in macros, others only in merge files, and others can be used in both. The  icon designates a command that is available in Macros; the  icon designates a command that is available in Merge. The presence of both icons indicates that the command is available in both features. When you see an icon in parentheses next to the command under the Programming Commands heading, it means that the command itself is not available in the feature, but there is an equivalent (or nearly equivalent) command or method in the feature that performs the same function as the command being described. Be sure to read further under the feature subheading for additional information.

Command Types

The programming commands can be categorized by the functions they perform. The categories are User Interface; Flow Control; Macro, Merge, or Subroutine Termination; External Condition Handling; Macro Execution; Variables; System Variables; Execution Control; Programming Aids; and Keystroke Commands.















User Interface

These commands communicate with the user. They display a prompt, allow input from the keyboard, or both. ({BELL} rings a bell.)

| | | |
|-----------------|---|---|
| {BELL} |  |  |
| {CHAR} |  |  |
| {INPUT} |  |  |
| {KEYBOARD} | |  |
| {LOOK} |  |  |
| {ORIGINAL KEY} |  | |
| {PAUSE} |  | |
| {PAUSE KEY} |  | |
| {PROMPT} |  |  |
| {STATUS PROMPT} |  |  |
| {TEXT} |  |  |

Flow Control

These commands can change the flow of macro or merge execution.

| | | |
|-------------------|---|---|
| {BREAK} |  |  |
| {CALL} |  |  |
| {CASE} |  |  |
| {CASE CALL} |  |  |
| {CHAIN} |  | |
| {CHAIN MACRO} | |  |
| {CHAIN PRIMARY} | |  |
| {CHAIN SECONDARY} | |  |
| {ELSE} |  |  |

| | | | |
|--------------------|--|-------|----|
| {END FOR} | | | Y |
| {END IF} | | 000 | YY |
| {END WHILE} | | 0000 | YY |
| {FOR} | | 0000 | Y |
| {FOR EACH} | | 0000 | |
| {GO} | | | Y |
| {IF} | | 0 000 | YY |
| {IF BLANK} | | | Y |
| {IF EXISTS} | | 0 0 | YY |
| {IF NOT BLANK} | | | Y |
| {LABEL} | | | |
| {NEST} | | 00 | Y |
| {NEST MACRO} | | | Y |
| {NEST PRIMARY} | | | YY |
| {NEST SECONDARY} | | | YY |
| {NEXT} | | 00 | YY |
| {ON CANCEL} | | 000 | YY |
| {ON ERROR} | | | Y |
| {ON NOT FOUND} | | 000 | |
| {OTHERWISE} | | 000 | YY |
| {PROCESS} | | | |
| {QUIT} | | 000 | YY |
| {RESTART} | | 000 | |
| {RETURN} | | 000 | Y |
| {RETURN CANCEL} | | 000 | YY |
| {RETURN ERROR} | | 000 | Y |
| {RETURN NOT FOUND} | | 000 | |
| {SHELL MACRO} | | 00 | |
| {STOP} | | | YY |
| {SUBST PRIMARY} | | | YY |
| {SUBST SECONDARY} | | | YY |
| {WHILE} | | 0 | Y |

Macro, Merge, or Subroutine Termination

These commands will terminate a macro, merge, or subroutine.

| | | | |
|--------------------|--|------|----|
| {BREAK} | | 00 | Y |
| {QUIT} | | 000 | Y |
| {RESTART} | | 0000 | |
| {RETURN} | | 0000 | Y |
| {RETURN CANCEL} | | 000 | YY |
| {RETURN ERROR} | | 000 | Y |
| {RETURN NOT FOUND} | | 00 | |
| {STOP} | | | Y |

External Condition Handling

These commands determine how a condition outside of macro or merge execution is responded to (e.g., when Cancel is pressed), or they create the condition.

| | | |
|--------------------|----|----|
| {CANCEL OFF} | CC | YY |
| {CANCEL ON} | CC | YY |
| {ON CANCEL} | CC | YY |
| {ON ERROR} | CC | YY |
| {ON NOT FOUND} | CC | |
| {RETURN CANCEL} | CC | YY |
| {RETURN ERROR} | CC | YY |
| {RETURN NOT FOUND} | CC | |

Macro Execution

These commands start a macro.

| | | |
|-----------------------|----|---|
| {ALT <i>letter</i> } | CC | |
| {CHAIN} | CC | |
| {CHAIN MACRO} | | Y |
| {KEY MACRO <i>n</i> } | CC | |
| {NEST} | CC | |
| {NEST MACRO} | | Y |
| {SHELL MACRO} | CC | |
| {VAR <i>n</i> } | CC | |
| {VARIABLE} | CC | Y |

Variables

These commands assign a value to a variable, determine the state of a variable, or execute (write out) a variable.

| | | |
|------------------|----|----|
| {ASSIGN} | CC | YY |
| {CHAR} | CC | YY |
| {IF EXISTS} | CC | YY |
| {LEN} | CC | YY |
| {LOCAL} | | YY |
| {LOOK} | CC | YY |
| {MID} | CC | YY |
| {NEXT} | CC | YY |
| {SHELL ASSIGN} | CC | |
| {SHELL VARIABLE} | CC | |
| {SYSTEM} | CC | YY |
| {TEXT} | CC | YY |
| {VAR <i>n</i> } | CC | |
| {VARIABLE} | CC | Y |

System Variables

These commands are used to determine the value of system variables.

| | | |
|----------------|----|---|
| {DATE} | | Y |
| {ORIGINAL KEY} | CC | |

| | | |
|----------|----|---|
| {STATE} | 00 | |
| {SYSTEM} | 00 | Y |

Execution Control

These commands affect the speed or visibility of execution on the screen.

| | | |
|---------------|------|---|
| {DISPLAY OFF} | 00 | |
| {DISPLAY ON} | 0000 | |
| {MENU OFF} | 0000 | |
| {MENU ON} | 00 | |
| {REWRITE} | | Y |
| {SPEED} | 00 | |
| {WAIT} | 00 | Y |

Programming Aids

These commands can be used as programming aids.

| | | |
|---------------|------|---|
| {;} (Comment) | 00 | |
| {BELL} | 00 | Y |
| {COMMENT} | | Y |
| {DISPLAY OFF} | 00 | |
| {DISPLAY ON} | 0000 | |
| {SPEED} | 0000 | |
| {STEP OFF} | | Y |
| {STEP ON} | 0000 | Y |

Keystroke Commands

These commands are mapped to certain keys if you have a keyboard with an enhanced BIOS. However, if you do not have a keyboard with an enhanced BIOS and you have not mapped these commands to keys on your keyboard, you must insert these commands from the Macro Commands menu to achieve their function in a macro (see *Action* under *Keyboard Layout, Edit* in *Reference* for information on mapping keys).

| | |
|----------------|--------|
| {Block Append} | 00 |
| {Block Copy} | 0000 |
| {Block Move} | 0000 |
| {Item Down} | 000000 |
| {Item Left} | 000000 |
| {Item Right} | 000000 |
| {Item Up} | 000000 |
| {Para Down} | 0000 |
| {Para Up} | 0000 |

Command Insertion

For information on how to insert Macro commands, see *Macros, Macro Editor* in *Reference*. For information on inserting Merge commands, see *Inserting Merge Commands* under *Merge* in *Reference*.

Command Syntax

Many of the commands use *parameters* or *arguments* which require a tilde (~) at the end. If the tilde is missing, the macro or merge will not work correctly and may use subsequent commands as part of the arguments for the current command. If a macro or merge is not working properly, check to make sure all the comments and commands have their tilde marks correctly placed. (See also *Troubleshooting* under *Notes* at the end of this appendix.)

The syntax and arguments for each command are displayed in the command access box when you insert the command, and in each command heading below. In the arguments, *var* represents a variable. *Expr* represents a number, string, variable, command, or expression (or a combination). Additional argument types are described under each command.

Programming Commands

The programming commands available in Macros and Merge are listed below with information about their use. Examples are included to clarify the instructions.

The Macros Keyboard Definition (see Keyboard Layout in Reference) contains several macros. Studying these examples will help you see how some of the macro commands interrelate. The examples have numerous comments included, which help you follow the commands of the macro.

{;}comment~ ☺ (⌘)

The text you type between the comment command {;} and the tilde (~) is ignored during macro execution. Comments are useful in helping to quickly recognize what each part of your macro does. You can also use comments to modify (“comment out”) a section of your macro so it will not execute with the macro (see *Commenting Out* under *Notes* at the end of this appendix).

Macros

The comments in the following example show how comments help you to understand what is happening in the macro.

```
{ASSIGN}Phrase1~One~
    {;}Assign "One" to var Phrase1~
{ASSIGN}Phrase2~Two~
    {;}Assign "Two" to var Phrase2~
{ASSIGN}Concatenation~{VARIABLE}Phrase1~ {VARIABLE}Phrase2~
    {;}Assign to var Concatenation the contents of var Phrase1 combined
    with var Phrase2~
{VARIABLE}Concatenation~
    {;}Execute var Concatenation (Type the string "OneTwo")~
```

Note the centered dots (·) between many of the words. Spaces are represented this way in the Macro Editor. If the space were not between Phrase1 and Phrase2, the variable Concatenation would hold "OneTwo."

Merge

The `{COMMENT}` merge command is equivalent to the `{;}` macro command. See *{COMMENT}* below.

`{ASSIGN}var~expr~` ↻ ↘

The `{ASSIGN}` command assigns to the global variable *var* the value returned by *expr*. Do not enclose the expression in quotes. Expressions are evaluated (see *Appendix J: Macros and Merge, Expressions*) and the result is assigned to the variable. Non-numeric characters and expressions that cannot be evaluated are treated as strings.

After a value has been assigned to a variable, the variable command (`{VARIABLE}var~`) can be placed anywhere you would normally place the variable contents. See *Appendix L: Macros and Merge, Variables* for more information on variables.

If you want to empty the variable of its contents, leave the *expr* argument empty (e.g., `{ASSIGN}Fred~`). It is a good idea to empty variables at the beginning of a macro or merge in which they are used (unless the macro or merge assigns new contents to them). When you empty a variable, it no longer “exists” (see *{IF EXISTS}* below).

Macros

In this example, variable Fred is first assigned a value, then an expression, then a string.

```
{ASSIGN}Fred~3~  
    {;}Assign:3-to-var-Fred~  
{ASSIGN}Fred~{VARIABLE}Fred~*2~  
    {;}Multiply-old-value-of-var-Fred-by-2, assign-the-result-(6)-as-the-new  
    value-of-var-Fred~  
{ASSIGN}Fred~WordPerfect~  
    {;}Assign-"WordPerfect"-string-to-var-Fred~
```

See the *Macro* subheading under the following commands for additional examples: `{;}` (comment), `{BREAK}`, `{ELSE}`, `{Item Down}`, `{Item Left}`, `{Item Right}`, `{Item Up}`, `{KTON}`, `{MID}`, `{STATE}` (in text), `{WHILE}`.

Merge

To assign a local variable, see *{LOCAL}* below.

In this example, variable Fred is first assigned a value, then an expression, then a string.

```
{ASSIGN}Fred~3~{COMMENT}  
    Assign 3 to variable Fred  
~{ASSIGN}Fred~{VARIABLE}Fred~*2~{COMMENT}  
    Multiply old value of var Fred by 2, assign the result (6) as the new value of var  
    Fred.  
~{ASSIGN}Fred~WordPerfect~{COMMENT}  
    Assign the string "WordPerfect" to var Fred~
```

See the *Merge* subheading under the following commands for additional examples: {CTON}, {IF}, {LEN}, {MID}, {NEST MACRO}, {PAGE ON}.

{BELL}

The {BELL} command causes your computer to sound a beep. This command is often useful in combination with the {CHAR}, {INPUT}, {KEYBOARD}, {PROMPT}, {STATUS PROMPT}, and {TEXT} commands which prompt the user to enter information. You can also use it to signal arrival at various points of the macro or merge, such as at the end of a lengthy process.

Macros

In this example, a bell will sound and the text "Hello *name*." will be typed at the cursor position. The {BELL} command can also be placed inside the prompts for {CHAR}, {TEXT} and {PROMPT} commands (i.e., {TEXT}|~{BELL}Please enter your name~).

```
{BELL}
    {;}Sound a bell~
{TEXT}Name~Please enter your name:~
    {;}Prompt for a name and place it in var Name~
Hello-{VARIABLE}Name~
    {;}Type the message with the name that was entered~
```

Merge

The following example checks the DATABASE.SF secondary file to see if the name entered by the user matches the Name field in one of the records in the file. If so, it asks for a password. If the name is not found in the secondary file, the user is notified that access is denied because he or she is not in the database.

The first {BELL} command in the example sounds a bell just before the name prompt. Within the IF structure, if the name is not found in the secondary file (the last record in the secondary file has "End" as the Name field), a double bell sounds ({BELL}{BELL}) at the error message "Access denied." If the name is found in the secondary file, another bell sounds at the prompt to enter the password.

```
{BELL}{TEXT}Name~Enter your name: ~{COMMENT}
    Sound a beep and prompt for a name. Place input in var Name
~{NEST SECONDARY}database.sf~{COMMENT}
    Begin using the DATABASE.SF secondary file. The record pointer is positioned at
    the first record in the file.
~{LABEL}CheckName~{COMMENT}
~{IF}"{FIELD}Name"!="{VARIABLE}Name"~{COMMENT}
    If the contents of the Name field and the var Name do not match
~{IF}"{FIELD}Name"!="End"~{COMMENT}
    and if not at the end of the secondary file
~{NEXT RECORD}{COMMENT}
    Move record pointer to the next record
~{GO}CheckName~{COMMENT}
```

```

        Check the next record to find a match
~{ELSE}{COMMENT}
        Otherwise (no match found in file)
~{BELL}{BELL}{PROMPT}Access Denied--You are not entered in the
database.~{WAIT}30~{COMMENT}
~{STOP}{COMMENT}
        End the merge
~{END IF}{COMMENT}
~{ELSE}{COMMENT}
        Otherwise (name found in file)
~{BELL}{TEXT}Password~Enter password: ~{COMMENT}
~{END IF}{STOP}

```

{Block Append} ↻

{Block Append} is a keystroke command and works like the Append feature. It appends a block of text to the end of a file that you designate. It is equivalent to the keystrokes Move (Ctrl-F4), **Block** (1), **Append** (4).

After this command executes, you are prompted for the name of the file to which you wish to append. See *Append* in *Reference* for details.

Macros

Suppose, for example, that you wanted to compile a new document from parts of one or more existing documents. Simply block each section in the order that you want it to appear in the new document, then execute the following macro after each block. Each block is appended to the file listed as *yourfile.wp*.

```

{Block Append}
    {;}Invoke Block Append~
yourfile.wp{Enter}
    {;}Include the name of your append file~

```

The following two macros are equivalent:

```

{Block}{Word Right}{Block Append}
{Block}{Word Right}{Move}ba

```

Merge

This command is not available in Merge.

{Block Copy} ↻

{Block Copy} is a keystroke command and is a shortcut to copying a block in Macros. It is equivalent to the keystrokes Move (Ctrl-F4), **Block** (1), **Copy** (2).

After this command executes, you are prompted to press **Enter** at the point where you want to copy. You can cursor to this position or have your macro move to it (see *Move, Block* in *Reference*).

Macros

For example, if you want to copy a block of text to the end of your document, first block the text, then run the following macro.

```
{Block Copy}
    {;}Invoke-Block-Copy~
{Home}{Home}{Down}
    {;}Move-cursor-to-end-of-document~
{Enter}
    {;}Retrieve-block~
```

The following two macros are equivalent:

```
{Block}{Word Right}{Block Copy}
{Block}{Word Right}{Move}bc
```

Merge

This command is not available in Merge.

{Block Move} ↻

{Block Move} is a keystroke command and is a shortcut to moving a block in *Macros*. It is equivalent to the keystrokes *Move* (Ctrl-F4), *Block* (1), *Move* (1).

After this command executes, you are prompted to press **Enter** at the point where you want to move to. You can cursor to this position or have your macro move to it (see *Move, Block* in *Reference*).

If *Block* is not on when the macro encounters {Block Move}, the command will act like Ctrl-Del and delete the word at the cursor.

Macros

For example, if you want to move a line of text to the beginning of your document, move the cursor to the line you want to move, then run the following macro.

```
{Home}{Home}{Left}
    {;}Move-cursor-to-beginning-of-line~
{Block}
    {;}Turn-on-Block~
{End}{Right}
    {;}Move-cursor-past-the-HRt-or-SRt~
{Block Move}
    {;}Invoke-Block-Move~
{Home}{Home}{Up}
    {;}Move-cursor-to-beginning-of-document~
{Enter}
    {;}Insert-blocked-text~
```

The following two macros are equivalent:

```
{Block}{Word Right}{Block Move}
{Block}{Word Right}{Move}
```

Merge

This command is not available in Merge.

{BREAK} ↻ ↘

{BREAK} is useful when you want to skip some commands when a Cancel, Error, or Not Found condition occurs. Usually it is used to break out of a loop where several nested IF statements are used. The location of the command in the file determines its function.

The rules that govern how this command functions are as follows:

Each of these rules assumes that the previous rules do not hold.

- 1 If a {BREAK} command is encountered within a FOR, FOR EACH, or WHILE structure, execution moves to the end of the structure (just after the {END FOR} or {END WHILE}). If these structures are nested, execution moves after the {END FOR} or {END WHILE} command of the current level.
- 2 If a {BREAK} command is encountered within an IF structure, execution moves to the end of the structure (just after the {END IF}). If the IF structure is nested within a {FOR}, {FOR EACH}, or a {WHILE} structure, the execution moved after the {END FOR} or the {END WHILE}. If the IF structures are nested, execution moves after the {END IF} command of the current IF structure.
- 3 If a {BREAK} command is encountered in a nested file, execution returns to the parent file.
- 4 If none of the above rules hold: In Macros, if a {BREAK} command is encountered, it is ignored. In Merge, if the {BREAK} command is encountered in a primary file, the primary file is ended and execution returns to the next iteration of the current primary file. If the {BREAK} command is encountered in a secondary file, the merge is terminated.

Macros

The following macro parses a full pathname entered by the user into path and filename portions. Notice that the {BREAK} command is within an IF structure, but is also within a FOR structure. The FOR structure takes precedence, so this {BREAK} command sends execution directly after the {END FOR} command (no more iterations of the FOR loop are performed), not after the {END IF} command (remaining iterations of the FOR loop would be performed).

```
{TEXT}String~Enter pathname:~
      {;} Ask user to enter full pathname and assign it to var String~
{ASSIGN}Length~{LEN}String~
      {;} Assign length of full pathname to var Length~
{FOR}Pos~{VARIABLE}Length~-1~-1~
      {;} From last character to first...~
      {ASSIGN}Char~{MID}String~{VARIABLE}Pos~1~
      {;} Assign the character to var Char~
```

```

[IF]"{VARIABLE}Char"="\`
    (;)If the character is a "`
    {ASSIGN}Pathlen`{VARIABLE}Pos`+1`
    (;)Assign the length of the path (not pathname) to var Pathlen`
{ASSIGN}Path`{MID}String`0`{VARIABLE}Pathlen`
    (;)Assign the path portion of var String to var Path`
{ASSIGN}Filelen`{VARIABLE}Length`-{VARIABLE}Pathlen`
    (;)Assign the length of the filename to var Filelen`
{ASSIGN}File`{MID}String`{VARIABLE}Pathlen`{VARIABLE}Filelen`
    (;)Assign the filename portion of var String to var File`
{BREAK}
    (;)Break out of the FOR loop.`
{END IF}
    (;)End of IF structure.`
{END FOR}
    (;)End of FOR structure.·The {BREAK} command sends execution here at the first
    "` found in the pathname`
{CHAR}Foo`Path·:{VARIABLE}path`·Filename:·{VARIABLE}file`...press Enter`
    (;)Display the path and filename portions until user presses a key.`

```

Merge

In the following example, the message “Counting...#” is displayed until it has counted up to 9. Then the message “I am no longer counting.” is displayed.

```

{ASSIGN}Counter`0`{COMMENT}
    Initialize var Counter
`{ASSIGN}Stop`9`{COMMENT}
    Assign the stop value
`{LABEL}Count`{COMMENT}
    `{IF}{VARIABLE}Counter`={VARIABLE}Stop`{COMMENT}
        If var Counter has reached 9
        `{BREAK}{COMMENT}
            Break to the end of the IF statement
    `{ELSE}{COMMENT}
        Otherwise
        `{ASSIGN}Counter`{VARIABLE}Counter`+1`{COMMENT}
            Increment var Counter
        `{PROMPT}Counting...{VARIABLE}Counter`{COMMENT}
            Send a message
        `{WAIT}5`{COMMENT}
            Wait .5 seconds
        `{GO}Count`{COMMENT}
            Repeat the loop
    `{END IF}{COMMENT}
        End of IF statement
    `{PROMPT}I am no longer counting.`{COMMENT}
        Send a message
    `{WAIT}20`{COMMENT}
        Display the message for 2 seconds
    `{QUIT}{COMMENT}
        End the merge`

```


{CALL}label ↻ ↘

The **{CALL}** command transfers execution to the subroutine *label*. When execution of the subroutine is completed (when a **{RETURN}** is encountered in the subroutine), execution returns to the command following **{CALL}**. (See *Subroutines* under *Notes* at the end of this appendix for more information about subroutines. See also *Levels* under *Notes* at the end of this appendix.)

Macros

In the following example, the macro searches for “test”. When it finds it, the macro deletes the entire line on which “test” was found. Then the macro inserts the bolded phrase “This line was deleted.”

```
{LABEL}SearchLoop~
    {;}Begin SearchLoop subroutine~
    {Search}test{Search}
    {;}Search for the word "test"~
    {CALL}Delete~
    {;}Commence Delete subroutine~
    {Bold}This line was deleted.{Bold}{Enter}
    {;}Place the bolded message in the document after line is deleted~
    {GO}SearchLoop~
    {;}Repeat SearchLoop subroutine~

{LABEL}Delete~
    {;}Subroutine Delete~
    {Home}{Home}{Left}
    {;}Move to beginning of line~
    {Del to EOL}
    {;}Delete the line~
{RETURN}
    {;}End of Delete subroutine~
```

Merge

In the following example, the merge checks the Name field in the current record of the secondary merge file. If the name is not blank, the salutation “Dear *name*:{HRt}{HRt}” is written in the merged document. If the name is blank, a generic salutation “Dear Sir or Madam:{HRt}{HRt}” is written.

```
{LABEL}Top~{COMMENT}
~{IF BLANK}Name~{COMMENT}
    If the name field is blank
    ~{CALL}Generic~{COMMENT}
        Call the subroutine "Generic"
    ~{ELSE}{COMMENT}
        Otherwise, write a name-specific salutation using the information in the Name field
        ~Dear {FIELD}Name:
    {COMMENT}
    ~{END IF}{COMMENT}
        End the IF statement
~{RETURN}{COMMENT}
    End of Top routine, return to point where Top was called.
```

```

~{LABEL}Generic~{COMMENT}
  Dear Sir or Madam:
{RETURN}}

```

{CANCEL OFF}

The `{CANCEL OFF}` command is used to stop the Cancel key from performing its normal function. The default condition is for Cancel to be enabled (`{CANCEL ON}`).

Once you have turned Cancel off with this command, you can use Ctrl-Break to cancel the macro or merge during execution.

Macros

In the following example, the macro pauses at the `{INPUT}` command to let the user edit the text of the document. The `{CANCEL OFF}` command lets the user press Cancel (F1) to undelete text while editing.

```

{CANCEL OFF}
{INPUT}Edit the desired text.~Press Enter when done.~
  {;}The {CANCEL OFF} command lets the user press Cancel to
  undelete text without terminating the {INPUT}.~
{CANCEL ON}
  {;}Restores the normal function of Cancel.~

```

`{CANCEL OFF}` is also useful if you want to be able to get a Cancel key as input from the user in the `{CHAR}`, `{LOOK}`, or `{PAUSE}` commands.

Merge

In the following example, the merge pauses at the `{INPUT}` command to let the user edit the text of the document. The `{CANCEL OFF}` command lets the user press Cancel (F1) to undelete text while editing.

```

{CANCEL OFF}{COMMENT}
~{INPUT}Type the text of the memo. Press F9 when done.~{COMMENT}
  The {CANCEL OFF} command lets the user press Cancel to undelete text without
  terminating the {INPUT}.
~{CANCEL ON}{COMMENT}
  Restores the normal function of Cancel.~

```

`{CANCEL OFF}` is also useful if you want to prevent a user from stopping a merge (by pressing Cancel) at a `{CHAR}`, or `{LOOK}` command.

{CANCEL ON}

The `{CANCEL ON}` command is used to enable the Cancel key after it has been disabled with the `{CANCEL OFF}` command (see *{CANCEL OFF}* above).

Macros

See the *Macros* subheading under *{CANCEL OFF}* above for an example of how to use this command.

Merge

See the *Merge* subheading under *{CANCEL OFF}* above for an example of how to use this command.

{CASE}expr~case1~label1~...~caseN~labelN~ ↻ ↘

The {CASE} command allows execution to branch to different locations in the file (designated by *label1*, *label2*, etc.) (see also *Subroutines* under *Notes* at the end of this appendix), depending on what the value returned by *expr* is. The value returned by *expr* is compared to each case. When a match is found, execution branches to the corresponding label. For example, your macro or merge might ask the user to answer Yes or No to a prompt. If the answer is Yes, one function will be performed. If the answer is No, another function will be performed. It is often helpful to format the {CASE} statement (place it on several lines) so it is more readable. Notice that there is an extra tilde (~) required at the end of the {CASE} statement.

The variable contents must match a case exactly. For example, a case of “y” will match “y” but not “Y”. If no match is found in the {CASE} statement, execution continues after the {CASE} statement. You can use an {OTHERWISE} command as the last case in the command to handle all cases that do not match.

It is possible to use the {ELSE} command instead of the {OTHERWISE} command to handle cases that do not match. However, using {ELSE} will produce an error if the {CASE} command is within an IF statement. For this reason, we recommend that you use {OTHERWISE}. The {CASE} command does not require that program execution return after the routine is completed. In other words, it does not *call* the routine rather, it *goes* to the routine. If you want program execution to return, use {CASE CALL} (see *{CASE CALL}* below).

Macros

In this example, the Error subroutine is *not* executed unless a character other than n or y is pressed. If the {CASE} command is changed to {CASE CALL}, the subroutines are *called*, and execution returns to the {QUIT} command.

```
{LABEL}GetChar~
{CHAR}Answer~Continue?(Y/N)-~
      {;}Assign-character-to-var-Answer~
{CASE}{VARIABLE}Answer~
  y~Yes~
  Y~Yes~
  n~No~
  N~No~
  {OTHERWISE}~Error~
~
      {;}If-var-Answer-contains-y-or-Y,go-to-label-Yes;if-var-Answer
      contains-n-or-N,go-to-label-No.~Otherwise,go-to-label-Error.~
{QUIT}
```

See the *Macros* subheading under *{CHAR}* for an additional example.

Merge

In this example, the Error subroutine is *not* executed unless a character other than n or y is pressed. If the *{CASE}* command is changed to *{CASE CALL}*, the subroutines are *called*, and execution returns to the *{STOP}* command.

```
{LABEL}GetChar~{COMMENT}
~{CHAR}Answer~Continue? (Y/N) ~{COMMENT}
    Assign y or n to var Answer
~{CASE}{VARIABLE}Answer~
    y~Yes~
    Y~Yes~
    n~No~
    N~No~
    {OTHERWISE}~Error~
~{COMMENT}
    If var Answer contains y or Y, go to label Yes; if var Answer contains n or N, go to
    label No. Otherwise, go to label Error.
~{STOP}
```

See the *Merge* subheading under *{COMMENT}* for an example of how to use these commands.

{CASE CALL}expr~case1~label1~...caseN~labelN~ ↻ ↘

The *{CASE CALL}* command is similar to the *{CASE}* command in that it can branch to different subroutines. The difference is that *{CASE CALL}* requires that execution return after a subroutine has executed (see *{CASE}* above).

Macros

In the following example, the user is prompted to select an author. The *{CASE CALL}* command allows the user to select the author by number or letter (note the *{^V}*s which turn on and *{^Q}*s which turn off the mnemonic attribute (see *Macros, Message Display in Reference*)). If a correct number or letter is not selected, the *{OTHERWISE}* case executes the routine again. Once the subroutine corresponding to the option selected is executed, execution returns to the *{LABEL}GetType~* command.

```
{LABEL}GetAuthor~
{CHAR}Author~{^V}1~J{^Q}oe;{^V}2~S{^Q}ue;{^V}3~M{^Q}arco~
    {;}Prompt-user-for-author~
{CASE CALL}{VARIABLE}Author~
    1~Joe~
    j~Joe~
    J~Joe~
    {;}If user enters 1, j, or J, call subroutine Joe~
    2~Sue~
    s~Sue~
    S~Sue~
    {;}If user enters 2, s, or S, call subroutine Sue~
    3~Marco~
```

```

m~Marco~
M~Marco~
    {;}If user enters 3, m, or M, call subroutine Marco~
{OTHERWISE}~GetAuthor~
    {;}Otherwise, send prompt again. Notice the tilde after the {OTHERWISE}
    command.~
.
.
.
    {;}Notice the extra tilde to end the CASE statement~
{LABEL}GetType~
.
.
.

```

Merge

In the following example, the user is prompted to select an author. The {CASE CALL} command allows the user to select the author by number. If a correct number is not selected, the {OTHERWISE} case executes the routine again. Once the subroutine corresponding to the option selected is executed, execution returns to the {LABEL}GetType~ command.

```

{LABEL}GetAuthor~{COMMENT}
~{CHAR}Author~] Joe; 2 Sue; 3 Marco: ~{COMMENT}
    Prompt user for author
~{CASE CALL}{VARIABLE}Author~
    1~Joe~
    2~Sue~
    3~Marco~
    {OTHERWISE}~GetAuthor~
~{COMMENT}
    If user enters 1, call subroutine Joe. If user enters 2, call subroutine Sue. If user
    enters 3, call subroutine Marco. Otherwise, send prompt again. Notice the extra
    tilde to end the CASE statement.
~{LABEL}GetType~
.
.
.

```

{CHAIN}macroname ~ ↻ (→)

The {CHAIN} command stores the name of the indicated macro and executes it after the current macro is completed. You can enter a full pathname if you wish (the .WPM extension is not necessary).

You can chain one macro at each level of nesting (see *Levels* under *Notes* at the end of this appendix). If more than one macro is chained at the current nest level, only the last macro chained is executed when the current macro is completed.

See *Chaining, Nesting, and Substituting* under *Notes* at the end of this appendix for more information on chaining.

Macros

The following macro executes a loop until the search text is not found. When the search fails, the loop is exited, completing the current macro. Execution is then transferred to the chained macro (NOTFOUND.WPM).

```
{CHAIN}NotFound~
    {;}Execute the NOTFOUND-macro when this macro is completed~
{ON NOT FOUND}{RETURN}~
    {;}When the search fails, stop this macro, and start the chained macro~
{LABEL}Loop~
    {;}Begin the repeating-subroutine~
    {Search}at{Search}
    {;}Search for "at"~
    {PAUSE}
    {;}Pause (allow user to edit)~
{GO}Loop~
    {;}Search was successful, repeat the search~
```

The section of the macro between the {LABEL} and {GO} commands is repeatedly executed until the search fails. As soon as the search fails, the NOTFOUND macro begins.

Notice the {ON NOT FOUND}{RETURN}~ command. If a macro is not nested and has not been called with {CALL} or {CASE CALL}, {RETURN} ends the macro (see {RETURN} below).

Merge

The {CHAIN} command in Macros is equivalent to the {CHAIN MACRO} command in Merge. See {CHAIN MACRO} below.

{CHAIN MACRO}macroname~ (C) >

The {CHAIN MACRO} merge command starts the named macro at the end of the merge, if the merge terminates normally. You can enter a full pathname for the *macroname* if you wish. You need not enter the .WPM extension.

Only the last macro chained during the merge is executed.

See *Chaining, Nesting, and Substituting* under *Notes* at the end of this appendix for more information on chaining.

In previous versions of WordPerfect, this merge command was represented as ^Gmacroname^G.

Macros

The {CHAIN} command in Macros is equivalent to the {CHAIN MACRO} command in Merge. See {CHAIN} above.

Merge

In the following example, the chained macro PRINT.WPM prints the resulting merge file when the merge is complete.

```
{CHAIN MACRO}print~{COMMENT}
Chain the macro PRINT.WPM when the merge is complete
                                                                    {DATE}
```

```
Dear {FIELD}Name~,
•
• (rest of the letter)
```

```
Sincerely,
```

```
Chris Smith
```

{CHAIN PRIMARY}filename~ ➤

This merge command continues the merge with the named primary file as soon as the current primary file is complete. You can enter a full pathname for the *filename* if you wish.

See *Chaining, Nesting, and Substituting* under *Notes* at the end of this appendix for more information on chaining.

Macros

This command is not available in Macros.

Merge

The {CHAIN PRIMARY}Envelope~ command in the following example executes the ENVELOPE.PF primary file after the last iteration of the current primary file.

```
{CHAIN PRIMARY}Envelope~
•
• (rest of current primary file)
```

{CHAIN SECONDARY}filename~ ➤

This merge command begins using records from the named secondary file when the end of the current secondary file is reached. This command is especially useful if you have broken a large secondary file into several smaller files. If you insert a {CHAIN SECONDARY} command in each smaller secondary file, they will act like one secondary file.

You can use a full pathname for *filename* if you wish.

See *Chaining, Nesting, and Substituting* under *Notes* at the end of this appendix for more information on chaining.

Macros

This command is not available in Macros.

Merge

The following example shows how a large secondary file has been broken into two smaller files. When a merge is executed using the first secondary file, the records of the second file will be used as soon as those in the first file are completed.

Large File:

```
{FIELD NAMES}
Name~
Status~
~{END RECORD}
=====
```

```
Jose{END FIELD}
Staff{END FIELD}
{END RECORD}
=====
```

-
- (additional records)
-

```
=====
Barbara{END FIELD}
Professional{END FIELD}
{END RECORD}
=====
```

```
Julie{END FIELD}
Part-time{END FIELD}
{END RECORD}
=====
```

-
- (additional records)
-

```
=====
Shoji{END FIELD}
Leave of Absence{END FIELD}
{END RECORD}
=====
```

First Small File:

```
{FIELD NAMES}
Name~
Status~
~{COMMENT}
~{PROCESS}{CHAIN SECONDARY}File2~{PROCESS}{COMMENT}
=====
```

The {PROCESS} commands ensure that the {CHAIN SECONDARY} command executes. The merge remembers the {CHAIN SECONDARY} command and filename so that it can execute it when the current secondary file has ended.

```
~{END RECORD}
=====
```



```
Jose{END FIELD}
Staff{END FIELD}
{END RECORD}
=====
```

- (additional records)

```
=====
Barbara{END FIELD}
Professional{END FIELD}
{END RECORD}
=====
```

Second Small File (named FILE2, and in the default directory):

```
{FIELD NAMES}
Name~
Status~
~{END RECORD}
=====
```

```
Julie{END FIELD}
Part-time{END FIELD}
{END RECORD}
=====
```

- (additional records)

```
=====
Shoji{END FIELD}
Leave of Absence{END FIELD}
{END RECORD}
=====
```

{CHAR}var~message~

This command is useful for creating menus and prompts. The {CHAR} command prompts the user with the *message* and waits until a single key is pressed. The key is then assigned to the indicated variable (see *Appendix L: Macros and Merge, Variables*). If Cancel is pressed, the macro or merge ends unless the {CANCEL OFF} or {ON CANCEL} commands have been previously executed. Once the key has been assigned to the variable, a {CASE}, {CASE CALL}, or {IF} command can be used to perform different operations depending on the key pressed.

See *Message Display* under *Notes* at the end of this appendix, and *Macros, Message Display* in *Reference* for information on affecting the way messages are displayed. See also *Prompting and User Input* under *Notes* at the end of this appendix.

After the {CHAR} command executes, the contents of the status line just previous to the execution of the {CHAR} command are restored. Use {PROMPT}~ or {STATUS PROMPT}~ before the {CHAR} command to clear the status line.

Macros

The user can press any key as the input (including a feature key such as Search). In the following example, the user is prompted to select a type of document, after which a subroutine is executed based on what the user entered.

```
[CHAR]DocType~1-Memo;2-Letter;3-Itinerary;~  
        {;}Prompt-the-user-for-input-and-assign-the-character-to-var-DocType~  
{CASE}{VARIABLE}DocType~  
    1~Memo~  
    2~Letter~  
    3~Itin~  
    *~  
    {;}Check var-DocType-and-branch-to-the-appropriate-subroutine~
```

See the *Macros* subheading under the following commands for additional examples: {BREAK}, {CASE}, {CASE CALL}, {KTON}, {RESTART}, {SHELL MACRO}.

Merge

See the *Merge* subheading under the following commands for examples of how to use this command: {CASE}, {CASE CALL}, {CTON}.

{COMMENT}comment~ (C) ~

Use this merge command to put comments in a primary or secondary merge file to make it easier to understand. You can also use this command to format the merge commands so that they are more readable, without inserting extra hard return or tab codes in the document.

Macros

The {COMMENT} command in Merge is equivalent to the {;} (comment) command in Macros (see {;} above).

Merge

The following shows how you can use the {COMMENT} command to format the primary document. This merge creates a list of employees and their salaries. The secondary file has three fields: Name, Type, and Salary. Any record in the secondary file that has 1, 2, or 7 in the Type field is not merged into the list.

```
{LABEL}List~{COMMENT}  
    This comment prevents the insertion of a [HRt] between execution of the {LABEL}  
    command and the {LOCAL} command.  
~{LOCAL}Type~{FIELD}Type~{COMMENT}  
    This comment prevents the insertion of a [HRt] between execution of the {LOCAL}  
    and {CASE CALL} commands.  
~{CASE}{VARIABLE}Type~  
    1~Next~  
    2~Next~  
    7~Next~
```

^(COMMENT)

Notice that there is no need for a {COMMENT} command between the cases. Since these are all part of the same command, no [HRt] will be inserted. The {COMMENT} command after the {CASE} command is necessary to prevent the insertion of [HRt] codes between the execution of the {CASE} command and the {LABEL}WriteIt command.

^(LABEL)WriteIt^(COMMENT)

This comment prevents the hard return between the {LABEL} command and {FIELD} command from appearing in the resulting document.

^(FIELD)Name^

{FIELD}Salary^

{COMMENT}

No comment command was inserted between the {FIELD} commands so that in the document resulting from the merge, the Name and Salary field information will be separated by a [HRt]. Notice the position of this {COMMENT} command. This positioning will put two [HRt] codes between each Name-Salary pair in the resulting list.

^(LABEL)Next^(COMMENT)

^(NEXT RECORD){COMMENT}

^(GO)List^

See the other Merge examples in this section to see more on how to use this command.

{CTON}character^ (C) ↗

The {CTON} merge command (Character **TO** Number) converts *character* (which may be any character in the WordPerfect character sets) to a unique number, its WordPerfect “key value.” {CTON} performs the inverse (opposite) function of the {NTOC} command (see *NTOC* below). You can use the key value to calculate the WordPerfect character set value.

To calculate the character set value for a given character,

- 1 Use the {CTON} command to obtain the key value.
- 2 Divide the key value by 256.

The quotient is the number of the WordPerfect character set (0-12). The remainder is the character number in the character set (0-255).

Macros

This command is not available in Macros. However, the {KTON} macros command is very similar to the {CTON} merge command. See *KTON* below.

Merge

This example calculates the character set value for a character:

{CHAR}Char^Type a character: ^(COMMENT)

Assign input to var Char

```

~{ASSIGN}Set~{CTON}{VARIABLE}Char~/256~{COMMENT}
    Assign quotient to var Set
~{ASSIGN}Num~{CTON}{VARIABLE}Char~/256~{COMMENT}
    Assign remainder to var Num
~{VARIABLE}Set~,~{VARIABLE}Num~{COMMENT}
    Write out character set value

```

{DATE} (C) >

The {DATE} merge command inserts the current date and/or time in the document, as formatted in the Date Format feature (see *Date in Reference*).

In previous versions of WordPerfect, this merge command was represented as ^D.

Macros

Although this command is not available in Macros, you can program a macro to access the Date feature, such as in the following example:

```
This document was printed:~{Date/Outline}c
```

*{Date/Outline} may be inserted in the macro by pressing **Date/Outline** (Shift-F5). When the macro is executed, {Date/Outline}c will select Date Code from the Date/Outline menu.*

For more information, see *Date in Reference*.

Merge

You can use either this command, or the Date Code (Shift-F5,2) (see *Date in Reference*) to insert the current date in merged documents. The first example below uses the {DATE} command; the second example shows how to achieve the same result using the Date Code.

Example 1:

{DATE}

```
{FIELD}Name~
{FIELD}Address~
```

Example 2:

```
October 24, 1991
(FIELD)Name
(FIELD)Address

Doc 1 Pg 1 Ln 1.33 Pos 1.8"
[Tab] [Pg] [Date: 10/24/91] [MRC]
[Mrg: (FIELD)Name: (AB*)]
[Mrg: (FIELD)Address]

Press Reveal Codes to restore screen
```

See the *Merge* subheading under the following commands for additional examples: {CHAIN MACRO}, {KEYBOARD}, {PRINT}, {QUIT}, {REWRITE}.

{DISPLAY OFF} ↻

The {DISPLAY OFF} macro command turns off the display of macro execution. If this command were not present, each action of the macro would be rapidly displayed on the screen as it was executed. In many cases, you may want to turn the display off because macro execution is faster when it does not display.

When you create a macro, the {DISPLAY OFF} command is inserted at the beginning of the macro. You can delete this command if you want the macro to display execution. (Exceptions: If you create a macro at the normal editing screen that ends at a menu or includes a {PAUSE} or {PAUSE KEY} command, the {DISPLAY OFF} command is not inserted.)

If you use this command to turn display off, the screen is not automatically cleared. Whatever is on the screen will remain on the screen (unless another command overwrites it) while the macro is executing.

If display is off when a {PAUSE} or {PAUSE KEY} (see *{PAUSE}* and *{PAUSE KEY}* below) is executed, the screen does not rewrite. The user will probably not know what to do at the pause. You should be sure to turn on display before using one of these commands.

Messages in the {CHAR}, {INPUT}, {PROMPT}, and {TEXT} commands always display on the screen, even when display is off.

Macros

In the following example, display is off until the file is ready to be printed. The {DISPLAY ON} command just before the {Print} command allows the Print menu to display.

```
{DISPLAY OFF}
{Home}{Home}{Up}
    (;)Move-to-beginning-of-file~
{Replace}n|{Search}{Search}
    (;)Replace-"|"-"with-"("w/o-confirm~
{Home}{Home}{Up}
    (;)Move-the-cursor-to-the-top-of-the-file~
{Replace}n|{Search}{Search}
    (;)Replace-"|"-"with-"("w/o-confirm~
{DISPLAY ON}
    (;)Turn-on-display-to-allow-menu-to-display~
{Print}s
    (;)Display-printer-select-menu~
```

Merge

This command is not available in Merge. Merges are always invisible unless you use the {REWRITE} command to rewrite the screen.

{DISPLAY ON} ↻

The {DISPLAY ON} macro command is used to turn on the display of macro execution after it has been turned off by the {DISPLAY OFF} command (see *{DISPLAY OFF}* above). Display On is the default for macro execution. Macro execution is slower when display is on.

{DISPLAY ON} does not itself rewrite the screen. The screen is only rewritten when a command subsequent to the {DISPLAY ON} command performs an action that rewrites the screen.

Macros

See the *Macro* subheading under the following commands for examples of how to use this command: {DISPLAY OFF}, {MENU OFF}.

Merge

This command is not available in Merge. Merges are always invisible unless you use the {REWRITE} command to rewrite the screen.

{DOCUMENT}filename~ (C) ↵

The {DOCUMENT} merge command inserts the named document into the merged document at the point the command is encountered. The document is not processed, so any merge commands in the inserted document are ignored. This command is commonly used for inserting variable paragraphs in contracts and similar documents, often called *document assembly*, or to have the merge build a primary or secondary file.

Macros

This command is not available in Macros. However, you can perform a similar function with the following macro:

```
{Retrieve}document.wp{Enter}
```

These commands retrieve DOCUMENT.WP into the current document.

Merge

In the following example, the {DOCUMENT} command inserts the document called ONEROOM.MRG in the merged document if variable Rooms is equal to 1, or the document TWOROOMS.MRG if variable Rooms is not equal to 1.

```
{IF}{VARIABLE}Rooms=1~{COMMENT}  
~{DOCUMENT}OneRoom.mrg~{COMMENT}  
~{ELSE}{DOCUMENT}TwoRooms.mrg~{COMMENT}  
~{END IF}
```

{ELSE}

The {ELSE} command is used in connection with the {IF}, {IF BLANK}, {IF EXISTS}, {IF NOT BLANK}, and {END IF} commands. It marks the beginning of the commands which execute should the IF value be zero (false) (see *IF* below).

The {ELSE} command is not a required part of an IF statement. It should be used when there are certain steps that need to be performed *only* when the IF value is zero (false). The commands below {END IF} execute whether or not the IF value is true.

It is possible to use the {ELSE} command instead of the {OTHERWISE} command to handle cases that do not match in a {CASE} or {CASE CALL} command. However, {ELSE} will not function correctly if the CASE statement is within an IF statement.

Macros

In the example below, if variable Number contains a negative number, add 2. If not, subtract 2.

```
{IF}{VARIABLE}Number<0~  
  {;}If-var-Number-is-less-than 0~  
  {ASSIGN}Number~{VARIABLE}Number+2~  
  {;}Add-2-to-var-Number~  
{ELSE}  
  {;}If not~  
  {ASSIGN}Number~{VARIABLE}Number-2~  
  {;}subtract-2-from-var-Number~  
{END IF}  
  {;}End-of-IF-statement~
```

See the *Macros* subheading under the following commands for additional examples: {IF EXISTS}, {Item Down}, {Item Left}, {Item Right}, {Item Up}, {RESTART}, {SHELL MACRO}, {STATE}.

Merge

See the *Merge* subheading under the following commands for examples of how to use this command: {DOCUMENT}, {IF}, {IF BLANK}, {IF EXISTS}, {LOOK}, {NEST MACRO}, {NEXT RECORD}, {SUBST PRIMARY}.

{END FIELD} ↘

The {END FIELD} merge command signals the end of a field in a secondary file. A Hard Return [HRT] is automatically inserted with this command for better readability.

*When you insert this command, a message appears at the bottom of the screen that lets you know the number of the field where the cursor is currently located. If you use the {FIELD NAMES} command, the name of the field appears instead of the number. Pressing **Home.Home.Up Arrow** will temporarily remove the message from the screen. However, as long as you have an {END FIELD}, {END RECORD}, or {FIELD NAMES} command in the file, the message will display when the cursor is after the command. If you delete all the {END FIELD}, {END RECORD}, and {FIELD NAMES} commands from the file, then press **Home.Home.Up Arrow**, the message will not reappear.*

In previous versions of WordPerfect, this merge command was represented by ^R.

Macros

This command is not available in Macros.

Merge

Records in the following secondary file have three fields:

```
International Exporting{END FIELD}
(801) 555-4421{END FIELD}
George Wiley{END FIELD}
{END RECORD}
```

```
=====
Tradewinds, Inc.{END FIELD}
(409) 555-3567{END FIELD}
Susan Escher{END FIELD}
{END RECORD}
=====
```

See the *Merge* subheading under the following commands for additional examples: {CHAIN SECONDARY}, {FIELD NAMES}.

{END FOR} ↻ ↘

This command signals the end of a {FOR} or {FOR EACH} loop (see {FOR} and {FOR EACH} below).

Macros

See the *Macros* subheading under the following commands for examples of how to use this command: {BREAK}, {FOR}, {FOR EACH}, {NEXT}.

Merge

See the *Merge* subheading under the following commands for examples of how to use this command: {FOR}, {MID}, {NEXT}.

{END IF} ⌂ ➤

The {END IF} command marks the end of an IF structure and is used with the {IF}, {IF BLANK}, {IF EXISTS}, {IF NOT BLANK}, and {ELSE} commands.

Macros

See the *Macros* subheading under the following commands for examples of how to use this command: {BREAK}, {ELSE}, {GO}, {IF}, {IF EXISTS}, {Item Down}, {Item Left}, {Item Right}, {Item Up}, {LEN}, {LOOK}, {NEXT}, {ORIGINAL KEY}, {RESTART}, {SHELL MACRO}, {STATE} (in text).

Merge

See the *Merge* subheading under the following commands for examples of how to use this command: {BELL}, {CALL}, {DOCUMENT}, {GO}, {IF}, {IF BLANK}, {IF EXISTS}, {IF NOT BLANK}, {MID}, {NEST MACRO}, {NEST PRIMARY}, {NEST SECONDARY}, {NEXT}, {NEXT RECORD}, {SUBST PRIMARY}, {SUBST SECONDARY}.

{END RECORD} ➤

This merge command signals the end of a record in a secondary file. A hard page break is inserted automatically with this code for better readability.

*When you insert this command, a message appears at the bottom of the screen that lets you know the number of the field where the cursor is currently located. If you use the {FIELD NAMES} command, the name of the field appears instead of the number. Pressing **Home.Home.Up Arrow** will temporarily remove the message from the screen. However, as long as you have an {END FIELD}, {END RECORD}, or {FIELD NAMES} command in the file, the message will display when the cursor is after the command. If you delete all the {END FIELD}, {END RECORD}, and {FIELD NAMES} commands from the file, then press **Home.Home.Up Arrow**, the message will not reappear.*

In previous versions of WordPerfect, this merge command was represented as ^E.

Macros

This command is not available in Macros.

Merge

See the *Merge* subheading under the following commands for examples of how to use this command: {CHAIN SECONDARY}, {END FIELD}, {FIELD NAMES}.

{END WHILE} ⌂ ➤

This command signals the end of a {WHILE} loop (see {WHILE} below).

Macros

See the *Macros* subheading under the following commands for examples of how to use this command: {SYSTEM}, {WHILE}.

Merge

See the *Merge* subheading under {SYSTEM} for an example of how to use this command.

{FIELD}field~ ➤

The {FIELD} merge command inserts the contents of the named (or numbered) field in the merged document or in another merge command.

In previous versions of WordPerfect, this merge command was represented as ^Fname^ or ^F#^.

Macros

This command is not available in Macros.

Merge

In the following example, the contents of the Name and Address fields in the current record in the secondary merge file are inserted in the letter.

```
{FIELD}Name~  
{FIELD}Address~  
  
Dear {FIELD}Name~:  
.  
.  
.
```

See the *Merge* subheading under the following commands for additional examples: {BELL}, {CALL}, {CHAIN MACRO}, {COMMENT}, {DATE}, {GO}, {IF BLANK}, {IF NOT BLANK}, {MID}, {MRG CMND}, {NEST MACRO}, {NEST PRIMARY}, {NEST SECONDARY}, {NEXT RECORD}, {PAGE OFF}, {PAGE ON}, {PRINT}, {REWRITE}, {SUBST PRIMARY}.

{FIELD NAMES}name1~... nameN~~ ➤

This merge command declares the names and order of the fields in a secondary file. Using this command in the secondary file allows you to reference the fields by name in a primary file. (Referencing fields by name in a primary file is usually much easier than remembering the order of the fields in the secondary file.)

The {FIELD NAMES} command must be the first command in the secondary file. If the records have more fields than those declared in this command, additional fields are numbered. The {FIELD NAMES} command must precede the records. When initially inserted, the command and field names are all on one line.

You may find it easier to read by formatting it on separate lines, as in the example below. An {END RECORD} command and its accompanying [HPg]

code are inserted automatically when you insert this command. You are limited to 100 named fields, and field names are limited to 39 characters.

After you insert this command, a message appears at the bottom of the screen that lets you know the name or number of the field where the cursor is currently located. Pressing **Home.Home.Up Arrow** will temporarily remove the message from the screen. However, as long as you have an {END FIELD}, {END RECORD}, or {FIELD NAMES} command in the file, the message will display when the cursor is after the command. If you delete all the {END FIELD}, {END RECORD}, and {FIELD NAMES} commands from the file, then press **Home.Home.Up Arrow**, the message will not reappear.

Macros

This command is not available in Macros.

Merge

In the following example, the first three fields of each record are named using the {FIELD NAMES} command. The additional fields are not named, and so are numbered Field 4, Field 5, etc. The {FIELD NAMES} command shown in this example has been formatted on several lines so that it is easier to read.

```
{FIELD NAMES}
Company~
Phone~
Contact~
~{END RECORD}
=====
International Exporting{END FIELD}
(801) 555-4421{END FIELD}
George Wiley{END FIELD}
Furniture{END FIELD}
45{END FIELD}
$1.5 million{END FIELD}
{END RECORD}
=====
Tradewinds, Inc.{END FIELD}
(409) 555-3567{END FIELD}
Susan Escher{END FIELD}
Memorabilia{END FIELD}
200{END FIELD}
$2.0 million{END FIELD}
{END RECORD}
=====
```

See the *Merge* subheading under {CHAIN SECONDARY} for an additional example.

{FOR}var~start~stop~step~  

The {FOR} command is useful for executing a series of commands a certain number of times. The commands between the {FOR} and {END FOR} commands are executed once for each value of *var* between *start* and *stop* inclusive, as incremented by *step*.

The start, stop, and step values can be entered in the command as expressions, variables, or other commands. Each expression, variable, or command is evaluated to a value, then the value is assigned to the variable.

In any FOR loop, {END FOR} command must be used to determine the end of the series of commands included in the loop. The {END FOR} command sends execution to the top of loop for the next iteration. You can also use the {NEXT} command to send execution to the next iteration (see *NEXT* below), but you must still include an {END FOR} to mark the end of the loop.

The {FOR} command itself initializes the variable with the start value. You do not need to pre-assign the variable. Each subsequent time the {FOR} command is executed (at the top of the loop), the variable is incremented by the *step* value.

Macros

The following example shows how you could use the {FOR} command to write out a line of 20 asterisks (*). (See *Merge* below for a more complicated example.)

```
{FOR}Counter~1~20~1~
    {;}For every value of Counter from 1 to 20 (values incremented
    by 1 each time through the loop)~
    *
    {;}Write an asterisk:(*)~
{END FOR}
    {;}End of {FOR} loop. Repeat the loop unless Counter=20~
```

See the *Macros* subheading under the following commands for additional examples: {BREAK}, {NEXT}.

Merge

The following example is similar to the example under *Macros* above; however, in this example the start value has been changed to 4, and the step to 3. Counter is initialized to 4 the first time through the loop, and so the asterisk will be written out. In subsequent iterations of the loop, only 5 more asterisks will be written out (Counter equals 7, 10, 13, 16, 19). After that the loop will end because 22 (the next increment after 19) is greater than 20 (the stop value).

```
{FOR}Counter~4~20~3~{COMMENT}
    For every value of Counter from 4 to 20 (values incremented by 3 each time
    through the loop)
    ~*{COMMENT}
    Write an asterisk (*)
~{END FOR}{COMMENT}
    End of {FOR} loop. Repeat the loop unless Counter>=20~
```

See the *Merge* subheading under the following commands for additional examples: {MID}, {NEXT}.

{FOR EACH}var~expr1~...~exprN~ ↻

The {FOR EACH} macro command is similar to the {FOR} command. The difference is that instead of having a sequential step value, each value to be assigned to the variable is included as an argument in the command. (The values can still be included as expressions, variables, or commands. They are evaluated before being assigned to the variable.) See *{FOR}* above for more information.

Like the {FOR} command, the {FOR EACH} command itself initializes the variable with the start value. You do not need to pre-assign the variable. Each time the {FOR EACH} command is executed (at the top of the loop), the variable is assigned the next value.

Remember to end the loop with an {END FOR} command.

Macros

In the following example, the macro will loop 5 times. Each time it loops, the variable "Count" will be equal to the specified value (i.e., on the first loop, Count=15, on the second loop, Count=10, on the third, Count=25, etc.).

```
{FOR EACH}Count~15~10~25~95~50~  
  {;}Var-Count-is-initialized-to-15-the-first-time-through-the-loop,then  
  10,then-25,and-so-on~  
  {VARIABLE}Count~  
  {;}Write-out-var-Count-followed-by-a-space~  
{END FOR}  
  {;}Perform-the-next-iteration-of-the-loop-unless-Count=50.-In-that-case,end-the-loop.~
```

When execution is complete, "15 10 25 95 50 " will have been written out. Note the space (represented by "~") after the {VARIABLE}Count~ command. This is the space after each number when they are written out.

Merge

This command is not available in Merge.

{GO}label~ ↻ ➤

The {GO} command transfers execution to the location in the macro or merge file indicated by *label*. It is used in conjunction with the {LABEL} command which marks the place to which execution is transferred.

The {GO} command is useful when you want to skip a part of your macro or merge, or to transfer control to another part based on a condition. Unlike the {CALL} command, {GO} does not require that execution return.

Macros

This macro checks to see if the Search command has been entered in variable Key. Notice that the steps between {END IF} and {LABEL} will be skipped if variable Key contains {Search}.

```
{IF}"{VARIABLE}Key~"="{Search}""  
  {;}If-var-Key-contains-{Search}~  
  {GO}Search~
```

```

        {;}Go-to-the-Search-Label~
{END IF}
    •
    • (middle section of macro)
    •
{LABEL}Search~
    {;}Perform the search~
    •
    • (Steps of the search)
    •
{QUIT}
    {;}Stop-macro-execution~

```

The macro ends after the search has been completed.

See the *Macros* subheading under the following commands for additional examples: {CHAIN}, {Item Down}, {Item Left}, {Item Right}, {Item Up}, {LEN}, {LOOK}, {ON CANCEL}, {ON ERROR}, {RETURN CANCEL}, {RETURN ERROR}.

Merge

In the following example, suppose you have a secondary file where the last record has “End” as the contents of its Name field. You could use the following commands to stop the merge when it gets to that record.

```

{IF}"{FIELD}Name"="End"{COMMENT}
    ~{GO}EndTheMerge~{COMMENT}
~{END IF}
    •
    • (more of the primary file)
    •
{LABEL}EndTheMerge~{COMMENT}
    ~{STOP}

```

See the *Merge* subheading under the following commands for additional examples: {BELL}, {COMMENT}, {IF}, {LOOK}, {ON CANCEL}, {ON ERROR}, {RETURN CANCEL}, {RETURN ERROR}.

{IF}*expr* ~ ↻ ➤

The {IF} command is used to execute a set of commands only *if* a certain condition exists. If the condition exists (*expr* is evaluated to be true), the commands directly after the {IF} command are executed.

The *expr* argument is usually a logical expression. An expression is true if it is evaluated as a non-zero number. For example, when the expression $4=4$ is evaluated, the result is -1 (which corresponds to true). The value is false if it results in a 0 or contains nothing at all. String values and commands must be enclosed in quotes to be evaluated correctly. See *Appendix J: Macros and Merge, Expressions* for more information on expression evaluation.

If the value is true, the commands directly after the {IF} execute. If the value is false (or there is no value at all), the commands after {IF} are skipped, and execution continues after the {END IF} command.

An {IF} statement always begins with {IF} and ends with {END IF}. If you want certain commands to execute only when the value is not true, use the {ELSE} command (see *{ELSE}* above).

It is possible to nest {IF} commands. See *{STATE}* below for an example of nested {IF} statements.

Macros

In the example below, the Setup menu is displayed only if the appropriate password is in variable Input.

```
{IF}"{VARIABLE}Input"="{VARIABLE>Password"
    {;}If-var Input contains the password-(as-stored-in-var Password)
    {Setup}
    {;}Enter the-Setup-menu
{END IF}
    {;}End-of-{IF}-structure
```

Notice that quotes are placed around both {VARIABLE}Input and {VARIABLE>Password because it is a string comparison.

See the *Macros* subheading under the following commands for additional examples: {BREAK}, {ELSE}, {GO}, {Item Down}, {Item Left}, {Item Right}, {Item Up}, {LEN}, {LOOK}, {NEXT}, {ORIGINAL KEY}, {RESTART}, {SHELL MACRO}, {STATE} (in text).

Merge

The following example writes "again " each time through the loop until variable Counter equals 0. Then it writes "Finished." (Final output is "again again Finished.") Notice that the {IF} statement does not use a logical expression. Rather, it checks to see if the IF value is non-zero.

```
{ASSIGN}Counter`3`{COMMENT}
    Set var Counter to 3 (loop will execute 3 times before IF value is false (zero))
`{LABEL}Loop`{COMMENT}
    `{IF}{VARIABLE}Counter`{COMMENT}
        If var Counter is non-zero
        `again {COMMENT}
            Write "again "
        `{ASSIGN}Counter`{VARIABLE}Counter`-1`{COMMENT}
            Subtract 1 from var Counter
        `{GO}Loop`{COMMENT}
            Repeat the loop
    `{ELSE}{COMMENT}
        `Finished.{COMMENT}
            Write "Finished."
    `{END IF}{COMMENT}
        End of {IF} statement
```

See the *Merge* subheading under the following commands for additional examples: {BELL}, {DOCUMENT}, {GO}, {MID}, {NEST PRIMARY}, {NEST SECONDARY}, {NEXT}, {NEXT RECORD}, {SUBST PRIMARY}.

{IF BLANK}field~ ➤

If the indicated field is blank, the commands after this command are executed. Be sure to end the set of commands to be executed with an {END IF} command. See also *{IF NOT BLANK}* below, and *If Blank, If Not Blank, and ?* under *Notes* at the end of this appendix.

Macros

This command is not available in Macros.

Merge

In the following example, the secondary merge file records have a Title field and a Name field. WordPerfect checks to see if the Title field of the current record is blank. If so, it inserts just the Name in the merged document. Otherwise, it inserts the Title before the Name.

```
{IF BLANK}Title~{COMMENT}
    If the Title field is blank in this record
    ~{FIELD}Name~ {COMMENT}
        Write out just the Name followed by a space
~{ELSE}{COMMENT}
    Otherwise
    ~{FIELD}Title~ {FIELD}Name~{COMMENT}
        Write the Title before the Name
~{END IF}{COMMENT}
    End of {IF BLANK} statement~
```

See the *Merge* subheading under the following commands for additional examples: {CALL}, {NEST MACRO}, {SUBST SECONDARY}.

{IF EXISTS}var~ ↻ ➤

The {IF EXISTS} command checks to see if the indicated variable has been assigned. If it has, the commands following {IF EXISTS} are executed. Like other IF statements, the {IF EXISTS} command requires an {END IF} and can use an {ELSE} (see *{IF}* above).

Common uses for the {IF EXISTS} command include establishing a default response at a menu (see Macro example below). {IF EXISTS} is also useful when you want execution to wait until a key is pressed (see *{LOOK}* below).

Macros

The following example shows how you can use the {IF EXISTS} command to set up a default response to a user prompt:

```
{TEXT}Author~1 Joe;2 Sue;3 Marco~1{Left}~
    {}User selects author~
{IF EXISTS}Author~
{ELSE}
```



```

        {;}If user pressed Enter at menu
    {ASSIGN}Author~1~
        {;}Use default of "1"
{END IF}

```

Merge

The following example shows how you can use the {IF EXISTS} command to set up a default response to a user prompt:

```

{TEXT}Author~1 Joe; 2 Sue; 3 Marco: ~{COMMENT}
    User selects author
~{IF EXISTS}Author~{COMMENT}
~{ELSE}{COMMENT}
    If user pressed Enter at menu
    ~{ASSIGN}Author~1~{COMMENT}
        Use default of "1"
~{END IF}

```

{IF NOT BLANK}field ~ ➤

If the indicated field is not blank, the commands following this merge command are executed. Be sure to end the set of commands to be executed with an {END IF} command. See also *{IF BLANK}* above.

Macros

This command is not available in Macros.

Merge

In the following example, the secondary merge file records have a Title field and a Name field. WordPerfect checks to see if the Title field of the current record is blank. If not, it inserts the Title before the Name. Otherwise, it inserts just the Name.

```

{IF NOT BLANK}Title~{COMMENT}
    If the Title field is not blank in this record
    ~{FIELD}Title~ {COMMENT}
        Write out the title and a space
~{END IF}{COMMENT}
    End of {IF NOT BLANK} statement
~{FIELD}Name~

```

{INPUT}message ~ ↻ ➤

This command prompts the user with the message, then pauses, allowing the user to perform any keystroke operations. Once input is terminated (see *Macros* and *Merge* subheadings below), the message is removed from the screen and execution continues.

Using an {INPUT} command after a {STATUS PROMPT} command will remove the previous {STATUS PROMPT} message (see {STATUS PROMPT} below).

See also *Prompting and User Input* under *Notes* at the end of this appendix for alternative means of obtaining user input.

While an executing merge is paused at an {INPUT} (or {KEYBOARD}) command, you can execute the {QUIT}, {NEXT RECORD}, or {STOP} commands from the keyboard. See *Inserting Merge Commands During Execution* under *Notes* at the end of this appendix.

In previous versions of WordPerfect, this merge command was represented as ^Omessage^O^C.

Macros

After the command below is executed, the user can do any editing. Execution continues when the user presses **Enter**.

```
{INPUT}Edit the codes. Press Enter when done.
```

See the *Macros* subheading under the following commands for additional examples: {CANCEL OFF}, {MENU OFF}, {Para Down}, {Para Up}.

Merge

After the command below is executed, the user can do any editing. Execution continues when the user presses **End Field** (F9).

```
{INPUT}Type the memo text. Press F9 when done.
```

See the *Merge* subheading under *{CANCEL OFF}* for an additional example.

{Item Down} ↵

The {Item Down} keystroke command is used in tables and paragraph numbering to move down one section or cell.

Macros

The following example moves down each cell in a column of a table and inserts the cell number in it.

```
{ASSIGN}PreviousCell~0~  
{LABEL}Number-Cells~  
  {IF} "{VARIABLE}PreviousCell"!="{SYSTEM}Cell"  
    {SYSTEM}Cell~  
      {;}Write-current-cell-number~  
    {ASSIGN}PreviousCell~{SYSTEM}Cell~  
      {;}Assign-current-cell-number-to-var-PreviousCell~  
    {Item Down}  
      {;}Move-down-to-the-next-cell~  
    {GO}Number-Cells~  
      {;}Repeat-the-loop~  
  {ELSE}  
    {QUIT}  
  {END IF}  
    {;}End-of-IF-statement~  
{RETURN}
```

Merge

This command is not available in Merge.

{Item Left} ⇐

The {Item Left} keystroke command is used in tables and paragraph numbering to move left one section, column, or cell.

Macros

The following example moves right to left to each cell in a table and inserts the cell number in it.

```
{ASSIGN}PreviousCell^0^
{LABEL}Number-Cells^
  {IF}"{VARIABLE}PreviousCell^"!="{SYSTEM}Cell^"
  {SYSTEM}Cell^
    {;}Write-current-cell-number^
  {ASSIGN}PreviousCell^{SYSTEM}Cell^
    {;}Assign-current-cell-number-to-var-PreviousCell^
  {Item Left}
    {;}Move-left-to-the-previous-cell^
  {GO}Number-Cells^
    {;}Repeat-the-loop^
  {ELSE}
    {QUIT}
  {END IF}
    {;}End-of-IF-statement^
{RETURN}
```

Merge

This command is not available in Merge.

{Item Right} ⇨

The {Item Right} keystroke command is used in tables and paragraph numbering to move right one section, column, or cell.

Macros

The following example moves left to right to each cell in a table and inserts the cell number in it.

```
{ASSIGN}PreviousCell^0^
{LABEL}Number-Cells^
  {IF}"{VARIABLE}PreviousCell^"!="{SYSTEM}Cell^"
  {SYSTEM}Cell^
    {;}Write-current-cell-number^
  {ASSIGN}PreviousCell^{SYSTEM}Cell^
    {;}Assign-current-cell-number-to-var-PreviousCell^
  {Item Right}
    {;}Move-right-to-the-next-cell^
  {GO}Number-Cells^
    {;}Repeat-the-loop^
  {ELSE}
    {QUIT}
```

```

{END IF}
    {;}End-of-IF-statement~
{RETURN}

```

Merge

This command is not available in Merge.

{Item Up} ↻

The {Item Up} keystroke command is used in tables and paragraph numbering to move up one section or cell.

Macros

The following example moves up each cell in a column of a table and inserts the cell number in it.

```

{ASSIGN}PreviousCell^0~
{LABEL}Number.Cells~
  {IF}"{VARIABLE}PreviousCell"!="{SYSTEM}Cell~"~
    {SYSTEM}Cell~
      {;}Write-current-cell-number~
      {ASSIGN}PreviousCell^{SYSTEM}Cell~
      {;}Assign-current-cell-number-to-var-PreviousCell~
      {Item Up}
      {;}Move-up-to-the-next-cell~
      {GO}Number.Cells~
      {;}Repeat-the-loop~
  {ELSE}
    {QUIT}
  {END IF}
    {;}End-of-IF-statement~
{RETURN}

```

Merge

This command is not available in Merge.

{KEYBOARD} (⌘) ➔

The {KEYBOARD} merge command pauses an executing merge to rewrite the screen and then let the user enter information from the keyboard. This command is similar to the {PAUSE} and {PAUSE KEY} commands in Macros. When **End Field** (F9) is pressed, the merge continues. See also *Prompting and User Input* under *Notes* at the end of this appendix for additional methods of obtaining user input.

While an executing merge is paused at a {KEYBOARD} (or {INPUT}) command, you can execute the {QUIT}, {NEXT RECORD}, or {STOP} commands from the keyboard. See *Inserting Merge Commands During Execution* under *Notes* at the end of this appendix.

In previous versions of WordPerfect, this merge command was represented as ^C.

Macros

This command is not available in Macros. Use {PAUSE}, {PAUSE KEY}, or {INPUT} instead.

Merge

In the following example, the {KEYBOARD} commands pause to let the user enter the "From:," "To:," and "Subject:" text as the merge is executed. The screen is rewritten at each {KEYBOARD} command which displays the portion of the file merged so far, so that the user knows what information to enter.

MEMO

From: {KEYBOARD}

To: {KEYBOARD}

Date: {DATE}

Subject: {KEYBOARD}

See the *Merge* subheading under {STATUS PROMPT} for an example of how to use this command.

{KTON}key~ ↻ (↵)

The {KTON} macro command (**Key TO Number**) converts *key* (which may be any key on the keyboard) to a unique number, its WordPerfect "key value." {KTON} performs the inverse (opposite) function of the {NTOK} command (see {NTOK} below).

If you take the {KTON} of a function key, an editing key, or a cursor key, the key value is the end result. This is the value you would use with the {NTOK} command.

If *key* is a character (characters are a subset of all keys), you can use the key value to calculate the WordPerfect character set value. The character set value can be assigned to a variable, and a variable can be used to input the character set value.

To calculate the character set value for a given key,

- 1 Use the {KTON} command to obtain the key value.
- 2 Divide the key value by 256.

The quotient is the number of the WordPerfect character set (0-12). The remainder is the character number in the character set (0-255). For additional values returned by the {KTON} command, see *Appendix T: Macros and Merge, Value Tables*.

Macros

For example, if you want your macro to calculate the character set value for a character, you could use the following:

```
{CHAR}Key~Type any key~  
  {;}Assign input to var-Key~  
{ASSIGN}Set~{KTON}{VARIABLE}Key~/256~  
  {;}Assign quotient to var-Set~  
{ASSIGN}Num~{KTON}{VARIABLE}Key~/256~  
  {;}Assign remainder to var-Num~  
{VARIABLE}Set~{VARIABLE}Num~  
  {;}Write out character set value~
```

This macro prompts for a key, then calculates and writes out the character set value for that key.

Merge

This command is not available in Merge; however, the {CTON} merge command is very similar to it (see {CTON} above).

{LABEL}label~ ↻ ➤

A {LABEL} command marks a place in the macro or merge file. Execution can be sent directly there from any place in the macro or merge file. The {CALL}, {CASE}, {CASE CALL}, and {GO} commands are used to direct execution to the label. The {LABEL} command can also be used to mark the beginning of a subroutine (see *Subroutines* under *Notes* at the end of this appendix).

The label name distinguishes each label from the others. Label names have no restrictions regarding length; however, only the first 15 characters are used to determine uniqueness. (In other words, ABCDEFGHIJKLMNO (15 characters) and ABCDEFGHIJKLMNOP (16 characters) would be considered by WordPerfect to be the same label name, but ABCDEFGHIJKLMN (14 characters) and ABCDEFGHIJKLMNO (15 characters) would be considered by WordPerfect to be different label names. The only character you cannot use in the label name is a tilde (~) because a tilde ends the label name. You *can* use spaces.

There can be many labels in the same macro or merge file as long as each one has a different name. If the name is duplicated, the first one is used. The other is ignored.

Macros

See the *Macros* subheading under the following commands for examples of how to use this command: {CALL}, {CASE}, {CASE CALL}, {CHAIN}, {GO}, {Item Down}, {Item Left}, {Item Right}, {Item Up}, {LEN}, {LOOK}, {ON CANCEL}, {ON ERROR}.

Merge

You may not use a label and local variable (see *Appendix L: Macros and Merge, Variables*) of the same name. If you do, when you execute the

merge, you will receive an error message "Label is already defined."
Rename either the label or local variable.

See the *Merge* subheading under the following commands for examples of how to use this command: {BELL}, {CALL}, {CASE CALL}, {COMMENT}, {GO}, {IF}, {LOOK}, {ON CANCEL}, {ON ERROR}.

{LEN}var or **{LEN}expr** ↻ ↘

This command determines the length of a variable or length of the value returned by an expression (see *Macros* and *Merge* subheadings below). This command is useful for validating or restricting the length of user input.

Macros

In *Macros*, you can only use a variable name for the *var* argument, not an expression.

In this example, if the user enters more than two letters at the {TEXT} prompt, the macro rejects it and the user is prompted again.

```
{LABEL}Get:State~  
  {TEXT}State~Enter a two-letter state abbreviation:~  
  {IF}{LEN}State~>2~  
    {GO}Get State~  
  {END IF}
```

See the *Macros* subheading under *{BREAK}* for an additional example.

Merge

In *Merge*, you can use text, {FIELD}, {VARIABLE}, or complex expressions in argument *expr*.

In the following example, variable NameLength is assigned 7, which is the number of characters in the name "Sharron".

```
{ASSIGN}Name~Sharron~  
{ASSIGN}NameLength~{LEN}{VARIABLE}Name~
```

See the *Merge* subheading under *{MID}* for an additional example.

{LOCAL}var expr ↻ ↘

The {LOCAL} merge command assigns the value returned by *expr* to the local variable *var*. Local variables are known only to the current file, and are deleted when the merge is finished or the file is exited. See also *{ASSIGN}* above, and *Appendix L: Macros and Merge, Variables* for more information.

You may not use a label and local variable of the same name. If you do, when you execute the merge, you will receive an error message "Label is already defined." Rename either the label or local variable.

Macros

This command is not available in Macros.

Merge

See the *Merge* subheading under *{COMMENT}* for an example of how to use this command.

{LOOK}var

The **{LOOK}** command checks to see if a key has been pressed by the user. If a key has been pressed, it is assigned to the variable; it is not executed. If a key has not been pressed, the contents of the variable are deleted and execution is continued without stopping.

Macros

In the following example, **{LOOK}** is used to simulate a Pause which does not terminate with the Enter key (see *PAUSE* below). (You could also do this with the **{PAUSE KEY}** command.)

```
{STATUS PROMPT}Press Exit to Quit.~
    {;}Exit terminates the pause~
{LABEL}Loop~
    {;}Top-of-the-loop~
    {LOOK}Key~
        {;}Check to see if a key was pressed~
    {IF}~{VARIABLE}Key~={Exit}~
        {;}If Exit was pressed~
        {GO}Next~
            {;}drop-out-of-the-loop~
    {END IF}
        {;}End-of-{IF}-structure~
    {VARIABLE}Key~
        {;}Perform the keystroke~
    {GO}Loop~
        {;}Go-to the top-of the loop~
```

See the *Macros* subheading under *{ORIGINAL KEY}* for an additional example.

Merge

The following merge sounds a beep until the user presses "s." Then the message "You did it!" is written out.

```
{PROMPT}Press "s" to stop the beep~{COMMENT}
    Send the message to the user
~{LABEL}Top~{COMMENT}
    ~{BELL}{LOOK}Key~{COMMENT}
        Sound the bell and check the last key pressed
    ~{IF}~{VARIABLE}Key~="s"~{COMMENT}
        If the user pressed "s"...
        ~{GO}End~{COMMENT}
            Break out of the loop
    ~{ELSE}~{COMMENT}
```



```

        Otherwise
        ^{GO}Top^{COMMENT}
        Repeat the loop
        ^{END IF}{COMMENT}

    ^{LABEL}End^{COMMENT}
    ^You did it!{QUIT}

```

{MENU OFF} ↻

Use the {MENU OFF} macro command to turn off display of menus (except pull-down menus).

When you define a macro from the normal editing screen using the mouse to access pull-down menus, WordPerfect automatically inserts a {MENU OFF} command before, and a {MENU ON} command after, the keystroke command that displays the menu (e.g., {MENU OFF}{Font}{MENU ON}). Normally, in macro execution, the intervening menus to the final option chosen on the pull-down menu would display as regular menus. The {MENU OFF} command prevents the display of these intervening menus.

If execution terminates while menus are off, WordPerfect will automatically turn them on again.

Macros

The macro in the following example displays a directory in List Files, allows the user to mark the files, then copies the files to the diskette in drive A.

```

(TEXT)Directory^Enter the directory^
    (;)Prompt-user-to-enter-directory^
(DISPLAY ON)
    (;)Turn-display-on-so-List-Files-will-display.^
(MENU OFF)
    (;)Turn-off-menus-so-that-the-menu-at-the-bottom-of-the-List-Files
    screen-will-not-display^
(List){VARIABLE}Directory^(Enter)
    (;)List-the-files-in-the-directory-entered-by-the-user^
(INPUT)Mark-files.-Press-Enter-when-done.^
    (;)After-sending-an-instructional-message,-pause-for-the-user-to-mark-the-files.^
cya:{Enter}
    (;)Copy-marked-files-to-A:^
(MENU ON)
    (;)Turn-menus-back-on-so-future-menus-will-display.^

```

Merge

This command is not available in Merge.

{MENU ON} ↻

The {MENU ON} macro command is used to turn on the display of menus after a {MENU OFF} command has been executed (see {MENU OFF} above).

Macros

See the *Macros* subheading under *{MENU OFF}* for an example of how to use this command.

Merge

This command is not available in Merge.

{MID}var⁻offset⁻count⁻ or {MID}expr⁻offset⁻count⁻ ↻ ↘

Use this command to extract substrings. The substring returned is the set of characters of the string resulting from the evaluation of *expr* (see *Merge* subheading below) or the string in *var* (see *Macros* subheading below), starting at the *offset* character and continuing *count* characters. This command is often used with *{LEN}* to parse non-integer numbers.

Macros

In Macros, you can only use a variable name for the *var* argument.

In the following example, the *{MID}* command converts the adverb “quickly” to the adjective “quick” by extracting the first five characters of the adverb.

```
{ASSIGN}Adverb~quickly~  
      {}Assign-string-"quickly"-to var Adverb~  
{MID}Adverb~0~5~  
      {}Extract-the-first-5-characters-from-the string in var Adverb and write them out~
```

For a more complex example, see the *Macros* subheading under *{BREAK}*.

Merge

In Merge, you can use text, *{FIELD}*, *{VARIABLE}*, or complex expressions in argument *expr*.

In this example, the secondary file has a *Name* field where the names are stored in the format Last, First. The subroutine below uses the *{MID}* command to extract the first name from the field. (For a simple example of *{MID}*, see the example under *Macros* above.)

```
{ASSIGN}Name~{FIELD}Name~{COMMENT}  
      Transfer the contents of the Name field to var Name  
~{FOR}Counter~0~{LEN}{VARIABLE}Name~1~{COMMENT}  
      For each character of var Name (from the 0th character to the last character)...  
  ~{IF}~{MID}{VARIABLE}Name~{VARIABLE}Counter~1~=","~{COMMENT}  
      ...Check to see if the current character is a comma (.).  
  ~{GO}End~{COMMENT}  
      If it is a comma, stop checking characters (break out of the loop)  
  ~{END IF}{COMMENT}  
      End of {IF} statement  
~{END FOR}{COMMENT}  
      End of {FOR} statement  
~{LABEL}End~{COMMENT}  
~{ASSIGN}ThisChar~{VARIABLE}Counter~+2~{COMMENT}  
      Assign to var ThisChar the position in the string of the first character of the first  
      name
```

```

^ {ASSIGN}FirstName^ {MID} {VARIABLE}Name^ {VARIABLE}ThisChar^ {LEN} {VARIABLE}
E}Name^ {COMMENT}

```

Assign to var FirstName the substring of characters in var Name starting at the first character of the first name ({VARIABLE}Counter) and continuing {LEN}{VARIABLE}Name characters (to the end of the name)

```

^ {VARIABLE}FirstName^ {COMMENT}

```

Write out the first name^

{MRG CMND}codes {MRG CMND} >-

This merge command lets you insert text, codes, and commands in the document being created by the merge. Any text, codes, or merge commands between the {MRG CMND} commands are sent directly to the merged document, without the commands being interpreted or executed.

In previous versions of WordPerfect, this merge command was represented as ^Vcodes^V.

Macros

This command is not available in Macros.

Merge

In the following example, the merge command {FIELD}Name^ is inserted in the resulting merged document.

```

{MRG CMND} {FIELD}Name^ {MRG CMND} {COMMENT}

```

Insert {FIELD}Name(tilde) in the resulting document^

{NEST}macroname^ ↻ (↵)

The {NEST} macro command transfers control to another macro. When the nested macro has finished, execution returns to the parent macro. It is somewhat like placing the contents of the specified macro where the {NEST} command is. See *Chaining, Nesting, and Substituting* under *Notes* at the end of this appendix for more information on nesting.

Macros

For example, suppose you often create macros that use cursor positioning to display large menus and messages on the screen (see *Macros, Message Display* in *Reference*). The messages will not display correctly if Reveal Codes is on. So, in each macro that uses cursor positioning, you need to first check whether Reveal Codes is on, and if so, turn it off. If you have already defined those keystrokes as a separate macro (say, CODESOFF.WPM), you can nest that macro in each macro that uses cursor positioning rather than re-entering the necessary commands. The macro that nests the CODESOFF.WPM macro would look something like this:

```

{NEST}CodesOff^

```

```

[;]Nest the CODESOFF.WPM macro to be sure Reveal Codes is
turned off before any messages are sent^

```

-
- (commands of the macro)
-

```
{PROMPT}{^P}{^A}{Up}The macro is altering your file. Please Wait
(;)Send the message, positioning it at position 1,23. This should
display correctly because the CODESOFF.WPM macro turned off Reveal Codes."
```

- (rest of the macro)

See the *Macros* subheading under *{RESTART}* for an additional example.

Merge

The `{NEST MACRO}` merge command is equivalent to the `{NEST}` macro command. See *{NEST MACRO}* below.

{NEST MACRO}macroname~ (C) }->

The named macro is executed when this merge command is encountered. When the macro is finished, the merge is continued with the code following the `{NEST MACRO}` command.

See also *Chaining, Nesting, and Substituting* under *Notes* at the end of this appendix for more information on nesting.

Macros

This command is not available in *Macros*. However, its function is identical to the `{NEST}` macro command. See *{NEST}* above for more information.

Merge

In the following example, the secondary file records each have an *Address*, *City*, *State*, and *ZipCode* field. During the merge, if a record is encountered with a blank *ZipCode* field, the merge nests a macro *ZIPCODE.WPM* that calculates the ZIP Code based on the address and stores it in a global variable named *ZipCode*.

```
•
• (first part of primary file)
•
{IF BLANK}ZipCode~{COMMENT}
    If the ZipCode field is blank
    ~{ASSIGN}Address~{FIELD}Address~{COMMENT}
    ~{ASSIGN}City~{FIELD}City~{COMMENT}
    ~{ASSIGN}State~{FIELD}State~{COMMENT}
        Assign the contents of the Address, City, and State fields to global variables that the
        nested macro can access
    ~{NEST MACRO}zipcode~{COMMENT}
        This macro calculates the zip code and stores it in the global var ZipCode
    ~{VARIABLE}ZipCode~{COMMENT}
        Write out the calculated ZIP code
~{ELSE}{COMMENT}
    If the ZipCode field is not blank
    ~{FIELD}ZipCode~{COMMENT}
        Write out the contents of the ZIPCode field
~{END IF}{COMMENT}
```

-
- (rest of primary file)
-

{NEST PRIMARY}filename~ ↘

This merge command is similar to {NEST MACRO}. When this command is encountered, control of the merge is turned over to the named primary file. When the commands in the nested file have been executed, control is returned to the original primary file, where execution is resumed after the {NEST PRIMARY} command. To change the secondary file, use the {NEST SECONDARY} command (see *{NEST SECONDARY}* below).

You can nest primary files up to 10 deep. However, using this command without specifying a filename ({NEST PRIMARY}~) is the same as using the {SUBST PRIMARY} command with the current primary filename as the argument ({SUBST PRIMARY}CurrentPrimaryFilename~) (see *{SUBST PRIMARY}* below). In this case, the {NEST PRIMARY} command does not use one of the 10 nest levels. See *Levels* under *Notes* at the end of this appendix for more information.

In previous versions of WordPerfect, this merge command was represented as ^Pfilename^P. Using {NEST PRIMARY}~ (without a filename) is equivalent to ^P^P in previous versions, which is equivalent to the {SUBST PRIMARY}CurrentPrimaryFile~ command.

Macro

This command is not available in Macros. Use the {NEST} command to nest a macro, or use the macro to begin a merge. For example,

```
{Merge/Sort}m
    {;}Begin-a-merge~
letter.pf{Enter}
    {;}Primary file=LETTER.PF~
address.sf{Enter}
    {;}Secondary file=ADDRESS.SF~
```

will begin a merge. You can then use the {NEST PRIMARY} command in the LETTER.PF file.

Merge

Be aware that a sort cannot be performed in a nested macro from a merge.

In the following example, the primary file is a letter to customers. It checks to see whether there is a balance due, and if so, it nests a primary file that creates an invoice section in the letter, using records of transactions in the nested secondary file TRANSACT.SF.

```
{FIELD}Name~.
```

It has been a pleasure serving you this year.

```
{IF}{FIELD}Balance<0{COMMENT}
  {NEST SECONDARY}Transact.sf{COMMENT}
  {NEST PRIMARY}Invoice.pf{COMMENT}
{END IF}
```

Sincerely,

See the *Merge* subheading under the following commands for additional examples: {ON ERROR}, {PAGE ON}.

{NEST SECONDARY}filename ➤

The named secondary file is opened and used for the subsequent merge commands. Use of the parent secondary file can only be continued when the merge on the nested secondary file has been completed. (You can, however, re-nest the original secondary file from the primary file, which will open a second copy of the secondary file and begin at the first record.) You can nest secondary files up to 10 deep (see *Levels* under *Notes* at the end of this appendix).

If you nest a secondary file from a secondary file, the first record in the nested secondary file effectively replaces the record containing the {NEST SECONDARY} command in the original secondary file and execution continues in the nested file. When execution returns to the parent secondary file, the record pointer skips to the next record *after* the one containing the {NEST SECONDARY} command. (For more information on the record pointer, see *Record Pointer* under *Notes* at the end of this section.) Therefore, you can only nest one secondary file per record in the parent secondary file. In addition, any field text following the {NEST SECONDARY} command is ignored.

Using this command without specifying a filename ({NEST SECONDARY}) returns an error.

Macros

This command is not available in Macros.

Merge

In this example, the original secondary file has a dummy record at the end of the file with "End" in the name field. The commands shown here nest the secondary file ADDRESS2.SF when the last record of the secondary file is reached.

```
{IF}"{FIELD}Name"="End"{COMMENT}
  {NEST SECONDARY}Address2.sf{COMMENT}
{END IF}
```

See the *Merge* subheading under the following commands for additional examples: {BELL}, {NEST PRIMARY}.

{NEXT} ↻ ➤

Use this command to execute the next iteration of a {FOR}, {FOR EACH}, or {WHILE} loop. Usually, the {END FOR} or {END WHILE} command that ends the loop sends execution to the next iteration. However, the {NEXT}

command can be used to send execution to the next iteration from *other* than the end of the loop. For example, you may use nested IF statements as part of the loop, where when a certain condition is true, you want to abandon the rest of the commands in the loop and go to the next iteration. In this case, you would use the {NEXT} command at the point where you want the next iteration to begin.

Even if you use the {NEXT} command in a loop, you must still use an {END FOR} or {END WHILE} command to mark the end of the loop.

Macros

The following macro writes out "*****@ @ @ @ @".

```
{FOR}1^1^15^1^
    {;}Repeat the loop 15 times, starting at 1, ending at 15, in increments
    of 1^
    {IF}{VARIABLE}1^>10^
        {;}If var 1 is > 10,^
        @
        {;}Write an @^
    {NEXT}
        {;}Skip to the next iteration of the loop^
    {END IF}
    *
    {;}Write an asterisk^
{END FOR}
```

Merge

The following merge writes out "*****@ @ @ @ @".

```
{FOR}Fred^1^15^1^ {COMMENT}
    Repeat the loop 15 times, starting at 1, ending at 15, in increments of 1
    ^{IF}{VARIABLE}Fred^>10^ {COMMENT}
        If var Fred is > 10,
        *@ {COMMENT}
        Write an @
    ^{NEXT} {COMMENT}
        Skip to the next iteration of the loop
    ^{END IF} {COMMENT}
    * {COMMENT}
        Write an asterisk
    ^{END FOR}
```

{NEXT RECORD} ➤

The {NEXT RECORD} merge command moves the record pointer in the secondary file to the next record (see *Record Pointer* under *Notes* at the end of this appendix). If it does not find the next record, it ends the merge, or returns the merge to the next command in the primary file if the secondary file was nested, and un-nests the secondary file.

While an executing merge is paused at a {KEYBOARD} or {INPUT} command, you can execute the {NEXT RECORD} command from the keyboard. See

Inserting Merge Commands During Execution under *Notes* at the end of this appendix.

In previous versions of WordPerfect, this command was represented by ^N.

Macros

This command is not available in Macros.

Merge

In this example, the secondary file is searched until the variable `CompanyName` matches the field `Company` in the secondary file. When a match is found, the contents of the `Amount` field of that record in the secondary file are written to the merged document.

```
{LABEL}DoCompany^{COMMENT}
^{IF}"{VARIABLE}CompanyName"="{FIELD}Company""{COMMENT}
  ^{NEXT RECORD}{COMMENT}
  ^{GO}DoCompany^{COMMENT}
^{ELSE}{COMMENT}
  ^{FIELD}Amount^
{END IF}
```

See the *Merge* subheading under the following commands for additional examples: `{BELL}`, `{COMMENT}`, `{SUBST PRIMARY}`.

{NTOC}number¹ (C) ↗

The `{NTOC}` merge command (**N**umber **T**O **C**haracter) converts a WordPerfect key value or character set number to its character equivalent. It performs the inverse (opposite) function of the `{CTON}` command (see *{CTON}* above). For example, the `{NTOC}` of 294 is "Ç".

If you want to calculate the key value for a given character set value,

- 1 Multiply the character set number by 256, then add the number of the character.

For example, "Ç" is character number 38 in character set 1. Multiply 256 times 1 (256), then add 38 (294). The `{NTOC}` of 294 is "Ç".

`{NTOC}` will return nothing if it is taken of a number that is not equivalent to a character.

Macros

This command is not available in Macros. However, the `{NTOK}` macro command is very similar to the `{NTOC}` merge command. See *{NTOK}* below for more information.

Merge

The following example prompts for a number, then returns the character equivalent.

```
{TEXT}Num>Type a key value number: ^{COMMENT}
^{NTOC}{VARIABLE}Num^
```


{NTOK}number ↻ (→)

The {NTOK} macro command (**Number TO Key**) converts a WordPerfect key value to its character or function equivalent. It performs the inverse (opposite) function of the {KTON} command (see {KTON} above). For example, if you take the {NTOK} of 32809 (Save), a Save is executed. If you take the {NTOK} of 1537, a “±” is written out.

You can also take the {NTOK} of a WordPerfect character set value. For example, the character set value for “±” is 6,1. Type **{NTOK}6,1** to obtain “±”.

If you want to calculate the key value for a given character set value,

- 1 Multiply the character set number by 256, then add the number of the character.

You can then use the {NTOK} command to obtain the character.

For example, “±” is character number 1 in character set 6. Multiply 256 times 6 (1536), then add 1 (1537). You can then take the {NTOK} of 1537 to obtain “±”. (For additional values see *Appendix T: Macros and Merge, Value Tables*.)

Macros

The following example prompts for a number, then returns the character or function equivalent.

```
{TEXT}Num~Type a number:~  
{NTOK}{VARIABLE}Num~
```

If the number returns a function, such as Save, and you do not want the function to execute, you can store it in a variable (e.g.,
{ASSIGN}Key~{NTOK}{VARIABLE}Num~~).

Merge

This command is not available in Merge; however, the {NTOC} merge command is very similar to it. See {NTOC} above.

{ON CANCEL}action ↻ (→)

The {ON CANCEL} command tells WordPerfect what to do if a user presses **Cancel** (F1) or if a {RETURN CANCEL} command has been returned by a subroutine or nested macro or merge. When a Cancel occurs, WordPerfect will know what to do next only if it has already encountered the {ON CANCEL} command. For this reason, it is a good idea to place the command before a Cancel can occur, otherwise execution will terminate when Cancel is pressed.

The valid *actions* available with this command in Macros are:

```
{BREAK}           {RETURN}  
{CALL}            {RETURN CANCEL}  
{GO}              {RETURN ERROR}  
{QUIT}           {RETURN NOT FOUND}  
{RESTART}
```

The valid *actions* available with this command in Merge are:

| | |
|---------|-----------------|
| {BREAK} | {RETURN} |
| {CALL} | {RETURN CANCEL} |
| {GO} | {RETURN ERROR} |
| {QUIT} | {STOP} |

Since {GO} and {CALL} require a tilde (~) after the label, there must be two tilde marks (~) at the end (e.g., {ON CANCEL}{GO}label~).

In Macros, the default response to a Cancel (if no {ON CANCEL} command is encountered) is {RETURN CANCEL}. In Merge, the default response to a Cancel (if no {ON CANCEL} command is encountered) is {STOP}.

If no *action* is specified in the command (i.e., {ON CANCEL}~), the cancel is ignored and execution continues as if there had been no cancel. In Macros, not only is the cancel condition ignored, but if the Cancel key was pressed, the key is thrown away. In other words, if either a {LOOK} or {ORIGINAL KEY} is used, they do not detect that the Cancel key was pressed. If you want the Cancel key to be used as input, use the {CANCEL OFF} command before the input is requested.

Macros

When Cancel is pressed (or a {RETURN CANCEL} is encountered), WordPerfect executes the last {ON CANCEL} command encountered at the current level (see *Levels* under *Notes* at the end of this appendix). If no {ON CANCEL} command was encountered during execution of the current level, WordPerfect looks to successively higher levels and executes the last one that was encountered. If none was encountered, the default ({RETURN CANCEL}) is executed.

If you chain or nest a macro, the {ON CANCEL} command is not passed from the parent file to the nested or chained file. Rather, the default ({RETURN CANCEL}) is in effect until another {ON CANCEL} command is encountered.

When execution returns from a lower level to a higher level, the last {ON CANCEL} command encountered at the higher level resumes effect.

In the following example, if the user presses **Cancel** (F1) during the macro, the subroutine End is executed.

```
{ON CANCEL}{GO}End~
•
•   (commands in macro)
•
{LABEL}End~
    {;}Beginning-of-End-subroutine~
    {PROMPT}Macro-cancelled prematurely.~
    {;}Send-notification-message~
    {WAIT}40~
    {;}Display-message-for-4-seconds~
```

```
{Screen}{Screen}
      {;}Clear message from screen
{QUIT}
      {;}Terminate execution
•
• (rest of macro)
•
```

See the *Macros* subheading under *{RETURN CANCEL}* for an additional example.

Merge

In Merge, when Cancel is pressed, the last `{ON CANCEL}` command encountered in the file is executed. If none was encountered, the default (`{ON CANCEL}{STOP}`) is executed.

The `{ON CANCEL}` command is local to the file in which it is encountered. It cannot be seen from other files. For example, a secondary file cannot use the `{ON CANCEL}` command from a primary file.

In the following example, if the user presses **Cancel** (F1) during the merge, the subroutine End is executed.

```
{ON CANCEL}{GO}End~{COMMENT}
      If Cancel (F1) is pressed during the merge, execute the End subroutine~
•
• (commands in primary file)
•
{LABEL}End~{COMMENT}
      Beginning of End subroutine
      ~{PROMPT}Merge cancelled prematurely.~{COMMENT}
      Send notification message
      ~{WAIT}40~{COMMENT}
      Display message for 4 seconds
      ~{STOP}{COMMENT}
      Terminate execution~
```

See the *Merge* subheading under *{RETURN CANCEL}* for an additional example.

{ON ERROR}*action*

The `{ON ERROR}` command tells WordPerfect what to do if an error is detected in macro or merge execution, or returned by WordPerfect or DOS, or if a `{RETURN ERROR}` command has been returned by a subroutine or nested macro or merge. Inserting this command without an action (`{ON ERROR}`) will cause WordPerfect to ignore the error and continue execution (when possible).

In Macros, any error that returns an error message to or from WordPerfect can be trapped with this command. In Merge, the errors that can be trapped with this command are:

- File not found
- Print queue errors
- End of file condition returned by a {NEXT RECORD}

If you chain a file that does not exist or is not found, the error condition is not generated until WordPerfect tries to execute the chained file (i.e., at the end of the current macro or merge file). See *Chaining, Nesting, and Substituting* under *Notes* at the end of this appendix, and the descriptions for the {CHAIN}, {CHAIN MACRO}, {CHAIN PRIMARY}, and {CHAIN SECONDARY} commands in this section for more information.

In Macros, the default *action* (if no {ON ERROR} is encountered) is {RETURN ERROR}. In Merge, the default *action* is {STOP}. For a list of other possible actions, see {ON CANCEL} above.

Macros

The range of effect of the {ON ERROR} command is the same as the {ON CANCEL} command (see the *Macros* subheading under {ON CANCEL} above).

In the following example, the macro requests that the user enter the name of a file. The macro then tries to retrieve it. The {ON ERROR} command specifies that the Error subroutine be executed if the file is not found when the macro tries to retrieve it.

```
{ON ERROR}{GO}Error~
      {;}If an error is generated, execute the Error subroutine~
{LABEL}GetFile~
  {TEXT}Filename~File to be retrieved::~~
      {;}Prompt user for file~
  {Retrieve}{VARIABLE}Filename~{Enter}
      {;}Retrieve the file~
  *
  * (more commands)
  *
{LABEL}Error~
  {;}If the file was not found when the macro tried to retrieve it,
  execution moves here~
  {Cancel}
      {;}Cancel-"Document to be retrieved:" prompt~
  {PROMPT}The file you entered is not in the default directory..Try again.~
      {;}Tell the user what happened~
  {WAIT}15~
      {;}Display the message for 1.5 seconds~
  {GO}GetFile~
      {;}Prompt again for the file~
```

See the *Macros* subheading under {RETURN ERROR} for an additional example.

Merge

The range of effect of the {ON ERROR} command is the same as the {ON CANCEL} command.

In the following example, a new primary file is nested. The {ON ERROR} command specifies that the Error subroutine be executed if the file is not found when the merge tries to nest it.

```
{ON ERROR}{GO}Error~{COMMENT}
    If an error is generated, execute the Error subroutine~
    *
    * (more merge commands)
    *
{NEST PRIMARY}invoice.pf~
    *
    * (more merge commands)
    *
{LABEL}Error~{COMMENT}
    If the file was not found when the merge tried to nest it, execution moves here
    ~{PROMPT}File not found. Move INVOICE.PF to default directory and start merge
    again.~{COMMENT}
    Send a message to the user
    ~{WAIT}15~{COMMENT}
    Display the message for 1.5 seconds
    ~{STOP}{COMMENT}
    Terminate execution~
```

See the *Merge* subheading under *{RETURN ERROR}* for an additional example.

{ON NOT FOUND}action~ ↻

The {ON NOT FOUND} macro command tells WordPerfect what to do if a search fails (e.g., Search, Word Search, or Name Search) or a {RETURN NOT FOUND} is returned by a nested macro or subroutine. If no {ON NOT FOUND} command is included before a Not Found condition occurs, the Not Found stops that level of macro execution (an {ON NOT FOUND} {RETURN NOT FOUND}~ is executed).

For a list of valid actions for this command, see *Macros* under *{ON CANCEL}* above. The range of effect of the {ON NOT FOUND} command is the same as the {ON CANCEL} command.

If you search for a nonexistent name with the Name Search feature, the Not Found condition is returned at the first character that does not match. You should insert an {Enter} command somewhere in the macro after the Not Found is generated to terminate the name search.

If during a name search all characters before the {Enter} match, a Not Found is *not* generated, even though there may be additional characters in the name of the file at the cursor. To check whether the file is an exact match, use {HPg} (Ctrl-Enter) to terminate the Name Search instead of {Enter}. When you use {HPg}, a Not Found is generated if the filename does not exactly match.

Macros

See the *Macros* subheading under the following commands for examples of how to use this command: {CHAIN}, {RETURN NOT FOUND}.

Merge

This command is not available in Merge.

{ORIGINAL KEY} ↻

The {ORIGINAL KEY} macro command evaluates the original (unmapped) action of the last key entered from the keyboard. The last key pressed could be either a key that was read before the macro started (which may be the key which invoked the macro) or a character input with a {CHAR}, {LOOK}, {TEXT}, {PAUSE}, or {PAUSE KEY} command.

Macros

This command is useful if your keyboard has been remapped with a keyboard definition (see *Keyboard Layout* in *Reference*). For example, you might want your macro to check if a user has typed a specific key, regardless of the keyboard definition. If you want to exit if F7 has been pressed, use the following macro:

```
{LOOK}Key`
      {;}Check-to-see-if-a-key-has-been-pressed..Assign-it-to-var-Key`
{IF}"{ORIGINAL KEY}"="{Exit}"`
      {;}If-the-unmapped-key-is-Exit`
      {Exit}
      {;}Exit`
{END IF}
      {;}End-of-{IF}-statement`
```

Merge

This command is not available in Merge.

{OTHERWISE}

Use this command as the last case in a {CASE} or {CASE CALL} command for cases other than the specified cases.

{PAGE OFF} ➤

The {PAGE OFF} merge command eliminates the hard page [HPg] between copies of the primary file in the merged document. Use {PAGE ON} to begin insertion of hard page codes again after you have used {PAGE OFF}. The {PAGE ON} and {PAGE OFF} commands are global to the merge; they may be included in any primary or secondary file and are in effect for all the files in the merge until the opposite command is encountered.

These commands are useful for merging labels or for including multiple records in a document. You can also use {PAGE OFF} with the {PRINT} command to eliminate blank pages between copies of the merged document when merging to the printer. (In previous versions of WordPerfect, this was accomplished with the ^N^P^P codes.)

The {PAGE OFF} command must be separated from the previous text by a hard return in order for the merge to perform a line feed before continuing the merge.

Macros

This command is not available in Macros.

Merge

In the following example, the records in the secondary file have 2 fields: Name and Salary. When merged with the primary file below, a list is created of each name and salary. Because Page is off, no hard page break is inserted between each iteration of the primary file, thereby creating a single list.

```
{PAGE OFF}
{FIELD}Name~ . . . . . {FIELD}Salary~
```

See the Merge example under the following commands for additional examples: {PAGE ON}, {PRINT}.

{PAGE ON} ➤

The {PAGE ON} merge command reinstates the use of hard page codes between copies of the primary file in the merged document. See {PAGE OFF} for more information.

Macros

This command is not available in Macros.

Merge

In the example below, the nested primary file LIST.PF creates a list of players on a given team—one team per page.

```
{ASSIGN}Team~{FIELD}Team~{COMMENT}
    Assign the current field to a global variable so that the nested primary file can use it
~These people are on your team (Team {FIELD}Team~:

{PAGE OFF}{COMMENT}
    This command prevents the insertion of a hard page between iterations of the nested
    primary file
~{NEST PRIMARY}List.pf~{COMMENT}
    This primary file uses another secondary file to produce the list. (It writes the
    Name field of all records whose Team field matches the current Team variable.)
~{PAGE ON}{COMMENT}
    This command restores the use of page breaks to allow one team list per page~
```

{Para Down} ↻

The {Para Down} keystroke command moves the cursor to the beginning of the next paragraph (just beyond the next [HRt]).

Macros

The following macro swaps two paragraphs.

```
{INPUT}Position-cursor-on-paragraph-to-be-moved-down, then press Enter.~
      {;}Prompt-user-and-pause-for-user-to-position-cursor~
{Move}pm
      {;}Move-paragraph-into-buffer~
{Para Down}
      {;}Position-cursor-down-one-paragraph~
{Home}{Home}{Left}
      {;}Position-cursor-at-beginning-of-the-line~
{Enter}
      {;}Retrieve-the-paragraph~
```

Merge

This command is not available in Merge.

{Para Up} ↻

The {Para Up} keystroke command moves the cursor to the beginning of the current paragraph (to the right of the previous [HRt]) or, if the cursor is already at the beginning, to the beginning of the previous paragraph.

Macros

The following macro swaps two paragraphs.

```
{INPUT}Position-cursor-on-paragraph-to-be-moved-up, then press Enter.~
      {;}Prompt-user-and-pause-for-user-to-position-cursor~
{Move}pm
      {;}Move-paragraph-into-buffer~
{Para Up}
      {;}Position-cursor-up-one-paragraph~
{Home}{Home}{Left}
      {;}Position-cursor-at-beginning-of-the-line~
{Enter}
      {;}Retrieve-the-paragraph~
```

Merge

This command is not available in Merge.

{PAUSE} ↻ (⏸)

The {PAUSE} macro command causes the macro to pause until **Enter** is pressed. This command lets the user edit or type new text as if there were no macro running. Macro execution proceeds after Enter is pressed. (If you want another key to end the pause, see *{PAUSE KEY}* below.)

{PAUSE} does not prompt the user. Because of this, the {PROMPT} and/or {BELL} commands are often used with {PAUSE}. See also *Prompting and User Input* under *Notes* at the end of this appendix for other methods of obtaining user input.

Macros

In the following example, after the {PROMPT} command is executed, the user can do any editing. Execution continues when the user presses Enter.

```
{STATUS PROMPT}Edit the codes.-Press Enter when done.~  
    {;}Send a prompt to the screen~  
{PAUSE}  
    {;}Pause for user to edit codes~  
{STATUS PROMPT}~  
    {;}Clear status prompt message
```

See the *Macros* subheading under {CHAIN} and {RESART} for additional examples.

Merge

This command is not available in Merge. However, the {KEYBOARD} merge command is very similar to the {PAUSE} macro command. See {KEYBOARD} above.

{PAUSE KEY}key~ ↻ (↘)

This command functions like the {PAUSE} command (see {PAUSE} above), except that you specify the key that terminates the pause.

Macros

If you wanted Exit (F7) to terminate the pause, you could use the following:

```
{STATUS PROMPT}Edit the codes.- Press Exit when done.~  
    {;}Send message to user.~  
{PAUSE KEY}{Exit}~  
    {;}Stop so that user can edit codes.-Execution continues when the user presses Exit  
    (F7)~  
{STATUS PROMPT}~  
    {;}Clear prompt.~
```

Merge

This command is not available in Merge. However, you may be able to use the {KEYBOARD} merge command instead. See {KEYBOARD} above.

{PRINT} ↘

The {PRINT} merge command sends all text that has been merged so far to the printer. Once the text is sent to the printer, it is cleared from the edit buffer (i.e., it is no longer in the “resulting document”). When you merge to the printer using this command, the usual page break is still inserted between each iteration of the primary file. To eliminate the extra page between each copy, insert the {PAGE OFF} command before the {PRINT} command (see the example under *Merge* below).

The {PRINT} command is ignored if encountered in a substructure during a merge.

In previous versions of WordPerfect, this command was represented as ^T.

Macros

This command is not available in Macros.

Merge

The secondary file to be used with the primary file in the following example contains 1000 records. If you were to merge it to the screen, the resulting document would have 1000 pages. So, this merge uses the {PRINT} command to send each letter to the printer as soon as it is merged.

ABC Company
245 West Center Street
Long Beach, California 90807

{DATE}

{FIELD}Name~
{FIELD}Company~

Dear {FIELD}Salutation~:

Thank you for your inquiry regarding our new product.

•
•
•

Sincerely,

Amy Wilcox
Product Manager{COMMENT}

~{PAGE OFF}{PRINT}

See the *Merge* subheading under {SYSTEM} for an additional example.

{PROCESS}codes{PROCESS} ➤

This merge command is designed to be used in a secondary merge file. The text, codes, or commands enclosed in the {PROCESS} commands are executed when they are encountered, regardless of the current location in the secondary file. For example, if this command is encountered while the merge is scanning the secondary file for a record, the codes are processed, even if they are not in the record being searched for.

To prevent the codes from being executed, you can use a {GO} command to send control of the merge to another part of the file, thus skipping over the {PROCESS} command.

Macros

This command is not available in Macros.

Merge

See the *Merge* subheading under *{CHAIN SECONDARY}* for an example of how to use this command.

{PROMPT}message

The **{PROMPT}** command displays the *message* on the status line. See *Message Display* under *Notes* at the end of this appendix for information on affecting the way messages are displayed on the screen. See also *Prompting and User Input* under *Notes* at the end of this appendix for other methods of prompting the user.

In previous versions of WordPerfect, this merge command was represented as ^Omessage^O.

Macros

See the *Macros* subheading under the following commands for examples of how to use this command: **{NEST}**, **{ON CANCEL}**, **{ON ERROR}**, **{SYSTEM}**, **{WHILE}**.

Merge

See the *Merge* subheading under the following commands for examples of how to use this command: **{BELL}**, **{ON CANCEL}**, **{ON ERROR}**, **{SYSTEM}**.

{QUIT}

The **{QUIT}** command stops the execution of the macro or merge. If macros are nested or chained, it stops their execution at that point.

While an executing merge is paused at a **{KEYBOARD}** or **{INPUT}** command, you can execute the **{QUIT}** command from the keyboard. See *Inserting Merge Commands During Execution* under *Notes* at the end of this appendix.

In previous versions of WordPerfect, this merge command was represented as ^Q.

Macros

See the *Macros* subheading under the following commands for examples of how to use this command: **{CASE}**, **{GO}**, **{ON CANCEL}**.

Merge

In a merge, the rest of the primary file after the **{QUIT}** command is written out to the resulting merged document before the merge terminates, but any commands after **{QUIT}** are not executed. Arguments in the commands following the **{QUIT}** command are included as text in the merged document. If you don't want the rest of the primary file written out, use the **{STOP}** command instead of **{QUIT}** (see *{STOP}* below).

If **{QUIT}** is used in a secondary file, the secondary file is abandoned at that point. However, the rest of the primary file is written out (but not processed) just as if the command had been encountered in the primary file, and then the merge terminates.

The following example is a standard memo sent by a Personnel department to each employee before his or her six-month salary review. The merge ends just after the {DATE} command, but the rest of the memo is written out to the resulting merged document.

MEMO

To: {KEYBOARD}
From: Kim Charleston, Personnel
Date: {DATE}{QUIT}
Subject: Six-month Review Preparation

•
• (rest of memo)
•

{RESTART} ↻

The {RESTART} macro command terminates all macro execution at the end of the current nested macro. This command can be used if you do not want a macro to return to the macro from which it was nested. The {RESTART} command can be inserted anywhere in the nested macro. The macro “remembers” the {RESTART} command and executes it after all other commands have been executed.

Macros

In the following example, the parent macro nests the macro CONTINUE.WPM which asks the user whether he or she wants to continue or stop. If the user elects to continue, execution returns to the parent file. If the user elects to stop, the {RESTART} command prevents execution from returning after the last command of the nested file.

Parent File:

•
•
•
{NEST}Continue~
•
•
•

Nested File (CONTINUE.WPM):

```
{CHAR}Answer~1-Continue;-2-Stop:~1{Left}~  
    {;}Prompt user~  
{IF}{VARIABLE}Answer~=1~  
    {;}If user elects to continue...~  
{RETURN}
```

```

        {;}...return-execution-to-the-parent-file.~
{ELSE}
        {;}Otherwise...~
        {RESTART}
        {;}...terminate-execution-at-the-end-of-this-file.~
{END IF}
        {;}End-of-IF-statement~
{PROMPT} You-have-elected-to-stop-the-macro.-Press-Enter-to-terminate-execution.~
        {;}Prompt-user~
{PAUSE}
        {;}Wait-for-user-to-press-Enter~
{Screen} {Screen}
        {;}Clear-the-screen.-Execution-stops-after-this-command.~

```

Merge

This command is not available in Merge.

{RETURN} ↻ ➤

The {RETURN} command marks the end of a subroutine and signals the macro or merge to return from a {CALL} or {CASE CALL} command.

Macros

If there is no {CALL} or {CASE CALL} to return to and the macro file containing this command is nested, {RETURN} signals the macro to return to the file from which it was nested. If the command is not in a nested file and there is no {CALL} or {CASE CALL} to return to, {RETURN} marks the end of a macro (see *{CALL}* above).

See the *Macros* subheading under the following commands for examples of how to use this command: {CALL}, {CHAIN}, {Item Down}, {Item Left}, {Item Right}, {Item Up}.

Merge

In Merge, the {RETURN} command must be paired with a {CALL} or {CASE CALL} command. If there is no {CALL} or {CASE CALL} to return to, an error message will be displayed.

See the *Merge* subheading under {CALL} for an example of how to use this command.

{RETURN CANCEL} ↻ ➤

The {RETURN CANCEL} command causes execution to leave the current level and indicates a Cancel to the next higher level (see *Levels* under *Notes* at the end of this appendix).

Macros

Since {RETURN CANCEL} is the default action to Cancel when no {ON CANCEL} command is used (see *{ON CANCEL}* above), the {RETURN CANCEL} command is most often used to reset the {ON CANCEL} action back to the default after it has been changed.

In this example, the {RETURN CANCEL} is used to set the {ON CANCEL} action to {RETURN CANCEL}.

```
{ON CANCEL}{GO}Send Message~
```

- (During this part of the macro, if the user presses Cancel, execution will be transferred to the Send Message label.)

```
{ON CANCEL}{RETURN CANCEL}~
```

- (During this part of the macro, if the user presses Cancel, a {RETURN CANCEL} is returned to the higher level.)

Merge

In this example, the {RETURN CANCEL} is used to set the {ON CANCEL} action to {RETURN CANCEL}.

```
{ON CANCEL}{GO}Send Message~
```

- (During this part of the merge, if the user presses Cancel, execution will be transferred to the Send Message label.)

```
{ON CANCEL}{RETURN CANCEL}~
```

- (During this part of the merge, if the user presses, a {RETURN CANCEL} is returned to the higher level.)

{RETURN ERROR}

The {RETURN ERROR} command causes execution to leave the current level and indicate an error to the next higher level (see *Levels* under *Notes* at the end of this appendix) (see also *ON ERROR* above).

Macros

Since {RETURN ERROR} is the default action when an error occurs and no {ON ERROR} command is used (see *ON ERROR* above), the {RETURN ERROR} command is most often used to reset the {ON ERROR} action back to the default after it has been changed.

In this example, the {RETURN ERROR} is used to reset the {ON ERROR} action back to the default.

```
{ON ERROR}{GO}Send Message~
```

- (During this part of the macro, if an error occurs, execution will be transferred to the Send Message label.)

```
{ON ERROR}{RETURN ERROR}~
```

- (During this part of the macro, if an error occurs, a {RETURN ERROR} is returned to the higher level.)

Merge

In this example, the {RETURN ERROR} is used to set the {ON ERROR} action to {RETURN ERROR}.

```
{ON ERROR}{GO}Send Message~
```

- (During this part of the merge, if an error occurs, execution will be transferred to the Send Message label.)

```
{ON ERROR}{RETURN ERROR}~
```

- (During this part of the merge, if an error occurs, a {RETURN ERROR} is returned to the higher level.)

{RETURN NOT FOUND} (↺)

The {RETURN NOT FOUND} macro command terminates macro execution on the current level and indicates a search Not Found condition to the next higher level (see *{ON NOT FOUND}* above). {RETURN NOT FOUND} can be used wherever you would use {RETURN} (see *{RETURN}* above).

Since {RETURN NOT FOUND} is the default action when a *Not Found* condition occurs and no {ON NOT FOUND} command is used (see *{ON NOT FOUND}* above), the {RETURN NOT FOUND} command is most often used to reset the {ON NOT FOUND} action back to the default after it has been changed.

Macros

In this example, the {RETURN NOT FOUND} is used to reset the {ON NOT FOUND} action back to the default.

```
{ON NOT FOUND}{GO}Send Message~
```

- (During this part of the macro, if a search string is not found, execution will be transferred to the Send Message label.)

```
{ON NOT FOUND}{RETURN NOT FOUND}~
```

- (During this part of the macro, if a search string is not found, a {RETURN NOT FOUND} is returned to the higher level.)

Merge

This command is not available in Merge.

{REWRITE} (↺) ↘

This merge command causes the screen to be rewritten. Since none of the merged document is written to the screen during a merge, you may want to use this command to display what has been merged at a certain point.

In previous versions of WordPerfect, this merge command was represented as ^U.

Macros

This command is not available in Macros; however, you can use the following (inserted in the Macro Editor or while defining a macro at the normal editing screen by pressing **Screen** (Ctrl-F3) twice):

```
{Screen}{Screen}
```

You can also use the {DISPLAY ON} command (see */DISPLAY ON* above).

Merge

In the following example, the {REWRITE} command is used so that the letter can be seen after it is merged.

```
ABC Company  
245 West Center Street  
Long Beach, California 90807
```

{DATE}

```
{FIELD}Name~  
{FIELD}Company~
```

```
Dear {FIELD}Salutation~:
```

```
Thank you for your inquiry regarding our new product.
```

-
- (rest of the letter)
-

```
Sincerely,
```

```
Amy Wilcox  
Product Manager{COMMENT}  
~{REWRITE}
```

{SHELL ASSIGN}shellvar~ expr~ ↻

The {SHELL ASSIGN} command assigns the value returned by *expr* to the Shell variable *shellvar*. Expressions are evaluated (see *Appendix J: Macros and Merge, Expressions*) and the result is assigned to the variable. Non-numeric characters and expressions that cannot be evaluated are treated as strings.

This command is only available if you are running WordPerfect under Shell 3.0 (or later). If you have a previous version of Shell (or if you do not own Shell), this command will do nothing.

After a value has been assigned to a variable, the variable command {SHELL VARIABLE}shellvar~ can be placed anywhere you would normally place the variable contents.

For more information on Shell variables, see the documentation that accompanies the Shell program.

{SHELL MACRO}*macroname*

The {SHELL MACRO} macro command invokes a Shell macro. This is useful when switching between various WordPerfect Corporation products.

This command is only available if you have Shell version 3.0. If you have a previous version of Shell (or if you do not own Shell), this command will do nothing.

You do not need to include the .SHM extension in *macroname*. However, you must include a path if the Shell macro is in a directory other than the directory specified in Location of Macro Files in Shell Setup (see your Shell documentation).

Macros

You can use the macro in the following example to execute a Shell macro, or to let you know why if it can't be executed.

```
{IF}{SYSTEM}ShellVer>2*256~
    {;}If Shell is version 3.0 or later~
    {SHELL MACRO}c:\shm\test~
    {;}Execute the Shell macro TEST.SHM~
{ELSE}
{IF}{SYSTEM}ShellVer=0~
    {;}Otherwise, if no Shell is running~
    {CHAR}AnyKey"ERROR: Shell not present. Press any key to continue."~
    {;}Inform user~
{ELSE}
    {;}Otherwise (if a Shell is running but is a version earlier than 3.0)~
    {CHAR}AnyKey"ERROR: Shell wrong version. Press any key to continue."~
    {;}Inform user~
{END IF}
{END IF}
```

Merge

This command is not available in Merge.

{SHELL VARIABLE}*shellvar*

This command accesses the contents of Shell variables. After a value has been assigned to a Shell variable (see {SHELL ASSIGN} above) the {SHELL VARIABLE}*shellvar* command can be placed anywhere you would normally place the variable contents. It can be placed within or as an argument for another command, or can be used by itself.

This command is only available if you have Shell version 3.0 or later. If you have a previous version of Shell (or if you do not own Shell), this command will do nothing.

For more information on Shell variables, see the documentation that accompanies the Shell program.

{SPEED}100ths second ↻

The {SPEED} macro command can slow down macro execution. It causes macro execution to wait the amount of time indicated by the *100ths second* argument between each command.

The default speed is no delay between commands (i.e., {SPEED}0).

Macros

For example, if you want macro commands to execute every 1.5 seconds, insert the following into your macro:

```
{SPEED}150
```

Merge

This command is not available in Merge.

{STATE} ↻

The {STATE} macro command returns a number representing the current operational state of WordPerfect. This lets you create macros which are aware of the environment in which they are executing. The operational states and their corresponding code numbers are listed below.

| | |
|-------|---|
| 3 | Current Document (1,2) |
| 4 | Normal Editing Screen |
| 8 | Editing Structure Other than Normal Editing Screen |
| 16 | Macro Definition Active |
| 32 | Macro Execution Active (always set) |
| 64 | Merge Active |
| 128 | Block Active |
| 256 | Typeover Active |
| 512 | Reveal Codes Active |
| 1024 | Yes/No Question Active |
| 2048 | In a list (See also the List system variable description under {SYSTEM} below.) |
| 4096 | Help Active |
| 32768 | Cannot go to Shell |

State 8 (Editing Structure Other than Normal Editing Screen) refers to a screen which is used for editing footnotes, headers, styles, etc. Macro Execution (32) is labeled as "always set" because the {STATE} command is only used in a macro as it is executing.

You can determine what the state of WordPerfect is by forming an AND (&) expression with a value called a *mask* (e.g., {STATE}&3). The result of the operation indicates the current state of WordPerfect.

To choose a mask, determine which state(s) you want to check for. Note the numbers associated with each state and add them together to calculate the mask value. For example, if you want to know what document you are currently in (1, 2, or 3), the mask value is 3. If you want to know if you are at the normal editing screen (4) and/or if Reveal Codes is active (512), the mask value is 516 (4+512=516).

Document 3 is a temporary document used during a merge. It is not normally accessible; however, you can check to see whether it is active with the {STATE} command.

After you have determined the appropriate mask, create an AND expression, then assign the result to a variable. For example,

```
{ASSIGN}DocNum={STATE}&3`
    {;}Assign-the-current-document-number-to-var-DocNum`
{ASSIGN}Active={STATE}&516`
    {;}Assign-the-result-(either-4,-512,-516,-or-0)-to-var-Active`
```

In this example, the mask values are 3 and 516. Variable DocNum contains the current document number and variable Active contains a number which indicates whether the normal editing screen (4), Reveal Codes (512), both (516), or neither (0) are active.

If the result of the AND operation is 0, then the state you were checking for is not present. If the result is a non-zero number, then some (or all) of the states you checked for are present.

For example, if you want to check for both types of editing screens ((4) and (8)), the mask is 12. {STATE}&12 gives four types of information. If the result is 0, then neither the normal editing screen nor another editing screen is active (some type of menu is active). If the result is 4, the normal editing screen is on. If the result is 8, a menu is active, but you are editing a style, footnote, etc. (e.g., you pressed Format while editing a footnote). If the result is 12, you are in the normal editing screen, and you are editing a style, footnote, etc.

Since the {IF} command interprets 0 as false, you can form {IF} statements that will perform functions when a certain condition exists. For example, the following macro returns you to the normal editing screen if Reveal Codes, Block, both, or neither is on.

```
{LABEL}Top`
{IF}{STATE}&4`
    {;}If-at-the-normal-editing-screen`
{IF}{STATE}&512`
    {;}And-Reveal-Codes-is-on`
    {Reveal Codes}
    {;}Turn-Reveal-Codes-off`
{ELSE}
    {;}Otherwise`
{IF}{STATE}&128`
    {;}If-Block-is-on`
    {Block}
    {;}Turn-off-Block`
{ELSE}
    {;}Otherwise`
    {RETURN}
    {;}Exit-the-loop`
{END IF}
{END IF}
{END IF}
{GO}Top`
    {;}Repeat-the-loop-until-the-{RETURN}-command-exits-it`
```

When {STATE} is executed by itself (not in an expression), it returns a number which represents the total state of WordPerfect. All applicable numbers are added together. For example, if the cursor is in the normal editing screen (4) of document 1 (1) and if Block is on (128) and a macro is executing (32), then the executional state of WordPerfect is $4+1+128+32=165$.

See also *{SYSTEM}* for information on accessing other system variables.

Macros

See the examples in the above description of this command.

Merge

This command is not available in Merge.

{STATUS PROMPT}message~

This command puts a message on the status line. Although you can use cursor positioning commands (see *Message Display* under *Notes* at the end of this appendix) to position the message elsewhere on the screen, the message may not redisplay correctly when the screen is rewritten if you do. Also, the {STATUS PROMPT} message only displays when the status line normally displays. So, for example, in Merge, the message would only display when a {KEYBOARD} command pauses the merge for input from the keyboard, and when the merge is finished.

If you position the message on the status line, only the first 48 characters will show.

When you use this command, the message is stored in memory, much like a variable. However, this spot in memory is shared with the {INPUT} command. If you use an {INPUT} command after a {STATUS PROMPT} command, the {STATUS PROMPT} message will be replaced in memory by the {INPUT} message. Since the {INPUT} command clears its own message from memory when execution continues after the command, the {STATUS PROMPT} message no longer exists after the {INPUT} command. This is one way to clear the {STATUS PROMPT} message from memory.

To clear a status prompt message without using {INPUT}, insert another {STATUS PROMPT} command with no message ({STATUS PROMPT}~). If you do not clear the message with an {INPUT} or {STATUS PROMPT}~ command, the message will be on the screen whenever the status line is displayed until you exit WordPerfect.

For additional methods of prompting the user, see *Prompting and User Input* under *Notes* at the end of this appendix.

Macros

In the following example, the {STATUS PROMPT} is used to display the date the macro was created on the status line before the rest of the macro executes.

```
{DISPLAY ON}
  {;}Turn Display on so the subsequent {STATUS PROMPT} message will show~
{STATUS PROMPT}Macro created: 10/25/89~
```

```

        {;}Display the creation date on the status-line~
{WAIT}15~
        {;}Give the user 1.5-seconds to read the message~
{STATUS PROMPT}~
        {;}Clear the {STATUS PROMPT} message from memory~
{DISPLAY OFF}
        {;}Turn Display off so that the rest of execution will not display.~
•
• (Rest of macro)
•

```

See the *Macros* subheading under the following commands for additional examples: {LOOK}, {PAUSE}.

Merge

The first command puts the message “Press F9 when done.” on the status line. The second command erases the message.

```

{STATUS PROMPT}Press F9 when done.~{COMMENT}
~{KEYBOARD}{COMMENT}
        When the merge pauses at the {KEYBOARD} command, the message is displayed
        on the status line.
~{STATUS PROMPT}~{COMMENT}
        This clears the {STATUS PROMPT} message from memory~

```

{STEP OFF}

The {STEP OFF} command turns off single step execution after it has been turned on (see {STEP ON} below).

Macros

See the example under {STEP ON} below.

Merge

See the example under {STEP ON} below.

{STEP ON}

The {STEP ON} command is useful for debugging macros and merges. It causes the macro or merge to execute one step at a time. Between each step, a message on the status line indicates what the next key or command is. The key or command executes when any key is pressed (see the *Macros* and *Merge* subheadings below).

During macro execution, press **Exit** (F7) to turn off the Step mode. Pressing **Cancel** (F1) terminates execution unless Cancel is turned off or is redefined (see {CANCEL OFF} and {ON CANCEL} above).

Macros

While step is on in macro execution, if the next step in the macro is a character (e.g., A), that character will be displayed. If it is a command, a label followed by a number will be displayed. The four labels are as follows:

| Label | Meaning |
|--------------------|---|
| ALT X | Alt- <i>letter</i> Macro Execution |
| KEY CMD <i>n</i> | WordPerfect Command, Cursor Control, etc. |
| KEY MACRO <i>n</i> | Soft Keyboard Macro Execution |
| MACRO CMD <i>n</i> | Specific Macro Command |

The *X* and *n* in the table above represent the letter or number that identifies the specific command of that type. Alt-*letter* macro commands are identified by the letter to which they are assigned. Soft keyboard macro commands are identified by the number assigned to the macro by the Keyboard Layout feature (see *Keyboard Layout* in *Reference*). Variables are identified by name. Keystroke commands and macro commands are identified by special code numbers that are listed below.

Keystroke Command Codes (KEY CMD)

| | |
|---|----------------------------|
| 1 ^A | 26 ^Z – Down |
| 2 ^B – Page Number | 27 ^ [– Escape |
| 3 ^C – Merge from Console | 28 ^\ |
| 4 ^D – Merge Date | 29 ^] |
| 5 ^E – Merge End Record | 30 ^^ – Reset Keyboard Map |
| 6 ^F – Merge Field | 31 ^_ |
| 7 ^G – Merge Macro | 32 Cancel |
| 8 ^H – Home | 33 Forward Search |
| 9 ^I – Tab | 34 Help |
| 10 ^J – Enter | 35 Indent |
| 11 ^K – Delete to End of Line | 36 List |
| 12 ^L – Delete to End of Page | 37 Bold |
| 13 ^M – Search Value for [SRt] | 38 Exit |
| 14 ^N – Merge Next Record | 39 Underline |
| 15 ^O – Merge Output Prompt | 40 End Field |
| 16 ^P – Merge Primary Filename | 41 Save |
| 17 ^Q – Merge Quit | 44 Setup |
| 18 ^R – Merge End Field | 45 Backwards Search |
| 19 ^S – Merge Secondary Filename | 46 Switch |
| 20 ^T – Merge Text to Printer | 47 Left/Right Indent |
| 21 ^U – Merge Update the Screen | 48 Date/Outline |
| 22 ^V – Ignore Meaning of Following Code | 49 Center |
| 23 ^W – Up | 50 Print |
| 24 ^X – Right & Search Wildcard – Same as Typical “?” | 51 Format |
| 25 ^Y – Left | 52 Merge Codes |
| | 53 Retrieve |
| | 56 Thesaurus |
| | 57 Replace |
| | 58 Reveal Codes |
| | 59 Block |
| | 60 Mark Text |

- | | | | |
|----|---|-----|---|
| 61 | Flush Right | 89 | PgUp |
| 62 | Columns/Table | 90 | PgDn |
| 63 | Style | 91 | Screen Down (by hitting "+" on numeric keypad) |
| 64 | Graphics | 92 | Screen Up (by pressing "-" on numeric keypad) |
| 65 | Macro | 93 | Typeover |
| 68 | Shell | 94 | Left Margin Release (reverse tab) |
| 69 | Spell | 95 | Hard Page (Ctrl-Enter) |
| 70 | Screen | 96 | Soft Hyphen (Ctrl--) |
| 71 | Move | 97 | Hyphen |
| 72 | Text In/Out | 98 | Required (Hard) Space (Home.Space Bar) |
| 73 | Tab Align | 99 | Para Up |
| 74 | Footnote | 100 | Para Down |
| 75 | Font | 101 | Item Left |
| 76 | Merge/Sort | 102 | Item Right |
| 77 | Macro Define | 103 | Item Up |
| 80 | Backspace | 104 | Item Down |
| 81 | Delete Right | 105 | Alt-Home |
| 82 | Delete Word (Ctrl- Backspace) | 106 | Delete Row (Ctrl-Delete)107 Menu Bar (Alt=) |
| 83 | Word Right | 108 | Block Append |
| 84 | Word Left | 109 | Block Move |
| 85 | Home.Home.Right (by pressing end key) | 110 | Block Copy |
| 86 | Home.Home.Left (by pressing begin key on the Victor computer) | | |
| 88 | GoTo (Ctrl-Home) | | |

Macro Command Codes (MACRO CMD)

- | | | | |
|----|---------------|----|----------------------|
| 1 | {ASSIGN} | 19 | {FOR EACH} |
| 2 | {BELL} | 20 | {GO} |
| 3 | {BREAK} | 21 | {IF} |
| 4 | {CALL} | 22 | {LABEL} |
| 5 | {CANCEL OFF} | 23 | {LOOK} |
| 6 | {CANCEL ON} | 24 | {NEST} |
| 7 | {CASE} | 25 | {NEXT} |
| 8 | {CASE CALL} | 26 | {SHELL MACRO} |
| 9 | {CHAIN} | 27 | {ON CANCEL} |
| 10 | {CHAR} | 28 | {ON ERROR} |
| 11 | {;} (comment) | 29 | {ON NOT FOUND} |
| 12 | {DISPLAY OFF} | 30 | {PAUSE} |
| 13 | {DISPLAY ON} | 31 | {PROMPT}32 {QUIT} |
| 14 | {ELSE} | 33 | {RESTART} |
| 15 | {END FOR} | 34 | {RETURN} |
| 16 | {END IF} | 35 | {RETURN CANCEL} |
| 17 | {END WHILE} | 36 | {RETURN ERROR} |
| 18 | {FOR} | | |

| | |
|-----------------------|---------------------|
| 37 {RETURN NOT FOUND} | 50 {STATUS PROMPT} |
| 38 {SPEED} | 51 {INPUT} |
| 39 {STEP ON} | 52 {VARIABLE} |
| 40 {TEXT} | 53 {SYSTEM} |
| 41 {STATE} | 54 {MID} |
| 42 {WAIT} | 55 {NTOK} |
| 43 {WHILE} | 56 {KTON} |
| 44 {Macro Commands} | 57 {LEN} |
| 45 {STEP OFF} | 58 [~] (hard tilde) |
| 46 {ORIGINAL KEY} | 59 {PAUSE KEY} |
| 47 {IF EXISTS} | 61 {OTHERWISE} |
| 48 {MENU OFF} | 62 {SHELL ASSIGN} |
| 49 {MENU ON} | 63 {SHELL VARIABLE} |

In Macros, the {STEP ON} feature is particularly useful when you want to track the contents of a variable. When a variable is encountered during macro execution with Step on, MACRO CMD 52 (for {VARIABLE}) is first displayed, then each letter of the name of the variable is displayed. Then its contents (if they exist) are displayed one character at a time. For example, if variable Num contains 14, the first message, MACRO CMD 52, is followed by an N, then a u, then an m, then a tilde (~), which are then followed by a l, then a 4.

If the execution command for a variable was entered as {VAR #}, VAR # is displayed instead of MACRO CMD 52. The contents are then displayed one character at a time as usual.

```

•
• (This section will execute normally.)
•
{STEP ON}
•
• (This section will execute one keystroke at a time.)
•
{STEP OFF}
•
• (This section will execute normally.)
•

```

Merge

When step is on in Merge, if the next thing to be executed is a command (e.g., {FIELD}), the command is displayed (there are no code lists as in Macros). Each character is displayed after it is written out to the resulting document.

You may find it more useful to step through merges with Reveal Codes on.

```

•
• (This section will execute normally.)
•

```


{STEP ON}

- (This section will execute one keystroke at a time.)

{STEP OFF}

- (This section will execute normally.)

{STOP} (C) ➤

This merge command stops all execution when it is encountered. It is similar to the {QUIT} command (see {QUIT} above) except that the rest of the primary file is not read in. If this command is found in a nested file, execution is not returned to the parent file. Chained files are also not executed.

While an executing merge is paused at a {KEYBOARD} or {INPUT} command, you can execute the {STOP} command from the keyboard. See *Inserting Merge Commands During Execution* under *Notes* at the end of this appendix.

Macros

This command is not available in Macros; however, {QUIT} in Macros is equivalent to {STOP} in Merge (see {QUIT} above).

Merge

See the *Merge* subheading under the following commands for examples of how to use this command: {BELL}, {CASE}.

{SUBST PRIMARY}filename~ ➤

This merge command is similar to the {NEST PRIMARY} and {CHAIN PRIMARY} commands, except with this command, the named primary file is used instead of the current primary file from the point of this command on. You are never returned to the original primary file; no commands following this one in the original primary file are executed. Local variables in a previous primary file are erased.

If you substitute a file that is not found, or no file ({SUBST PRIMARY}~), an error will be returned. You can use the {ON ERROR} command to determine what should happen if this error occurs (see {ON ERROR} above).

Macros

This command is not available in Macros.

Merge

In the following example, the secondary file has a field named DaysOverDue. The merge begins using the primary file INVOICE.PF, which substitutes a different primary file depending on the number in the DaysOverDue field. At the end of each substituted file, the record pointer is moved to the next record and INVOICE.PF is substituted so that it will execute again.

Primary file INVOICE.PF:

```
{IF}{FIELD}DaysOverDue<30{COMMENT}
  {SUBST PRIMARY}0-29.PF{COMMENT}
~{ELSE}{COMMENT}
  {IF}{FIELD}DaysOverDue<60{COMMENT}
    {SUBST PRIMARY}30-59.PF{COMMENT}
  ~{ELSE}{COMMENT}
    {IF}{FIELD}DaysOverDue<90{COMMENT}
      {SUBST PRIMARY}60-89.PF{COMMENT}
    ~{ELSE}{COMMENT}
      {SUBST PRIMARY}Over90.PF{COMMENT}
    ~{END IF}{COMMENT}
  ~{END IF}{COMMENT}
~{END IF}
```

Structure of primary files 0-29.PF, 30-59.PF, 60-89.PF, and Over90.PF:

```
*
• (body of file)
*
{NEXT RECORD}{COMMENT}
~{SUBST PRIMARY}Invoice.pf
```

{SUBST SECONDARY}filename

This merge command changes to the named secondary file and uses the first record in that file. If you substitute a file that is not found or no file (`{SUBST PRIMARY}~`), an error will be returned. You can use the `{ON ERROR}` command to determine what should happen if this error occurs (see *ON ERROR* above).

In previous versions of WordPerfect, this merge command was represented as ^Sfilename^S.

Macros

This command is not available in Macros.

Merge

In the following example, two secondary files are used, each sorted by ZIP Code in descending order so that any records where the Zip field is blank will be at the end of the file. The `{SUBST SECONDARY}` command in the primary file below causes the records with no ZIP Code to be skipped, and the next secondary file to be used.

```
{IF BLANK}Zip~{SUBST SECONDARY}File2~{COMMENT}
  If the Zip field is blank, substitute the secondary file
~{END IF}
```

{SYSTEM}sysvar

The `{SYSTEM}` command returns the value of the given system variable. These system variables allow the macro or merge to be aware of the current state of

WordPerfect. You can use mask values (see *[STATE]* above) to check for multiple system variables. Valid system variables are listed below.

Rather than use the name of the system variable in the {SYSTEM} command, you can use the number given in parentheses next to each system variable name below. Using the number instead of the name is especially useful when you want to use the same macro in different international versions of WordPerfect. Since the system variable names are translated in international versions, running an English version of a macro would create an error at the {SYSTEM} command. The numbers, however, are the same across international versions.

Some of these system variables have restrictions when used in Merge. See the description of each variable for possible restrictions.

| Sysvar | Value(s) Returned |
|---------------|--------------------------|
| Attrib (1) | Current font attribute: |
| | 0 Normal |
| | 1 Extra Large |
| | 2 Very Large |
| | 4 Large |
| | 8 Small |
| | 16 Fine |
| | 32 Superscript |
| | 64 Subscript |
| | 128 Outline |
| | 256 Italics |
| | 512 Shadow |
| | 1024 Redline |
| | 2048 Double Underline |
| | 4096 Bold |
| | 8192 Strikeout |
| | 16384 Underline |
| | 32768 Small Caps |

If two attributes are present, their respective numbers will be added together.

| | |
|----------------|--|
| Cell (2) | Current cell position in a table, e.g., A4, E7, etc. This system variable is undefined if the cursor is not in a table when the command is encountered. |
| CellAttr (23) | Attributes of the current cell (see <i>Attrib</i> above for values). This system variable is undefined if the cursor is not in a table when the command is encountered. |
| CellState (24) | State of a cell. Divide the value returned by 256 (<i>value returned/256</i> —see <i>Appendix J: Macros and Merge, Expressions</i>) to determine the following states: |
| | 0 Left justified. |
| | 1 Full justified. |
| | 2 Center justified. |

| Sysvar | Value(s) Returned |
|----------------|---|
| | 3 Right justified. |
| | 4 Decimal aligned. |
| | Mod the value returned by 256 (<i>value returned%256</i> —see <i>Appendix J: Macros and Merge, Expressions</i>) to determine the following states: |
| | 1 Justify is cell-specific. |
| | 2 Attribute is cell-specific. |
| | 4 Cell is bottom-aligned. |
| | 8 Cell is center-aligned. |
| | 16 Contents type is "text." |
| | 32 Contents is a formula. |
| | 64 Cell is locked. |
| | This system variable is undefined if the cursor is not in a table when the command is encountered. |
| Column (3) | Current column number (numbered sequentially from left to right) in a table or in columns. |
| Direction (31) | Text entry direction. |
| | 0 Direction is left-to-right. |
| | 1 Direction is right-to-left (e.g., the Arabic version of WordPerfect). |
| Document (4) | Current modification status of the document on the screen: |
| | 1 Document has been modified |
| | 4 Document has been modified since last generated. |
| | 256 Document is blank. (Blank documents are those that appear in document screens 1 and 2 when you first start WordPerfect, and when you exit to a clear screen. A document from which you delete all text and codes is not "blank.") |
| | 512 Cursor is between [Tbl Def] and [Tbl Off] codes (in a table). |
| | 1024 Cursor is between [Math On] and [Math Off] or the end of the file (Math is on). |
| | 2048 Cursor is between [Outline On] and [Outline Off] or the end of the file (Outline is on). |

| Sysvar | Value(s) Returned |
|---------------|--|
| | 4096 Cursor is between [Column On] and [Column Off] or the end of the file (Column is on). All other values are undefined and not guaranteed to be 0. |
| EditType (33) | Current editing mode. Returns information useful for international keyboards that need to use ASCII mnemonics in menus. 0 Cursor is in a menu prompt. 1 Cursor is in a main editing screen (e.g., Doc 1 or 2). 2 Cursor is in a substructure editing screen (e.g., Header/Footer edit). 4 Cursor is in line editing mode (e.g., entering a description for a paper/size type). 8 Cursor is in window editing (e.g., the Document Comment window). |
| Endnote (5) | Number of the current endnote. |
| Entry (29) | When in a list, the string name of the currently highlighted entry. This system variable is not available when the cursor is not in a list (see <i>List</i> below) or in the following lists: Keyboard Edit, Macro Commands, Merge Commands, and Equation Palette. |
| Equation (6) | Number of the current equation, according to the following formula: Return value/32=first level Return value%32=second level For example, if the current equation is 1.2, {SYSTEM}Equation~ will return 34 (i.e., 34/32=1, 34%32=2). |
| Figure (7) | Number of the current figure, according to the formula described under system variable <i>Equation</i> above. |
| Footnote (8) | Number of the current footnote. |
| KeyState (26) | Current keyboard status. 1 Right Shift key pressed 2 Left Shift key pressed. 4 Ctrl key pressed. 8 Alt key pressed. 16 Scroll Lock active. 32 Num Lock active. |

| Sysvar | Value(s) Returned |
|---------------|--|
| | 64 Caps Lock active. |
| | 128 Insert active. |
| Language (32) | Current language as found under Format: Other (Shift-F8,4,4). A two-letter abbreviation is returned. |
| | AF Afrikaans |
| | CA Catalan |
| | HR Croatian |
| | CZ Czechoslovakian |
| | DK Danish |
| | NL Dutch |
| | OZ English—Australia |
| | CE English—Canada |
| | UK English—United Kingdom |
| | US English—United States |
| | SU Finnish |
| | CF French—Canada |
| | FR French—France |
| | GA Galician |
| | DE German—Germany |
| | SD German—Switzerland |
| | GR Greek |
| | MA Hungarian |
| | IS Icelandic |
| | IT Italian |
| | NO Norwegian |
| | BR Portuguese—Brazil |
| | PO Portuguese—Portugal |
| | RU Russian |
| | SL Slovak |
| | ES Spanish |
| | SV Swedish |
| | YK Ukranian |
| Left (9) | Item (character or code) immediately to the left of the cursor (see <i>Appendix T: Macros and Merge, Value Tables</i>). |
| Line (10) | Vertical position of the cursor in 1200ths of an inch. |
| List (11) | Number of items in the current list. (For purposes of this system variable, a <i>list</i> is any list in WordPerfect where you can perform a name search.) |
| | 65535 The cursor is not in a list. |
| | 0 The list is empty. |

| Sysvar | Value(s) Returned |
|---------------|--|
| | <p>Other The number of items in the list. In List Files, "Current" and "Parent" each count as an item on the list. Therefore, {SYSTEM>List~ while in List Files returns 2 plus the number of files in the list (e.g., if there are 3 files in the list, {SYSTEM>List~ returns 5; if there are no files in the list, it returns 2).</p> |
| Menu (13) | Number of the menu currently active (see <i>Appendix T: Macros and Merge Value Tables</i>). |
| Name (12) | Name of the current document, (e.g., JONES.LTR.) Since there is no filename associated with the merged document while the merge is executing, this system variable is not available in Merge. |
| Network (30) | Current network status. <ul style="list-style-type: none"> 0 WordPerfect is not operating in a network environment. 1 WordPerfect is operating in a network environment. |
| Page (14) | Current page number. |
| Path (15) | Path to the current document. (e.g., C:\WP51\). (Note the slash on the end of the path.) Since there is no path associated with the merged document during a merge, this system variable is not available in Merge. |
| Pos (16) | Current horizontal cursor position in WordPerfect units (1200ths of an inch). |
| Print (17) | Current print status. <ul style="list-style-type: none"> 1 No characters have been sent to printer. 2 An attempt has been made to send characters to printer. 8 Printer is waiting for a Go. 16 Trying to rush job. 32 Trying to cancel job. 64 Network down. 128 Printing in progress. 256 Downloading a file. 2048 Last print job aborted abnormally. <p>All others are undefined and not guaranteed to be 0.</p> |

| Sysvar | Value(s) Returned |
|---------------|---|
| Right (18) | Item on which the cursor is resting. In Merge, this command will return 0 if the cursor is resting on a Soft Return or Soft Page code (see <i>Appendix T: Macros and Merge, Value Tables</i>). |
| Row (22) | Current row number in a table (equals 0 if the cursor is not in a table when the command is encountered). |
| RowState (27) | Header status of current row. 0 Not in a table, or in a table but current row is not a header row. 1 In a table, and current row is a header row. |
| ShellVer (25) | Current Shell version number. (Shell is an optional WordPerfect Corporation program.) The formula for determining the version from the number returned is: (Major Version# * 256) + Minor Version# For example, if you were running WordPerfect under Shell version 1.1, 257 would be returned (i.e., (1*256)+1). If you were running under Shell 2.0, 512 would be returned (i.e., (2*256)+0). |
| TableBox (19) | Number of the current Table box, according to the formula described under system variable <i>Equation</i> above. |
| TextBox (20) | Number of the current Text box, according to the formula described under system variable <i>Equation</i> above. |
| UserBox (21) | Number of the current User-Defined box, according to the formula described under system variable <i>Equation</i> above. |
| Version (28) | Current WordPerfect version number. This system variable works for version 5.1 and later, but will produce an error in version 5.0. The formula for determining the version from the number returned is: (Major Version# * 256) + Minor Version# For example, if you are running WordPerfect version 5.1, 1281 is returned (i.e., (5*256)+1). |

Macros

The following example sends a message to the screen while printing is in progress, then another message when printing has stopped.

```
{PROMPT}Printing is in progress...`
      {;}Send message to user`
{WHILE}{SYSTEM}Print`&128`
      {;}While printing is in progress`
```



```

{WAIT}100~
  {;}Let WordPerfect print~
{END WHILE}
  {;}End {WHILE} loop~
{PROMPT}Printing has stopped.~{WAIT}40~
  {;}Send new message to user.--Display it for 4 seconds.~

```

The {WAIT} command is very important in this macro because it gives the computer processor time to print. If it weren't there, nothing would print because the macro loop would take all of the computer processor's time.

See the *Macros* subheading under the following commands for additional examples: {Item Down}, {Item Left}, {Item Right}, {Item Up}, {SHELL MACRO}.

Merge

In the following example, the message "Document is printing..." displays while printing is in progress.

```

{PRINT}{COMMENT}
  Print what has been merged to this point
~{PROMPT}Document is printing...~{COMMENT}
  Send message to user
~{WHILE}{SYSTEM}Print~&128~{COMMENT}
  While printing is in progress
  ~{WAIT}100~{COMMENT}
  Wait 10 seconds to allow printing to continue
~{END WHILE}{COMMENT}
  Repeat the WHILE loop~

```

{TEXT}var~message~ ↻ ➤

The {TEXT} command prompts the user by displaying a message on the status line. The input (up to 129 characters) from the user is then assigned to the variable (see {CHAR} above). See *Message Display* under *Notes* at the end of this appendix for information on affecting the way messages are displayed. See also *Prompting and User Input* under *Notes* at the end of this appendix for additional methods of obtaining user input.

After the {TEXT} command executes, the contents of the status line just previous to the execution of the {TEXT} command are restored, even if the contents are the message of a previously executed {PROMPT} or {STATUS PROMPT} message to reappear after the {TEXT} command. Use {PROMPT}~ or {STATUS PROMPT}~ before the {TEXT} command to clear the status line.

Macros

See the *Macros* subheading under the following commands for examples of how to use this command: {BELL}, {BREAK}, {BREAK}, {IF EXISTS}, {LEN}, {MENU OFF}, {NTOK}, {ON ERROR}.

Merge

See the *Merge* subheading under the following commands for examples of how to use this command: {BELL}, {IF EXISTS}, {NTOC}.

{VARIABLE}var~ ↻ ➤

This command accesses the contents of global and local (Merge) variables. If you have both a global and local variable by the same name, this command accesses the local variable. There is no way to access global variables while local variables of the same name exist. (For an explanation of global and local variables, see *Appendix L: Macros and Merge, Variables*.)

You can also pass information between macro variables and Shell variables (Shell version 3.0 or higher) using the {SHELL ASSIGN} and {SHELL VARIABLE} commands (see {SHELL ASSIGN} and {SHELL VARIABLE} above).

After a value has been assigned to a variable (see {ASSIGN} and {LOCAL} above), the {VARIABLE}var~ command can be placed anywhere you would normally place the variable contents. It can be placed within or as an argument for another command, or by itself.

A variable can hold no more than 129 keystrokes (characters).

Macros

See the *Macros* subheading under the following commands for examples of how to use this command: {;} (comment), {ASSIGN}, {BELL}, {BREAK}, {CASE}, {CASE CALL}, {CHAR}, {ELSE}, {FOR EACH}, {GO}, {IF}, {Item Down}, {Item Left}, {Item Right}, {Item Up}, {KTON}, {LOOK}, {MENU OFF}, {NEXT}, {NTOK}, {ON ERROR}, {RESTART}, {SHELL ASSIGN}, {SHELL VARIABLE}, {WHILE}.

Merge

See the *Merge* subheading under the following commands for examples of how to use this command: {ASSIGN}, {BELL}, {BREAK}, {CASE}, {CASE CALL}, {COMMENT}, {CTON}, {DOCUMENT}, {IF}, {LEN}, {LOOK}, {MID}, {NEST MACRO}, {NEXT}, {NEXT RECORD}, {NTOC}.

{WAIT}10ths second~ ↻ ➤

The {WAIT} command delays further execution of the macro or merge for the indicated time. This command is useful when you want a message to be displayed for a certain amount of time.

Macros

See the *Macros* subheading under the following commands for examples of how to use this command: {ON CANCEL}, {ON ERROR}, {STATUS PROMPT}, {SYSTEM}.

Merge

See the *Merge* subheading under the following commands for examples of how to use this command: {BELL}, {ON CANCEL}, {ON ERROR}, {SYSTEM}.

{WHILE}expr ↻ ↘

While the expression *expr* is true, the commands between the {WHILE} and the {END WHILE} are repeatedly executed. This command is like the {FOR} command, except that it does not increment a value each time through the loop. In order to end the loop, use another command that will force the expression to be evaluated as false (see *Loops* under *Notes* at the end of this appendix).

Remember that if you use a variable in *expr*, the variable must already exist *before* the {WHILE} command is executed.

Macros

In this example, the message "Counting" will be displayed until variable "Count" reaches 50.

```
{ASSIGN}Count`0`
    {;}Initialize var Count. (This command creates the variable, then
    assigns it "0")
{WHILE}{VARIABLE}Count`<50`
    {PROMPT}Counting`
    {;}Send the prompt "Counting".
    {ASSIGN}Count`{VARIABLE}Count`+1`
    {;}Increment var Count each time through the loop
{END WHILE}
```

See the *Macros* subheading under *{SYSTEM}* for an additional example.

Merge

See the *Merge* subheading under *{SYSTEM}* for an example of how to use this command.

Notes

Chaining, Nesting, and Substituting

Chaining a macro or merge file causes the named file to take over control of execution as soon as execution of the current file is complete. A single file (whether a macro file, a primary merge file, or a secondary merge file) can only use one chain command. If you include more than one chain command, the last one encountered during the merge or macro will be the only one executed. The chained file only executes when the current macro or merge file is completely finished.

The commands that chain are as follows:

```
{CHAIN}                ↻
{CHAIN MACRO}           ↘
{CHAIN PRIMARY}        ↘
{CHAIN SECONDARY}      ↘
```

Nesting is the process of moving control of execution to another file (such as another macro file or primary file). Execution then returns to the parent file (directly after the nest command) when execution of the nested file terminates.

Nesting a macro is similar to calling a subroutine (see *Subroutines* below), except that the nested macro is not a part of the calling macro. It is a separate macro, referenced by giving the macro name or the full pathname if it is not in the Keyboard/Macro Files directory currently specified in Location of Files (see *Location of Files in Reference*). You do not need to include the .WPM extension in the filename or pathname. Because execution is automatically returned when the nested macro has finished, you do not need to place a {RETURN} command at the end of a nested macro.

If there are certain procedures which you frequently use in your macros or merges, you can put them in smaller macros and nest them when they are needed.

You can nest macro and merge files several levels deep (see *Levels* below). The main file nests a second file; the second file nests a third. After the third file has finished, the rest of the second file is executed. After the second file has finished, the remaining part of the main file is executed.

The commands that nest are as follows:

```
{NEST}                ↻
{NEST MACRO}          ➤
{NEST PRIMARY}       ➤
{NEST SECONDARY}     ➤
```

Substituting is the process of permanently changing control of execution to another file. The substitution takes effect as soon as the command is encountered, and control is never returned to the file that contained this command.

The commands that substitute are as follows:

```
{SUBST PRIMARY}      ➤
{SUBST SECONDARY}   ➤
```

Commenting Out

You can use the {;} or {COMMENT} commands to *comment out* sections of a macro or merge that you don't want to execute. This practice is useful for testing and debugging your macros and merges.

Anything (including commands) between the {;} or {COMMENT} command and the next tilde is ignored in execution. It is easy to comment out commands such as {NEST}, {CHAIN}, etc., where there is only one tilde associated with the command:

```
{;}{NEST}thefile~
```

However, if the commands to be commented out have more than one tilde, you must insert a {;} or {COMMENT} to correspond to each one:

```
{COMMENT}{ASSIGN}Number~{COMMENT}45~
```

When multiple tildes are involved, you may find it easier to delete the additional tildes in the section to be commented out so that you only have to use one {;} or {COMMENT} command. The tildes would have to be re-inserted if you later decided to restore the section.

In Merge, a frequent use of {COMMENT} is to comment out Hard Return and Tab codes you use to format commands in the file. If you do not comment out these codes, they are included in the resulting merged document. See the *Merge* subheadings in this appendix for examples. The example under the {COMMENT} command may be especially useful.

If Blank, If Not Blank, and ?

In previous versions of WordPerfect, a question mark (?) was used to avoid merging a field if it was blank (e.g., ^F1?^). This system had certain drawbacks, such as that anything in the primary file after the question mark but on the same line was also not printed if the field was blank. The {IF BLANK} and {IF NOT BLANK} commands in WordPerfect 5.1 have eliminated this problem, plus added all the flexibility of a standard IF structure.

Although you can still use the question mark construction (represented as either {FIELD}*fieldname*? or ^F*fieldname*?^ in 5.1), you will probably find the {IF BLANK} and {IF NOT BLANK} commands more flexible.

In some cases, however, you may want to use the question mark as a shortcut for the {IF BLANK} and {IF NOT BLANK} commands. So, for example, if you had a list of addresses where you wanted to include the company name if it existed, but not if it didn't, you could use the question mark as a short cut method of handling the "company name line." For example:

```
{FIELD}name~  
{FIELD}company?~  
{FIELD}address~
```

In this example, if the company field is blank, no information is merged and the entire line is eliminated, moving the address up one line. To accomplish the same task with the {IF NOT BLANK} command, you could do the following:

```
{FIELD}name~  
{IF NOT BLANK}company~{COMMENT}  
~{FIELD}company~  
{END IF}{COMMENT}  
~{FIELD}address~
```

To insert the question mark, simply type it as the last character of the field name. If you are using numbered fields, type the question mark after the number.

Inserting Merge Commands During Execution

While an executing merge is paused at a {KEYBOARD} or {INPUT} command, you can execute the {QUIT}, {NEXT RECORD}, or {STOP} commands from the keyboard.

- 1 Press **Merge Codes** (Shift-F9).
- 2 Select **Quit** (1), **Next Record** (2), or **Stop** (3).

Selecting the command from this menu functions as if the command had been encountered in the primary file. See the description of each of these commands in this appendix for more information on their functions.

Levels

In Macros, you can have up to 30 levels of execution. Each {NEST} command uses 2 levels (one for executing the macro and one for a possible CHAIN command). Each {CALL}, {CASE CALL}, nested {IF}, {FOR}, {FOR EACH}, or {WHILE} command uses 1 level.

In Merge, you can have up to 20 levels of execution *per file*. {IF} commands do not require a level (they can be nested indefinitely). Each {CASE}, {CASE CALL}, {FOR}, or {WHILE} uses 1 level.

Execution levels are maintained in *stacks*. In general, Macros and Merge use separate stacks to maintain levels. However, all expressions use the Macros stack. Generally, expressions use one level, but may use more if they are very complex. However, since expressions use the Macros stack, they effectively use no levels in Merge.

In Merge, the levels for the nesting commands ({NEST MACRO}, {NEST PRIMARY}, or {NEST SECONDARY}) are maintained in a separate stack. They do not require an execution level (because for each file nested you have 20 new execution levels), but they do require a nesting level. Files can be nested 10 deep.

The levels used by these commands are released when the command ends normally. For example, the level used by a {CALL} command is released when a {RETURN} command ends the subroutine. The level used by the {IF} command is released when the {END IF} is executed, and the level used by the {FOR} command is released at the {END FOR} after the last iteration of the FOR loop. If any of these level-using structures are exited abnormally, the level is not released. For example, if you use a {GO} or {CASE} in the subroutine executed by a {CALL} command, the level corresponding to the {CALL} is never released. If you "lose" 28 levels in this way, the macro will terminate.

Loops

Whenever the same commands repeat several times, that section of the macro or merge is called a loop. For example,

```
{LABEL}Top`
    {;}Top-of-the-loop`
endless-loop
    {;}Type:"endless-loop"
```

```
{GO}Top`
      {;}Go-to-top`
```

In this example, the words “endless loop” are written continuously to the screen. There is no way to stop execution without pressing **Cancel** (F1), Ctrl-Break, etc. When you create a loop, it is very important to have a way for the loop to end.

In the following example, a count is kept of the number of times the text has been written to the screen. After the tenth time, the loop ends.

```
{ASSIGN}Counter`0`
      {;}Assign-0-to-var-Counter`
{LABEL}Top`
      {;}Top-of-the-loop`
      {ASSIGN}Counter`{ VARIABLE}Counter`+1`
      {;}Add-1-to-var-Counter`
Loop·{ VARIABLE}Counter`{Enter}
      {;}Type "loop"-#`
      {IF}{ VARIABLE}Counter`=10`
      {;}If-this-is-the-tenth-time`
      {QUIT}
      {;}Quit-the-macro`
      {ELSE}
      {;}Otherwise.`
      {GO}Top`
      {;}Go-to-top (repeat-the-loop)`
      {END IF}
      {;}End-of-{IF}-structure`
```

There are many types of loops you can create with Macros and Merge commands. You can use an IF structure as in the above examples, or you can use the {FOR}, {FOR EACH}, or {WHILE} commands (see each command above). You can also create loops by going to or calling subroutines (with the {GO} or {CALL} command). The structure you should use for any given loop will depend on the task you are trying to accomplish.

Message Display

The following commands send a message to the screen when executed:

| | | |
|-----------------|---|---|
| {CHAR} | Ⓞ | Y |
| {INPUT} | Ⓞ | Y |
| {PROMPT} | Ⓞ | Y |
| {STATUS PROMPT} | Ⓞ | Y |
| {TEXT} | Ⓞ | Y |

In Macros, you can use control characters to both position the message on the screen, and affect the display attributes (such as Bold, Mnemonics, etc.) of the message. For information on using control characters and display attributes, see *Macros, Message Display in Reference*.

In Merge, neither control characters nor display attributes are available when sending messages to the screen. However, there are techniques for affecting the display of messages:

Soft Wrapping

If you include a message that is longer than you have room for on the status line, it will automatically wrap to the next line. The first line of the message will *scroll* up to the next line. The message will continue to scroll until the entire message is displayed. (You can also use this method in Macros.)

Hard Wrapping

If you want to determine where a message will wrap (rather than letting it soft wrap), insert a hard return [HRt] at each point in the message where you want it to wrap.

Nesting a Macro

If you decide you need or want the positioning and attributes available for message display in Macros, you can put the command that sends the message in a nested macro (see *{NEST MACRO}* above).

See also *Prompting and User Input* below.

Previous Merge Commands

In previous versions of WordPerfect, some of the Merge programming commands were available, although they were represented using a caret (^) and a letter as a control character, instead of a command made up of a word or words in braces, as they are in WordPerfect 5.1. The following is a list of these command equivalents:

| Old Command | New Command |
|--------------------|--------------------|
| ^C | {KEYBOARD} |
| ^D | {DATE} |
| ^E | {END RECORD} |
| ^Fname^ or ^F#^ | {FIELD} |
| ^Gmacroname^G | {CHAIN MACRO} |
| ^N | {NEXT RECORD} |
| ^Omessage^O | {PROMPT} |
| ^Pfilename^P | {NEST PRIMARY} |
| ^Q | {QUIT} |
| ^R | {END FIELD} |
| ^Sfilename^S | {SUBST SECONDARY} |
| ^T | {PRINT} |
| ^U | {REWRITE} |
| ^V | {MRG CMND} |

If you have files that use the old code format, see *Previous Versions* under *Merge* in *Reference* for information on whether or not they will have to be converted for use with WordPerfect 5.1.

Prompting and User Input

The following commands can be used to prompt the user of your macro or merge, and/or obtain input from the user:

| | | |
|-----------------|----|----|
| {CHAR} | ☺ | ☺ |
| {INPUT} | ☺ | ☺☺ |
| {KEYBOARD} | | ☺☺ |
| {LOOK} | ☺ | ☺ |
| {PAUSE} | ☺☺ | |
| {PAUSE KEY} | ☺☺ | |
| {PROMPT} | ☺☺ | ☺ |
| {STATUS PROMPT} | ☺☺ | ☺☺ |
| {TEXT} | ☺ | ☺ |

These commands are similar to each other in some ways, yet different in others. The following chart shows some of these differences and similarities. Comparisons are based on the following features of each command:

- Whether or not a message is sent with the command or command combination.
- If a message is sent, whether the message remains on the screen until 1) the screen is rewritten or the message is overwritten with a new command, 2) input is terminated, or 3) you exit WordPerfect or the message is overwritten with a new command.
- Whether or not execution stops at the command or command combination for user input.
- If execution stops for input, whether the input goes directly into the document or into a variable.
- If execution stops for input, the method of terminating input.
- Whether the command or command combination is available in Macros only, Merge only, or in both features.

| | Sends Message? | Message Duration | Stops for user Input? | Input Goes To: | Input Termination method: | Available In: |
|----------------------------------|-----------------------|-------------------------|------------------------------|-----------------------|----------------------------------|----------------------|
| Commands | | | | | | |
| {CHAR} | Yes | R/O | Yes | Var. | 1 Char. | Both |
| {INPUT} | Yes | T | Yes | Doc. | Enter (Macros) F9 (Merge) | Both |
| {KEYBOARD} | No | n/a | Yes | Doc. | F9 | Merge |
| {LOOK} | No | n/a | No | Var. | 1 Char. or nothing | Both |
| {PAUSE} | No | n/a | Yes | Doc. | Enter | Macros |
| {PAUSE KEY} | No | n/a | Yes | Doc. | Specified key | Macros |
| {PROMPT} | Yes | R/O | No | n/a | n/a | Both |
| {STATUS PROMPT} | Yes | E/O | No | n/a | n/a | Both |
| {TEXT} | Yes | R/O | Yes | Var. | Enter | Both |
| Command Combinations | | | | | | |
| {PROMPT} with {KEYBOARD} | Yes | R/O | Yes | Doc. | F9 | Merge |
| *{PROMPT} with {PAUSE} | Yes | R/O | Yes | Doc. | Enter | Macros |
| *{PROMPT} with {PAUSE KEY} | Yes | R/O | Yes | Doc. | Specified key | Macros |
| {STATUS PROMPT} with {KEYBOARD} | Yes | E/O | Doc | F9 | Merge | |
| {STATUS PROMPT} with {PAUSE} | Yes | E/O | Yes | Doc. | Enter | Macros |
| {STATUS PROMPT} with {PAUSE KEY} | Yes | E/O | Yes | Doc. | Specified key | Macros |

Legend

n/a Not applicable
T Termination
R/O Rewrite Screen or Overwrite Message
E/O Exit WordPerfect or Overwrite Message

*Although {PROMPT} messages normally appear only until the screen is rewritten or the message is overwritten, the message will reappear even after these normal terminations if the {PROMPT} command is followed by a {CHAR} or {TEXT} command.

This occurs because the {CHAR} and {TEXT} commands take a "picture" of the status line, execute, and then redisplay the "picture."

Record Pointer

In a secondary merge file, there are several records. As a merge is executed, the primary file is repeated once for every record in the secondary file (unless you use commands to alter this procedure). The *record pointer* is the internal device that WordPerfect uses to keep track of which record it is currently using from the secondary file. (You cannot see the record pointer.)

At the start of a merge, the record pointer points to the first record in the secondary file. As the merge progresses, the pointer automatically moves to each subsequent record as the primary file is repeated. You can force the record pointer to move to the next record by using the {NEXT RECORD} command (see {NEXT RECORD} above). This is useful when you want to change to the next record *without* repeating the primary file.















Subroutines

A subroutine is a set of commands you may want to execute several times in a macro or merge. Instead of repeating the commands each time you need them, you can include them only once, then send execution to that spot each time you want the commands performed. Inserting a call to a subroutine functions as if its commands were placed at each point of the call. There is no limit to the number of subroutines you can have in a macro or merge file.

A subroutine is identified by two commands. The first command, {LABEL}, marks the beginning of a subroutine. The second command, {RETURN}, marks the end.

Some subroutines do not need a {RETURN} at the end, if the commands in the subroutine guarantee correct branching or returning.

The commands you can use to send execution to a subroutine are as follows:

| | | |
|----------------|---|---|
| {CALL} |  |  |
| {CASE} |  |  |
| {CASE CALL} |  |  |
| {GO} |  |  |
| {ON CANCEL} |  |  |
| {ON ERROR} |  |  |
| {ON NOT FOUND} |  |  |

Since there may be more than one subroutine in a macro, the name associated with each one must be unique. The label name must be entered as an argument in the command that sends execution to the subroutine, and must match the name following the {LABEL} command identifying the beginning of the subroutine.

If you use local variables in Merge, you cannot use the same name for both a subroutine and a local variable.

Troubleshooting

If you have trouble getting your macros or merges to work, check to see that you have not made one of the following errors:

- Incorrect number or placement of tildes. (See the descriptions of the commands you are using.)
- Using a variable name instead of the `{VARIABLE}var~` command, or vice versa. (See the description of the commands you are using, and *Appendix L: Macros and Merge, Variables.*)
- Other syntax errors. (See the descriptions of the commands you are using.)
- Infinite loops. (See *Loops* above.)
- Trying to reference a global variable while a local variable of the same name is accessible. (See *Appendix L: Macros and Merge, Variables.*)
- Using a local variable and label of the same name. (See `{LOCAL}` and `{LABEL}` above.)
- In a merge, accessing a global variable that contains keystrokes. (See *Keystroke Commands in Variables* in *Appendix L: Macros and Merge, Variables.*)
- Missing a `{RETURN}` at the end of a subroutine. (See *Subroutines* above.)
- Misspelled variable or label names.
- Going to or calling non-existent labels, or accessing non-existent variables.
- In a merge, incorrect placement of `{COMMENT}` command and its tilde, or lack of use of the `{COMMENT}` command. (See `{COMMENT}` and *Commenting Out* above.)
- Nesting files too many deep or using too many levels. (See *Levels* above.)
- Performing a numeric operation on a string. This usually happens when you use an invalid character (such as a space, period, or comma) in an expression which is assigned to a variable. (See *Appendix J: Macros and Merge, Expressions.*)
- Missing loop terminators (see `{END FOR}`, `{END IF}`, `{END WHILE}`).
- Missing or improper nested files, or improper termination of nested files (see `{NEST}` `{NEST MACRO}`, `{NEST PRIMARY}`, `{NEST SECONDARY}`, and *Chaining, Nesting, and Substituting* above).
- In a merge, using an open code (such as a Tab Set or Margin Set) in a primary file that is not reset at the beginning of the file. (Any open code encountered during the merge will be in effect in the resulting merged document until another open code of the same type is found to counteract it.)

See Also: Lessons 23 through 25; Macros; Macros, Define; Macros, Execute; Macros, Macro Editor; Macros, Message Display; Merge; Appendix I; Appendix J; Appendix L

Appendix L: Macros and Merge, Variables

A variable represents a place in memory where data is stored. As its name indicates, the data in a variable is changeable. You might want to use variables to calculate and/or keep track of values and text which change during execution.

System Variables vs. User-Defined Variables

There are two major types of variables in WordPerfect: *system variables* and *user-defined variables*. System variables are variables that WordPerfect creates and maintains, and that contain information about the current state of WordPerfect. You cannot change the names or contents of these variables, but you can find out and use their contents at any given time. The names and possible contents of system variables are listed in *Appendix K: Macros and Merge, Programming Commands* under the description of the {SYSTEM} command.

User-defined variables are variables that *you* create and name, and whose contents *you* determine. You can perform operations on these variables to change their contents. There are two sub-categories of user-defined variables: *global* variables and *local* variables.

Global vs. Local Variables

Global variables are accessible from anywhere inside a macro or merge.

Local variables are available only in Merge, and are stored in a separate place in memory from global variables. They are only accessible from the file in which they are created. For example, if you create a local variable named Number in a primary file, the secondary files or other primary files cannot access the information stored in it. When you nest merge files, the local variables in the parent file are not visible to the nested file. When you return to the parent file, the local variables in the nested file no longer exist, but the local variables in the parent file are once again accessible.

Local variables take precedence over global variables. For example, suppose you have both a global and local variable named Number. If you try to access the global variable Number from inside a file where the local variable Number is accessible, you will get the contents of the local variable. The global variable Number still exists, but is inaccessible until execution moves to where the local variable Number is no longer accessible (out of the merge file).

Naming Variables

Variables, whether local or global, must have a unique name by which you refer to them. The name may consist of any combination of the characters in the WordPerfect character sets. However, in Macros, only the first 7 letters are used to determine uniqueness. So, in Macros, ABCDEFG and ABCDEFGH are considered by WordPerfect to refer to the same variable. In Merge, however, 15 characters are used to determine uniqueness. In Merge, ABCDEFG (7 characters long) and ABCDEFGH (8 characters long) would be considered two separate,

unique variables, but ABCDEFGHIJKLMNO (15 characters long) and ABCDEFGHIJKLMNOP (16 characters long) would be considered the same variable.

Variable names are not case sensitive. *Abc*, *AbC*, *ABC*, *abc*, are all considered by WordPerfect to be the same variable.

Variables receive their names when they are *assigned*. See *Assigning Variables* below for more information.

Variable Contents

All user-defined variables can contain text or numbers. In addition to text and numbers, global variables can also contain keystrokes. The method you use to assign variables may affect the kinds of codes, commands, and keystrokes that can be assigned to a variable (see *Assigning Variables* and *Keystroke Commands in Variables* below). A user-defined variable can only hold 129 keystrokes. A keystroke can be a character, an extended character, a keystroke command (in Macros), or a programming command.

Assigning Variables

You assign a global variable with the {ASSIGN} command, and a local variable with the {LOCAL} command. For example, the following two statements assign two different variables, one global, and one local:

```
{ASSIGN}Number~45~  
{LOCAL}Number~36~
```

The {ASSIGN} command creates a global variable named *Number* and puts in “45” as its contents. The {LOCAL} command creates a separate, local variable named *Number* and puts in “36” as its contents. (See the description of the {ASSIGN} and {LOCAL} commands in *Appendix K: Macros and Merge, Programming Commands*.)

In a macro, the {ASSIGN} command can only be entered from within the Macro Editor (see *Inserting Commands under Macros, Macro Editor in Reference*). In a merge file, the {ASSIGN} and {LOCAL} commands are inserted from the normal editing screen by pressing **Merge Codes** (Shift-F9) twice, then selecting the command from the command access box (see *Inserting Merge Commands under Merge in Reference*).

In addition to using the {ASSIGN} and {LOCAL} commands, the following commands also assign variables:

```
{CHAR}           {LOOK}  
{FOR}           {SHELL ASSIGN}  
{FOR EACH}     {TEXT}
```

See *Appendix K: Macros and Merge, Programming Commands* for a description of each of these commands.

The following rules determine whether the variable assigned by the {CHAR}, {LOOK}, and {TEXT} commands are local or global in a merge:

- If a local variable by the name used in the command exists, the command will assign to the local variable.
- If no local variable by the name used in the command exists, but a global variable by that name does exist, the command will assign to the global variable.
- If no variable exists by the name used in the command, a global variable of that name is created and assigned by the command.

The {FOR} command in Merge assigns by the same rules as above, except that if no variable exists by the name used in the command (the third rule), a *local* variable is created and assigned by the command. This feature allows recursion using the {FOR} command in Merge.

You can also assign global variables without using a command from the normal editing screen or while in macro definition mode (see *Macros, Define* in *Reference* for an explanation of macro definition mode). To assign the variable,

- 1** Press **Macro Commands** (Ctrl-PgUp).
- 2** If you are in macro definition mode, a menu appears. Select **Assign (3)**. Otherwise, skip to the next step.
- 3** Enter the variable name.

You are prompted for the value.

- 4** Enter the variable contents.

Using this method, you can only assign characters and numbers to the variable. You can enter up to 79 characters. If the characters you enter form a valid expression (see *Appendix J: Macros and Merge, Expressions*), the expression is evaluated and the result is assigned to the variable.

To assign a block of existing text to the variable (from the normal editing screen or while in macro definition mode),

- 1** Block the text (Alt-F4).
- 2** Press **Macro Commands** (Ctrl-PgUp).
- 3** If you are in macro definition mode, a menu appears. Select **Assign (3)**. Otherwise, skip to the next step.
- 4** Enter the variable name.

The first 128 characters are assigned to the variable.

When codes are encountered in the block, they are converted (if possible) to the keystroke command that would normally insert that code. For example, if the block includes a [Tab] code, it is converted to the keystroke command {Tab}.

The codes that can be converted are:

| Code | Converted To |
|------------------------|--------------|
| [♦Indent] | { Indent } |
| [♦Indent♦] | { Indent } |
| [Tab] | { Tab } |
| [Center] | { Tab } |
| [Flsh Rgt] | { Tab } |
| [♦Mar Rel] | { Tab } |
| [–] (Hyphen character) | – (dash) |
| – (Hard Hyphen) | – (dash) |
| [HRt] | { Enter } |
| [Hrt–Spg] | { Enter } |
| [Dorm HRt] | { Enter } |
| [SRt] | (space) |
| [SPg] | (space) |
| [HPg] | { HPg } |

See *Keystroke Commands in Variables* below for information on limitations to using variables that contain keystroke commands.

If a variable already exists, assigning new contents to it replaces the previous contents without warning.

Executing Variables

You can *execute* or *write out* a variable anywhere you would want its contents. For example, by executing a variable you can do the following:

- Use the contents of the variable as a subroutine.
- Insert the contents as text in a document, or in the message strings of programming commands.
- Provide variable arguments in other programming commands.

To execute a variable, you use the {VARIABLE} command. For example, the statement {VARIABLE}Number~ would execute the variable named Number.

If a global variable is named with a single-digit number (1, 2, 3, 4, 5, 6, 7, 8, 9, or 0) (see *Naming Variables* above), the Macro Editor allows a short-cut method of inserting the command to execute the variable in the macro. You can press **Ctrl-v,Alt-#**, where # is the number of the variable, or if you have pressed **Macro Define** (Ctrl-F10) to turn on command insert mode, just press **Alt-#**. The command that is inserted looks like this: {VAR #}. This command is equivalent to {VARIABLE}#~. Remember, however, that this shortcut is only available in the Macro Editor.

Another advantage of naming global variables with a single digit is that you can execute them at the normal editing screen. For example, if you are at the normal editing screen and want to know the current contents of variable 5, press **Alt-5**.

The contents are executed/written out. You cannot use this method to execute variables of other names at the normal editing screen.

Variable Duration

Local variables exist only in the file in which they are created. Once the file that created the variables is exited or the merge ends, the local variables are erased (and the memory assigned to them is released). However, the contents of global variables remain in memory until you exit WordPerfect. To conserve memory, local variables instead of global variables whenever possible.

If you want to erase a variable before WordPerfect does it for you, assign the variable “nothing” by using the following commands: `{ASSIGN}var~` or `{LOCAL}var~`. These commands not only empty the variable of its contents, but also release the memory used by the variable. After this command, the variable no longer exists. It is a good idea to empty variables at the beginning of a macro or merge in which they are used (unless the macro or merge assigns new contents to them).

Operations on Variables

All variables can be compared to each other, and user-defined variables can have other operations performed on them. Operations are performed using various programming commands (see *Appendix K: Macros and Merge, Programming Commands* and *Appendix J: Macros and Merge, Expressions*).

Keystroke Commands in Variables

In Macros, variables are executed as keystrokes. Therefore, if in a macro you assign keystroke commands (such as `{Search}`, `{Tab}`, `{Enter}`, `{Home}`) to a variable, then execute the variable, the keystrokes will be performed.

Keystroke commands do not exist in Merge, so it is impossible to assign keystroke commands to a variable in a merge.* (You also cannot enter keystroke commands in the merge files themselves.)

However, since all variables assigned in macros are global, it is possible to execute within a merge a global variable that was assigned keystroke commands by a macro. If you do so, the keystrokes will not be executed as they would be in a macro. Instead, the character equivalents of the keystroke commands will be written out to the document.* If you are combining macros and merges and find unexpected control characters (such as `^I`, `^F`, `^H`) or extended characters in the merged document, see that you have assigned your variables properly.

There is one exception to this rule. When you use a `{CHAR}` command in a merge, the user can press **Enter at the prompt to assign the variable the keystroke command `{Enter}`. If you then execute the variable, a `{HRt}` is inserted in the resulting document. See `{CHAR}` in Appendix K: Macros and Merge, Programming Commands for more information on using the `{CHAR}` command.*

See Also: Macros; Macros, Define; Macros, Execute; Macros, Macro Editor; Macros, Message Display; Merge; Appendix I; Appendix J; Appendix K

Appendix T: Macros and Merge, Value Tables

This appendix lists the values returned by various macro and merge programming commands.

{KTON} and {NTOK}

The following table shows the WordPerfect key values used with the {KTON} and {NTOK} programming commands. For more information on using these commands, see *Appendix K: Macros and Merge, Programming Commands*.

| Value | Key Name | Key | Macro Command |
|-------|-----------------------------|---------------------|-------------------------------|
| 32768 | Compose | Ctrl-2 | {Compose} ¹ |
| 32769 | ^A | Ctrl-a | {^A} |
| 32770 | ^B | Ctrl-b | {^B} |
| 32771 | ^C | Ctrl-c | {^C} |
| 32772 | ^D | Ctrl-d | {^D} |
| 32773 | ^E | Ctrl-e | {^E} |
| 32774 | ^F | Ctrl-f | {^F} |
| 32775 | ^G | Ctrl-g | {^G} |
| 32776 | Home or ^H | Home or Ctrl-h | {Home} |
| 32777 | Tab or ^I | Tab | {Tab} |
| 32778 | Enter or ^J | Enter or Ctrl-j | {Enter} |
| 32779 | Delete to End of Line or ^K | Ctrl-End or Ctrl-K | {Del EOL} |
| 32780 | Delete to End of Page or ^L | Ctrl-PgDn or Ctrl-l | {Del EOP} |
| 32781 | ^M | Ctrl-m | {^M} |
| 32782 | ^N | Ctrl-n | {^N} |
| 32783 | ^O | Ctrl-o | {^O} |
| 32784 | ^P | Ctrl-p | {^P} |
| 32785 | ^Q | Ctrl-q | {^Q} |
| 32786 | ^R | Ctrl-r | {^R} |
| 32787 | ^S | Ctrl-s | {^S} |
| 32788 | ^T | Ctrl-t | {^T} |
| 32789 | ^U | Ctrl-u | {^U} |
| 32790 | ^V | Ctrl-v | {^V} |
| 32791 | Up Arrow or ^W | ↑ or Ctrl-w | {Up} |
| 32792 | Right Arrow or ^X | → or Ctrl-x | {Right} |
| 32793 | Left Arrow or ^Y | ← or Ctrl-y | {Left} |
| 32794 | Down Arrow or ^Z | ↓ or Ctrl-z | {Down} |
| 32795 | Escape or ^[| Esc or Ctrl-[| {Esc} |
| 32796 | ^\ ^] | Ctrl-\ Ctrl-] | {^\ {^]} |
| 32797 | ^ ^^ | Ctrl- Ctrl-^ | {^ {Keyboard} ¹ |
| 32799 | ^_ | Ctrl-_ | {^_} |
| 32800 | Cancel | F1 | {Cancel} |
| 32801 | ◆Search | F2 | {Search} |
| 32802 | Help | F3 | {Help} |
| 32803 | Indent | F4 | {Indent} |
| 32804 | List | F5 | {List} |
| 32805 | Bold | F6 | {Bold} |
| 32806 | Exit | F7 | {Exit} |
| 32807 | Underline | F8 | {Underline} |
| 32808 | End Field ² | F9 | {End Field} |
| 32809 | Save | F10 | {Save} |
| 32810 | Reveal Codes | F11 ³ | {Reveal Codes} |
| 32811 | Block | F12 ² | {Block} |
| 32812 | Setup | Shift-F1 | {Setup} |
| 32813 | ◆Search | Shift-F2 | {Search Left} |
| 32814 | Switch | Shift-F3 | {Switch} |
| 32815 | ◆Indent◆ | Shift-F4 | {L/R Indent} |
| 32816 | Date/Outline | Shift-F5 | {Date/Outline} |
| 32817 | Center | Shift-F6 | {Center} |

| Value | Key Name | Key | Macro Command |
|-------|-------------------|--|-------------------------|
| 32818 | Print | Shift-F7 | {Print} |
| 32819 | Format | Shift-F8 | {Format} |
| 32820 | Merge Codes | Shift-F9 | {Merge Codes} |
| 32821 | Retrieve | Shift-F10 | {Retrieve} |
| 32822 | | Shift-F11 ¹ | {Shft F11} ¹ |
| 32823 | | Shift-F12 ⁴ | {Shft F12} ¹ |
| 32824 | Thesaurus | Alt-F1 | {Thesaurus} |
| 32825 | Replace | Alt-F2 | {Replace} |
| 32826 | Reveal Codes | Alt-F3 | {Reveal Codes} |
| 32827 | Block | Alt-F4 | {Block} |
| 32828 | Mark Text | Alt-F5 | {Mark Text} |
| 32829 | Flush Right | Alt-F6 | {Flush Right} |
| 32830 | Columns/Table | Alt-F7 | {Columns/Tables} |
| 32831 | Style | Alt-F8 | {Style} |
| 32832 | Graphics | Alt-F9 | {Graphics} |
| 32833 | Macro | Alt-F10 | {Macro} |
| 32834 | | Alt-F11 ⁴ | {Alt F11} ¹ |
| 32835 | | Alt-F12 ⁴ | {Alt F12} ¹ |
| 32836 | Shell | Ctrl-F1 | {Shell} |
| 32837 | Spell | Ctrl-F2 | {Spell} |
| 32838 | Screen | Ctrl-F3 | {Screen} |
| 32839 | Move | Ctrl-F4 | {Move} |
| 32840 | Text In/Out | Ctrl-F5 | {Text In/Out} |
| 32841 | Tab Align | Ctrl-F6 | {Tab Align} |
| 32842 | Footnote | Ctrl-F7 | {Footnote} |
| 32843 | Font | Ctrl-F8 | {Font} |
| 32844 | Merge/Sort | Ctrl-F9 | {Merge/Sort} |
| 32845 | Macro Define | Ctrl-F10 | {Macro Define} |
| 32846 | | Ctrl-F11 ⁴ | {Ctrl F11} ¹ |
| 32847 | | Ctrl-F12 ⁴ | {Ctrl F12} ¹ |
| 32848 | Backspace | Backspace | {Backspace} |
| 32849 | Delete | Del | {Del} |
| 32850 | Delete Word | Ctrl-Backspace | {Del Word} |
| 32851 | Word Right | Ctrl-→ | {Word Right} |
| 32852 | Word Left | Ctrl-← | {Word Left} |
| 32853 | End of Line | End or Home, Home, → ⁵ | {End} |
| 32854 | Beginning of Line | Begin (Victor computer) or Home, Home, ← ⁵ | |
| 32855 | | | {Invalid} |
| 32856 | Go To | Ctrl-Home | {Goto} |
| 32857 | Page Up | PgUp | {Page Up} |
| 32858 | Page Down | PgDn | {Page Down} |
| 32859 | Screen Down | + | {Screen Down} |
| 32860 | Screen Up | - | {Screen Up} |
| 32861 | Typeover | Ins | {Typeover} |
| 32862 | Margin Release | Shift-Tab | {Left Mar Rel} |
| 32863 | Hard Page | Ctrl-Enter | {HPg} |
| 32864 | Soft Hyphen | Ctrl-~ | {SHy} |
| 32865 | Hyphen Character | Alt-~ | {-} |
| 32866 | Hard Space | Home, Space Bar ⁵ | { } ¹ |
| 32867 | Paragraph Up | Ctrl-↑ | {Para Up} |
| 32868 | Paragraph Down | Ctrl-↓ | {Para Down} |
| 32869 | Item Left | Alt-← | {Item Left} |
| 32870 | Item Right | Alt-→ | {Item Right} |
| 32871 | Item Up | Alt-↑ | {Item Up} |
| 32872 | Item Down | Alt-↓ | {Item Down} |
| 32873 | | Alt-Home ⁴ | {Alt Home} |
| 32874 | Delete Row | Ctrl-Del | {Block Move} |
| 32875 | Menu Bar | Alt-= | {Menu Bar} ¹ |
| 32876 | Block Append | | {Block Append} |
| 32877 | Block Move | Ctrl-Del | {Block Move} |
| 32878 | Block Copy | Ctrl-Ins | {Block Copy} |
| 64513 | | | {ASSIGN} |

| Value | Key Name | Key | Macro Command |
|-------|----------------|-----------|-----------------------|
| 64514 | | | {BELL} |
| 64515 | | | {BREAK} |
| 64516 | | | {CALL} |
| 64517 | | | {CANCEL OFF} |
| 64518 | | | {CANCEL ON} |
| 64519 | | | {CASE} |
| 64520 | | | {CASE CALL} |
| 64521 | | | {CHAIN} |
| 64522 | | | {CHAR} |
| 64523 | | | {:} |
| 64524 | | | {DISPLAY OFF} |
| 64525 | | | {DISPLAY ON} |
| 64526 | | | {ELSE} |
| 64527 | | | {END FOR} |
| 64528 | | | {END IF} |
| 64529 | | | {END WHILE} |
| 64530 | | | {FOR} |
| 64531 | | | {FOR EACH} |
| 64532 | | | {GO} |
| 64533 | | | {IF} |
| 64534 | | | {LABEL} |
| 64535 | | | {LOOK} |
| 64536 | | | {NEST} |
| 64537 | | | {NEXT} |
| 64538 | | | {SHELL MACRO} |
| 64539 | | | {ON CANCEL} |
| 64540 | | | {ON ERROR} |
| 64541 | | | {ON NOT FOUND} |
| 64542 | | | {PAUSE} |
| 64543 | | | {PROMPT} |
| 64544 | | | {QUIT} |
| 64545 | | | {RESTART} |
| 64546 | | | {RETURN} |
| 64547 | | | {RETURN CANCEL} |
| 64548 | | | {RETURN ERROR} |
| 64549 | | | {RETURN NOT FOUND} |
| 64550 | | | {SPEED} |
| 64551 | | | {STEP ON} |
| 64552 | | | {TEXT} |
| 64553 | | | {STATE} |
| 64554 | | | {WAIT} |
| 64555 | | | {WHILE} |
| 64556 | Macro Commands | Ctrl-PgUp | {Macro Commands} |
| 64557 | | | {STEP OFF} |
| 64558 | | | {ORIGINAL KEY} |
| 64559 | | | {IF EXISTS} |
| 64560 | | | {MENU OFF} |
| 64561 | | | {MENU ON} |
| 64562 | | | {STATUS PROMPT} |
| 64563 | | | {INPUT} |
| 64564 | | | {VARIABLE} |
| 64565 | | | {SYSTEM} |
| 64566 | | | {MID} |
| 64567 | | | {NTOK} |
| 64568 | | | {KTON} |
| 64569 | | | {LEN} |
| 64570 | | | {~} |
| 64571 | | | {PAUSE KEY} |
| 64573 | | | {OTHERWISE} |
| 64574 | | | {SHELL ASSIGN} |
| 64575 | | | {SHELL VARIABLE} |

¹To insert this command in a macro, you must define a macro containing `{NTOK}x`, where `x` is the value listed on the table. Then map this macro to a key using *Keyboard Layout* (see *Keyboard Layout in Reference*). With the keyboard containing the mapping selected, enter the Macro Editor. Press the key to which the macro is mapped to insert the command.

²Pressing **End Field** (F9) inserts “[Mrg.End Field][HRt]” into the document.

³F11 and F12 are remapped on the original keyboard to correspond to Alt-F3 and Alt-F4, respectively.

⁴This key is not mapped on the original keyboard.

⁵Although it is impossible to take the `{KTON}` of multiple keys, you can produce (execute) this keystroke sequence by taking the `{NTOK}` of its key value.

{SYSTEM}Right[~]
{SYSTEM}Left[~]

The following table shows the values returned by the `{SYSTEM}Right~` and `{SYSTEM}Left~` commands when the item to the right or left of the cursor is a WordPerfect code. In most cases, a five-digit number is returned. For some codes, however, the keystroke corresponding to the code is returned instead of a number. These codes are listed at the end of the table.

For some codes, the value returned differs depending on whether it was returned by the `{SYSTEM}Right~` or `{SYSTEM}Left~` command. These values are marked with (R) and (L), respectively, in the table below.

For more information on using these commands, see *Appendix K: Macros and Merge, Programming Commands*.

| Value | Code |
|----------------------|------------------------------|
| 33024 | [Just On] |
| 33280 | [Just Off] |
| 34304 | [Center Pg] |
| 34560 | [Col On] |
| 34816 | [Col Off] preceded by [HPg] |
| 35328 | [W/O On] |
| 35584 | [W/O Off] |
| 35840 | [HRt-SPg] |
| 36096 | [Note Num] |
| 36352 | [Box Num] |
| 36608 | [End C/A] |
| 36864 | [DSRt] |
| 37376 | [SPg] (not a space or [HRt]) |
| 37632 | [ISRt] |
| 38656 (R), 38400 (L) | [Block] |
| 39168 | [Dorm HRt] |
| 39424 | [/] (Cancel Hyphenation) |
| 39680 | [End Del] |
| 40448 | [Hyph Off] |
| 40704 | [Hyph On] |
| 40960 | [] (Hard Space) |
| 41216 | [+] |
| 41472 | [!] |
| 41728 | [=] |
| 41984 | [T] |
| 42240 | [*] |
| 42496 | [!] |
| 42752 | [Math On] |

| Value | Code |
|-------|---|
| 43008 | [Math Off] |
| 43264 | [_] (Hyphen Character) |
| 44032 | - (Soft Hyphen) |
| 44800 | [Col Off] if created by [HPg] but not preceded by [HPg] |
| 45056 | [Col Off] if created by [SPg] but not preceded by [HPg] |
| 45312 | [N] |
| 45568 | [Outline Off] |
| 49408 | [TAB] |
| 49410 | [Tab] |
| 49424 | [TAB] (with dot leader) |
| 49426 | [Tab] (with dot leader) |
| 49474 | [Dec Tab] |
| 49480 | [RGT TAB] |
| 49482 | [Rgt Tab] |
| 49490 | [Dec Tab] (with dot leader) |
| 49496 | [RGT TAB] (with dot leader) |
| 49498 | [Rgt Tab] (with dot leader) |
| 49504 | [Fish Rgt] |
| 49520 | [Fish Rgt] (with dot leader) |
| 49536 | [Mar Rel] |
| 49608 | [CNTR TAB] |
| 49610 | [Cntr Tab] |
| 49624 | [CNTR TAB] (with dot leader) |
| 49626 | [Cntr Tab] (with dot leader) |
| 49632 | [Center] |
| 49648 | [Center] (with dot leader) |
| 49664 | [→Indent] |
| 49665 | [→Indent←-] |
| 49920 | [EXT LARGE] |
| 49921 | [VRY LARGE] |
| 49922 | [LARGE] |
| 49923 | [SMALL] |
| 49924 | [FINE] |
| 49925 | [SUPRSCPT] |
| 49926 | [SUBSCPT] |
| 49927 | [OUTLN] |
| 49928 | [ITALC] |
| 49929 | [SHADW] |
| 49930 | [REDLN] |
| 49931 | [DBL UND] |
| 49932 | [BOLD] |
| 49933 | [STKOUT] |
| 49934 | [UND] |
| 49935 | [SM CAP] |
| 50176 | [ext large] |
| 50177 | [vry large] |
| 50178 | [large] |
| 50179 | [small] |
| 50180 | [fine] |
| 50181 | [suprscpt] |
| 50182 | [subscpt] |
| 50183 | [outln] |
| 50184 | [italic] |
| 50185 | [shadw] |
| 50186 | [redln] |
| 50187 | [dbl und] |
| 50188 | [bold] |
| 50189 | [stkout] |
| 50190 | [und] |
| 50191 | [sm cap] |
| 50432 | [Block Pro:Cn] |
| 50433 | [Block Pro:Off] |
| 53248 | [Ln Height] |

| Value | Code |
|-------|---------------------|
| 53249 | [L/R Mar] |
| 53250 | [Ln Spacing] |
| 53251 | [HZone] |
| 53252 | [Tab Set] |
| 53253 | [T/B Mar] |
| 53254 | [Just] |
| 53255 | [Suppress] |
| 53256 | [Pg Numbering] |
| 53259 | [Paper Sz/Typ] |
| 53504 | [Color] |
| 53505 | [Font] |
| 53760 | [Math Def] |
| 53761 | [Col Def] |
| 53762 | [Par Num Def] |
| 53763 | [Ftn Opt] |
| 53764 | [End Opt] |
| 53765 | [Fig Opt] |
| 53767 | [Txt Opt] |
| 53768 | [Usr Opt] |
| 53769 | [Equ Opt] |
| 53771 | [Tbi Def] |
| 53773 | [Link] |
| 53774 | [Link End] |
| 53776 | [Tbi Opt] |
| 53777 | [Brdr Opt] |
| 54016 | [Decml/Algn Char] |
| 54017 | [Undrin] |
| 54018 | [New Ftn Num] |
| 54019 | [New End Num] |
| 54020 | [Pg Num] |
| 54021 | [Ln Num] |
| 54022 | [Acv] |
| 54023 | [Force] |
| 54024 | [BLine] |
| 54026 | [Wrld/Ltr Spacing] |
| 54027 | [Just Lim] |
| 54028 | [New Fig Num] |
| 54029 | [New Tbi Num] |
| 54030 | [New Txt Num] |
| 54031 | [New Usr Num] |
| 54032 | [New Equ Num] |
| 54033 | [Lang] |
| 54034 | [Pg Num Style] |
| 54528 | [Header A] |
| 54529 | [Header B] |
| 54530 | [Footer A] |
| 54531 | [Footer B] |
| 54784 | [Footnote] |
| 54785 | [Endnote] |
| 55040 | [Mark] |
| 55041 | [End Mark] |
| 55042 | [Def Mark] |
| 55043 | [Index] |
| 55044 | [ToA] |
| 55045 | [Endnote Placement] |
| 55047 | [Ref] |
| 55048 | [Target] |
| 55049 | [Subdoc] |
| 55050 | [Subdoc Start] |
| 55051 | [Subdoc End] |
| 55296 | [Date] |
| 55297 | [Par Num] |
| 55298 | [Ovrstk] |

| Value | Code |
|----------------------|--|
| 55299 | [Insert Pg Num] |
| 55552 | [Ptr Crnd] |
| 55553 | [Cnd] EOP] |
| 55554 | [Comment] |
| 55555 | [Kern] |
| 55556 | [Outline On] |
| 55557 | [Leading Adj] |
| 55808 | [Fig Box] |
| 55809 | [Tbl Box] |
| 55810 | [Text Box] |
| 55811 | [Usr Box] |
| 55812 | [Equ Box] |
| 55813 | [HLine] |
| 55814 | [VLine] |
| 56064 (R), 56067 (L) | [Style:On] or [Outline Lvl # Style On] |
| 56065 (R), 56067 (L) | [Style:Off] or [Outline Lvl # Style Off] |
| 56066 (R), 56067 (L) | [Open Style] or [Outline Lvl # Open Style] |
| 56320 | [Cell] |
| 56321 | [Row] |
| 56322 | [Tbl Off] |
| 56579 | [Hrd Row] |
| 56864 | [Mrg:ASSIGN] |
| 56865 | [Mrg:BELL] |
| 56866 | [Mrg:BREAK] |
| 56867 | [Mrg:CALL] |
| 56868 | [Mrg:CANCEL OFF] |
| 56869 | [Mrg:CANCEL ON] |
| 56870 | [Mrg:CASE] |
| 56871 | [Mrg CASE CALL] |
| 56872 | [Mrg:CHAIN MACRO] |
| 56873 | [Mrg:CHAIN PRIMARY] |
| 56874 | [Mrg:CHAIN SECONDARY] |
| 56875 | [Mrg:CHAR] |
| 56876 | [Mrg:COMMENT] |
| 56877 | [Mrg:CTON] |
| 56878 | [Mrg:DATE] |
| 56879 | [Mrg DOCUMENT] |
| 56880 | [Mrg:ELSE] |
| 56881 | [Mrg:END FIELD] |
| 56882 | [Mrg:END FOR] |
| 56883 | [Mrg:END IF] |
| 56884 | [Mrg:END RECORD] |
| 56885 | [Mrg:END WHILE] |
| 56886 | [Mrg:FIELD] |
| 56887 | [Mrg:FOR] |
| 56889 | [Mrg:GO] |
| 56890 | [Mrg:IF] |
| 56891 | [Mrg:IF BLANK] |
| 56892 | [Mrg:IF EXISTS] |
| 56893 | [Mrg:IF NOT BLANK] |
| 56894 | [Mrg:KEYBOARD] |
| 56895 | [Mrg:LABEL] |
| 56896 | [Mrg:LOCAL] |
| 56897 | [Mrg:LOOK] |
| 56898 | [Mrg:MID] |
| 56899 | [Mrg:MRG CMND] |
| 56900 | [Mrg:NEST MACRO] |
| 56901 | [Mrg:NEST PRIMARY] |
| 56902 | [Mrg:NEST SECONDARY] |
| 56903 | [Mrg:NEXT] |
| 56904 | [Mrg:NEXT RECORD] |
| 56905 | [Mrg:NTOC] |
| 56906 | [Mrg:PROCESS] |

| Value | Code |
|---------|--------------------------|
| 56907 | [Mrg:ON CANCEL] |
| 56908 | [Mrg:ON ERROR] |
| 56909 | [Mrg:PAGE OFF] |
| 56910 | [Mrg:PAGE ON] |
| 56911 | [Mrg:PRINT] |
| 56912 | [Mrg:PROMPT] |
| 56913 | [Mrg:QUIT] |
| 56914 | [Mrg:RETURN] |
| 56915 | [Mrg:RETURN CANCEL] |
| 56916 | [Mrg:RETURN ERROR] |
| 56917 | [Mrg:REWRITE] |
| 56918 | [Mrg:STEP OFF] |
| 56919 | [Mrg:STEP ON] |
| 56920 | [Mrg:SUBST PRIMARY] |
| 56921 | [Mrg:SUBST SECONDARY] |
| 56922 | [Mrg:SYSTEM] |
| 56923 | [Mrg:TEXT] |
| 56924 | [Mrg:VARIABLE] |
| 56925 | [Mrg:WAIT] |
| 56926 | [Mrg:WHILE] |
| 56927 | [Mrg:STATUS PROMPT] |
| 56928 | [Mrg:INPUT] |
| 56929 | [Mrg:LEN] |
| 56930 | [Mrg:FIELD NAME] |
| 56931 | [Mrg:STOP] |
| {^A} | ^A |
| {^B} | ^B |
| {^C} | ^C |
| {^D} | ^D |
| {^E} | ^E |
| {^F} | ^F |
| {^G} | ^G |
| {Home} | ^H |
| {Enter} | [HRt] |
| {^N} | ^N |
| {^O} | ^O |
| {^P} | ^P |
| {^Q} | ^Q |
| {^R} | ^R |
| {^S} | ^S |
| {^T} | ^T |
| {^U} | ^U |
| {^V} | ^V |
| {Up} | ^W |
| {Right} | ^X |
| {Left} | ^Y |
| {Down} | ^Z |
| {Esc} | ^[|
| {^} | ^\ ^ ^ ^^ |
| {^} | ^ ^ ^ ^^ |
| {^^} | ^^ |
| {SHy} | - (Soft Hyphen, Ctrl- -) |

{SYSTEM}Menu~

The tables below list the values returned by the {SYSTEM}Menu~ command. The first table lists the values by menu, the second by number. The Keystrokes column of both tables lists the keystrokes required to produce the menu. Bracketed keystrokes indicate that any one of the given keys may be pressed at that point in the keystroke series to produce the menu. Other conditions required to produce the menu appear in the Conditions column. Block is assumed to be off unless otherwise specified.

For more information on using the {SYSTEM}Menu~ command, see *Appendix K: Macros and Merge, Programming Commands*.

By Menu

This table lists the menus by general menu name and then by the keystrokes required to produce the menu. Use this table when you know the menu but not the value that will be returned at that menu.

| Value | Keystrokes | Conditions |
|--------------------------------|--------------------------------|---------------------------------|
| Cancel Menu (F1) | | |
| 49 | F1 | |
| Setup Menu (Shift-F1) | | |
| 177 | Shift-F1 | |
| 346 | Shift-F1,1 | |
| 363 | Shift-F1,1,2 | |
| 221 | Shift-F1,2 | |
| 221 | Shift-F1,2,1 | monochrome graphics card |
| 190 | Shift-F1,2,1 | Hercules RAM Font graphics card |
| 180 | Shift-F1,2,1 | CGA graphics card |
| 187 | Shift-F1,2,1 | EGA,VGA graphics card |
| 221 | Shift-F1,2,[2,3] | |
| 341 | Shift-F1,2,4 | |
| 341 | Shift-F1,2,4,[1,2,3,5,6],[1,2] | |
| 342 | Shift-F1,2,5 | |
| 343 | Shift-F1,2,6 | |
| 347 | Shift-F1,3 | |
| 68 | Shift-F1,3,1 | |
| 44 | Shift-F1,3,2 | |
| 181 | Shift-F1,3,3 | |
| 348 | Shift-F1,3,4 | |
| 349 | Shift-F1,3,6 | |
| 101 | Shift-F1,3,7 | |
| 55 | Shift-F1,3,8 | |
| 45 | Shift-F1,4 | |
| 344 | Shift-F1,4,1 | |
| 45 | Shift-F1,4,2 | |
| 345 | Shift-F1,4,3 | |
| 400 | Shift-F1,4,3,2 | |
| 400 | Shift-F1,4,3,2,2 | |
| 379 | Shift-F1,4,3,3 | |
| 380 | Shift-F1,4,3,4 | |
| 220 | Shift-F1,4,7 | |
| 105 | Shift-F1,4,8 | |
| 252 | Shift-F1,4,8,2,Enter | |
| 369 | Shift-F1,4,8,2,Enter | running on a network |
| 202 | Shift-F1,4,8,3 | |
| 199 | Shift-F1,4,8,4 | |
| 11 | Shift-F1,4,8,5 | |
| 208 | Shift-F1,5 | |
| 209 | Shift-F1,5,7 | |
| 235 | Shift-F1,5,7,1 | |
| 303 | Shift-F1,5,8 | |
| 235 | Shift-F1,5,8,2 | |
| 56 | Shift-F1,6 | |
| Thesaurus Menu (Alt-F1) | | |
| 32824 | Alt-F1 | word not found |
| 110 | Alt-F1 | word found |
| Shell Menu (Ctrl-F1) | | |
| 103 | Ctrl-F1 | not under Shell |
| 102 | Ctrl-F1 | under Shell |

| Value | Keystrokes | Conditions |
|--------------------------------|------------------------|---------------------|
| Search Menus (F2 and Shift-F2) | | |
| 32801 | F2' | |
| 32813 | Shift-F21 | |
| 32776 | Home,F2 | |
| Replace Menu (Alt-F2) | | |
| 32825 | Alt-F2 | |
| Spell Menu (Ctrl-F2) | | |
| 17 | Ctrl-F2 | |
| 64 | Ctrl-F2,[1,2,3] | word not found |
| Help Menu (F3) | | |
| 32802 | F3 | |
| Switch Menu (Shift-F3) | | |
| 31 | Shift-F3 | Block on |
| Screen (Ctrl-F3) | | |
| 38 | Ctrl-F3 | |
| 362 | Ctrl-F3,2 | |
| 365 | Ctrl-F3,2,4 | |
| Move (Ctrl-F4) | | |
| 30 | Ctrl-F4 | |
| 23 | Ctrl-F4 | Block on |
| 32 | Ctrl-F4,[1,2,3] | Block on or off |
| 112 | Ctrl-F4,4 | |
| List Files (F5) | | |
| 389 | F5 | |
| 305 | F5.directory,Enter | |
| 401 | F5.directory,Enter,5 | |
| 114 | F5.directory,Enter,6 | no document summary |
| 334 | F5.directory,Enter,6 | document summary |
| 335 | F5.directory,Enter,6,3 | |
| 232 | F5.directory,Enter,9 | |
| 241 | F5.directory,Enter,9,5 | |
| Date/Outline (Shift-F5) | | |
| 24 | Shift-F5 | |
| 57 | Shift-F5,3 | |
| 350 | Shift-F5,4 | |
| 79 | Shift-F5,6 | |
| 256 | Shift-F5,6,9 | |
| 263 | Shift-F5,6,9,2 | |
| 339 | Shift-F5,6,9,2,3 | |
| 201 | Shift-F5,6,9,2,4 | |
| 263 | Shift-F5,6,9,3 | |
| 339 | Shift-F5,6,9,3,3 | |
| 201 | Shift-F5,6,9,3,4 | |
| 370 | Shift-F5,6,9,4 | |
| 390 | Shift-F5,6,9,6 | |
| Mark Text (Alt-F5) | | |
| 78 | Alt-F5 | |
| 2 | Alt-F5 | Block on |
| 159 | Alt-F5,1 | |
| 154 | Alt-F5,1,[1,3] | |
| 244 | Alt-F5,1,[1,3],5 | |
| 32806 | Alt-F5,4,Enter | Block on |
| 121 | Alt-F5,5 | |
| 88 | Alt-F5,5,1 | |
| 89 | Alt-F5,5,1,3 | |

| Value | Keystrokes | Conditions |
|-------------------------------|----------------------------|--|
| 85 | Alt-F5,5,2,[1,3] | |
| 36 | Alt-F5,5,4,1 | |
| 176 | Alt-F5,6 | |
| Text In/Out (Ctrl-F5) | | |
| 60 | Ctrl-F5 | |
| 32840 | Ctrl-F5 | Block on |
| 70 | Ctrl-F5,1 | |
| 249 | Ctrl-F5,2 | |
| 388 | Ctrl-F5,3 | |
| 250 | Ctrl-F5,4 | |
| 259 | Ctrl-F5,5 | |
| 260 | Ctrl-F5,5,[1,2,3] | |
| 390 | Ctrl-F5,5,[1,2,3],1 | |
| 261 | Ctrl-F5,5,[1,2,3],3 | |
| 314 | Ctrl-F5,5,4 | |
| Center (Shift-F6) | | |
| 32817 | Shift-F6 | Block on |
| Flush Right (Alt-F7) | | |
| 32829 | Alt-F7 | Block on |
| Exit (F7) | | |
| 32806 | F7,y | Long Document Name=No |
| 65027 | F7,y | Long Document Name=Yes |
| Print (Shift-F7) | | |
| 74 | Shift-F7 | |
| 126 | Shift-F7 | running on a network |
| 32818 | Shift-F7 | Block on |
| 217 | Shift-F7,4 | |
| 153 | Shift-F7,6 | |
| 76 | Shift-F7,s | |
| 76 | Shift-F7,s,2 | |
| 133 | Shift-F7,s,2,3 | |
| 47 | Shift-F7,s,2,4 | |
| 134 | Shift-F7,s,3 | |
| 46 | Shift-F7,s,3,2 | |
| 21 | Shift-F7,s,3,2,[4,5,6,7] | |
| 184 | Shift-F7,s,3,2,[4,5,6,7],1 | |
| 191 | Shift-F7,s,3,2,[4,5,6,7],2 | |
| 75 | Shift-F7,s,3,3 | |
| 42 | Shift-F7,s,3,4 | |
| 182 | Shift-F7,s,3,4,1 | cursor on Soft Fonts; soft fonts available |
| 132 | Shift-F7,s,3,5 | |
| 410 | Shift-F7,s,3,7,3 | |
| 252 | Shift-F7,u | |
| 202 | Shift-F7,g | |
| 199 | Shift-F7,t | |
| Columns/Table (Alt-F7) | | |
| 353 | Alt-F7 | cursor not in a table |
| 298 | Alt-F7 | cursor in a table |
| 301 | Alt-F7,Ins | cursor in a table |
| 309 | Alt-F7,Del | cursor in a table |
| 318 | Alt-F7,Ctrl-F4 | cursor in a table |
| 324 | Alt-F7,Ctrl-F4,[1,2,3] | cursor in a table |
| 16 | Alt-F7,1 | cursor not in a table |
| 325 | Alt-F7,1 | cursor in a table |
| 33 | Alt-F7,1,3 | cursor not in a table |
| 255 | Alt-F7,1,3,1 | cursor not in a table |
| 296 | Alt-F7,2 | cursor not in a table |

| Value | Keystrokes | Conditions |
|-------|------------------------------|--|
| 311 | Alt-F7,2 | cursor in a table |
| 296 | Alt-F7,2,1 | cursor not in a table |
| 306 | Alt-F7,2,1 | cursor in a table |
| 323 | Alt-F7,2,1,1 | cursor in a table |
| 351 | Alt-F7,2,1,2 | cursor in a table |
| 148 | Alt-F7,2,1,2,1 | cursor in a table |
| 149 | Alt-F7,2,1,2,2 | cursor in a table |
| 352 | Alt-F7,2,1,3 | cursor in a table |
| 308 | Alt-F7,2,1,4 | cursor in a table |
| 367 | Alt-F7,2,1,5 | cursor in a table |
| 298 | Alt-F7,2,2 | cursor not in a table; a table exists in document |
| 307 | Alt-F7,2,2 | cursor in a table |
| 301 | Alt-F7,2,2,Ins | cursor not in a table; a table exists in document |
| 309 | Alt-F7,2,2,Del | cursor not in a table; a table exists in document |
| 318 | Alt-F7,2,2,Ctrl-F4 | cursor not in a table; a table exists in document |
| 324 | Alt-F7,2,2,Ctrl-F4,[1,2,3] | cursor not in a table; a table exists in document |
| 325 | Alt-F7,2,2,1 | cursor not in a table; a table exists in document |
| 307 | Alt-F7,2,2,1 | cursor in a table |
| 311 | Alt-F7,2,2,2 | cursor not in a table; a table exists in document |
| 231 | Alt-F7,2,2,2 | cursor in a table |
| 306 | Alt-F7,2,2,2,1 | cursor not in a table; a table exists in document |
| 148 | Alt-F7,2,2,2,1 | cursor in a table |
| 323 | Alt-F7,2,2,2,1,1 | cursor not in a table; a table exists in document |
| 351 | Alt-F7,2,2,2,1,2 | cursor not in a table; a table exists in document |
| 148 | Alt-F7,2,2,2,1,2,1 | cursor not in a table; a table exists in document |
| 149 | Alt-F7,2,2,2,1,2,2 | cursor not in a table; a table exists in document |
| 352 | Alt-F7,2,2,2,1,3 | cursor not in a table; a table exists in document |
| 308 | Alt-F7,2,2,2,1,4 | cursor not in a table; a table exists in document |
| 367 | Alt-F7,2,2,2,1,5 | cursor not in a table; a table exists in document |
| 307 | Alt-F7,2,2,2,2 | cursor not in a table; a table exists in document |
| 149 | Alt-F7,2,2,2,2 | cursor in a table |
| 307 | Alt-F7,2,2,2,2,1 | cursor not in a table; a table exists in document |
| 231 | Alt-F7,2,2,2,2,2 | cursor not in a table; a table exists in document |
| 148 | Alt-F7,2,2,2,2,2,1 | cursor not in a table; a table exists in document |
| 149 | Alt-F7,2,2,2,2,2,2 | cursor not in a table; a table exists in document |
| 317 | Alt-F7,2,2,2,2,3 | cursor not in a table; a table exists in document |
| 312 | Alt-F7,2,2,2,3 | cursor not in a table; a table exists in document |
| 299 | Alt-F7,2,2,3 | cursor not in a table; a table exists in document |
| 317 | Alt-F7,2,2,3 | cursor in a table |
| 145 | Alt-F7,2,2,3,[1,2,3,4,5,6,7] | cursor not in a table; a table exists in document |
| 300 | Alt-F7,2,2,3,8 | cursor not in a table; a table exists in document |
| 298 | Alt-F7,2,2,4 | cursor not in a table; a table exists in document |
| 297 | Alt-F7,2,2,5 | cursor not in a table; a table exists in document |
| 366 | Alt-F7,2,2,5,3 | cursor not in a table; a table exists in document; formula exists in cell |
| 313 | Alt-F7,2,2,6 | cursor not in a table; a table exists in document |
| 333 | Alt-F7,2,2,6,3 | cursor not in a table; a table exists in document |
| 298 | Alt-F7,2,2,7 | cursor not in a table; a table exists in document |
| 302 | Alt-F7,2,2,8 | cursor not in a table; a table exists in document |
| 312 | Alt-F7,2,3 | cursor in a table |
| 28 | Alt-F7,3 | cursor not in a table |
| 299 | Alt-F7,3 | cursor in a table |
| 145 | Alt-F7,3,[1,2,3,4,5,6,7] | cursor in a table |
| 300 | Alt-F7,3,8 | cursor in a table |
| 298 | Alt-F7,4 | cursor in a table |
| 297 | Alt-F7,5 | cursor in a table |
| 366 | Alt-F7,5,3 | cursor in a table; formula exists in cell |
| 313 | Alt-F7,6 | cursor in a table |
| 333 | Alt-F7,6,3 | cursor in a table |
| 298 | Alt-F7,7 | cursor in a table |
| 302 | Alt-F7,8 | cursor in a table |

| Value | Keystrokes | Conditions |
|---------------------------|--------------------------|--------------------------|
| Footnote (Ctrl-F7) | | |
| 3 | Ctrl-F7 | |
| 137 | Ctrl-F7,1 | |
| 67 | Ctrl-F7,1,4 | |
| 92 | Ctrl-F7,1,4,5 | |
| 223 | Ctrl-F7,1,4,7 | |
| 392 | Ctrl-F7,2 | |
| 135 | Ctrl-F7,2,4 | |
| 92 | Ctrl-F7,2,4,5 | |
| Format (Shift-F8) | | |
| 32819 | Shift-F8 | Block on |
| 155 | Shift-F8 | |
| 5 | Shift-F8,1 | |
| 331 | Shift-F8,1,3 | |
| 20 | Shift-F8,1,4 | |
| 122 | Shift-F8,1,5,y | |
| 12 | Shift-F8,1,8 | |
| 358 | Shift-F8,1,8,t | |
| 1 | Shift-F8,2 | |
| 156 | Shift-F8,2,2 | |
| 6 | Shift-F8,2,3 | |
| 240 | Shift-F8,2,3,[1,2] | |
| 117 | Shift-F8,2,4 | |
| 240 | Shift-F8,2,4,[1,2] | |
| 368 | Shift-F8,2,6 | |
| 9 | Shift-F8,2,6,4 | |
| 328 | Shift-F8,2,7 | |
| 229 | Shift-F8,2,7,2 | |
| 230 | Shift-F8,2,7,2,Enter,1 | |
| 229 | Shift-F8,2,7,2,Enter,2 | |
| 330 | Shift-F8,2,7,2,Enter,3 | |
| 234 | Shift-F8,2,7,2,Enter,5 | |
| 359 | Shift-F8,2,7,2,Enter,7 | |
| 356 | Shift-F8,2,7,2,Enter,8,y | |
| 329 | Shift-F8,2,7,2,Enter,9 | |
| 327 | Shift-F8,2,7,5 | form is not [ALL OTHERS] |
| 230 | Shift-F8,2,7,5,1 | form is not [ALL OTHERS] |
| 229 | Shift-F8,2,7,5,2 | form is not [ALL OTHERS] |
| 330 | Shift-F8,2,7,5,3 | form is not [ALL OTHERS] |
| 234 | Shift-F8,2,7,5,5 | form is not [ALL OTHERS] |
| 359 | Shift-F8,2,7,5,7 | form is not [ALL OTHERS] |
| 356 | Shift-F8,2,7,5,8,y | form is not [ALL OTHERS] |
| 329 | Shift-F8,2,7,5,9 | form is not [ALL OTHERS] |
| 326 | Shift-F8,2,7,5 | form is [ALL OTHERS] |
| 234 | Shift-F8,2,7,5,3 | form is [ALL OTHERS] |
| 329 | Shift-F8,2,7,5,4 | form is [ALL OTHERS] |
| 10 | Shift-F8,2,8 | |
| 169 | Shift-F8,3 | |
| 132 | Shift-F8,3,3 | |
| 11 | Shift-F8,3,4 | |
| 125 | Shift-F8,3,5 | |
| 25 | Shift-F8,4 | |
| 150 | Shift-F8,4,1 | |
| 164 | Shift-F8,4,5 | |
| 7 | Shift-F8,4,6 | |
| 84 | Shift-F8,4,6,2 | |
| 93 | Shift-F8,4,6,3 | |
| 151 | Shift-F8,4,6,3,Enter | |
| 403 | Shift-F8,4,8 | |

| Value | Keystrokes | Conditions |
|-------------------------------|--|--|
| Style (Alt-F8) | | |
| 59 | Alt-F8 | |
| 387 | Alt-F8,3 | |
| 174 | Alt-F8,3,2 | |
| 201 | Alt-F8,3,5 | paired styles only |
| 387 | Alt-F8,4 | not outline style |
| 263 | Alt-F8,4 | outline style |
| 174 | Alt-F8,4,2 | not outline style |
| 339 | Alt-F8,4,3 | outline style |
| 201 | Alt-F8,4,4 | outline style |
| 201 | Alt-F8,4,5 | not outline style; paired styles only |
| 370 | Alt-F8,5 | |
| Font (Ctrl-F8) | | |
| 147 | Ctrl-F8 | |
| 63 | Ctrl-F8 | Block on |
| 148 | Ctrl-F8,1 | Block on or off |
| 149 | Ctrl-F8,2 | Block on or off |
| 132 | Ctrl-F8,4 | |
| 146 | Ctrl-F8,5 | |
| Merge Codes (Shift-F9) | | |
| 340 | Shift-F9 | |
| 340 | Shift-F9,6 | |
| Graphics (Alt-F9) | | |
| 136 | Alt-F9 | |
| 393 | Alt-F9,1 | |
| 138 | Alt-F9,1,1 | |
| 257 | Alt-F9,1,1,2 | |
| 139 | Alt-F9,1,1,4 | |
| 140 | Alt-F9,1,1,5 | anchor type=Page |
| 152 | Alt-F9,1,1,5 | anchor type=Character |
| 142 | Alt-F9,1,1,6 | anchor type=Paragraph |
| 141 | Alt-F9,1,1,6 | anchor type=Page |
| 142 | Alt-F9,1,1,6,1 | anchor type=Page |
| 142 | Alt-F9,1,1,6,2, number of columns,Enter | |
| 210 | Alt-F9,1,1,7 | |
| 402 | Alt-F9,1,1,9 | Contents=Graphics; DrawPerfect on Shell |
| 374 | Alt-F9,1,1,9 | Contents=Equation; DrawPerfect on Shell |
| 373 | Alt-F9,1,1,9 | DrawPerfect not on Shell |
| 225 | Alt-F9,1,1,9,Alt-F9 | Contents=Text |
| 399 | Alt-F9,1,1,9,F5 | Contents=Equation |
| 399 | Alt-F9,1,1,9,Shift-F3 | Contents=Equation; equation palette active |
| 374 | Alt-F9,1,1,9,Shift-F3 | Contents=Equation; editing window active |
| 375 | Alt-F9,1,1,9,Shift-F3 | Contents=Equation; display window active |
| 381 | Alt-F9,1,1,9,Shift-F1 | Contents=Equation |
| 400 | Alt-F9,1,1,9,Shift-F1,2 | Contents=Equation |
| 379 | Alt-F9,1,1,9,Shift-F1,3 | Contents=Equation |
| 380 | Alt-F9,1,1,9,Shift-F1,4 | Contents=Equation |
| 144 | Alt-F9,1,4 | |
| 145 | Alt-F9,1,4,1 | |
| 239 | Alt-F9,1,4,[4,5] | |
| 183 | Alt-F9,1,4,7 | Contents=Equation |
| 143 | Alt-F9,1,4,7 | Contents≠Equation |
| 115 | Alt-F9,1,4,7,[1,2] | Contents≠Equation |
| 394 | Alt-F9,2 | |
| 138 | Alt-F9,2,1 | |
| 144 | Alt-F9,2,4 | |
| 395 | Alt-F9,3 | |
| 138 | Alt-F9,3,1 | |
| 144 | Alt-F9,3,4 | |

| Value | Keystrokes | Conditions |
|--------------------------------|-------------------------------------|--------------------------------------|
| 396 | Alt-F9,4 | |
| 138 | Alt-F9,4,1 | |
| 144 | Alt-F9,4,4 | |
| 227 | Alt-F9,5 | |
| 224 | Alt-F9,5,1 | |
| 243 | Alt-F9,5,1,1 | |
| 361 | Alt-F9,5,1,2 | |
| 226 | Alt-F9,5,1,2,2 | |
| 226 | Alt-F9,5,2 | |
| 242 | Alt-F9,5,2,1 | |
| 140 | Alt-F9,5,2,2 | |
| 226 | Alt-F9,5,2,2,5 | |
| 224 | Alt-F9,5,3 | |
| 226 | Alt-F9,5,4 | |
| 397 | Alt-F9,6 | |
| 138 | Alt-F9,6,1 | |
| 144 | Alt-F9,6,4 | |
| Merge/Sort (Ctrl-F9) | | |
| 62 | Ctrl-F9 | |
| 390 | Ctrl-F9,1 | |
| 52 | Ctrl-F9,2, file,Enter, file,Enter | |
| 61 | Ctrl-F9,2, file,Enter, file,Enter,3 | |
| 109 | Ctrl-F9,2, file,Enter, file,Enter,5 | Select active |
| 108 | Ctrl-F9,2, file,Enter, file,Enter,6 | |
| 107 | Ctrl-F9,2, file,Enter, file,Enter,7 | |
| Save (F10) | | |
| 65027 | F1 | Long Document Name=Yes |
| 32809 | F10 | Long Document Name=No |
| Retrieve (Shift-F10) | | |
| 32821 | Shift-F10 | |
| Macro (Alt-F10) | | |
| 32833 | Alt-F10 | |
| Macro Define (Ctrl-F10) | | |
| 32845 | Ctrl-F10 | |
| 222 | Ctrl-F10 | macro already defined |
| 236 | Ctrl-F10,2 | macro already defined |
| Other Keys | | |
| 32795 | Esc | repeat value |
| 32848 | Backspace | deleting code and Reveal Codes off |
| 32776 | Home | |
| 32856 | Ctrl-Home | |
| 32849 | Del | deleting code and Reveal Codes off |
| 32780 | Ctrl-PgDn | |
| 253 | Ctrl-PgUp | macro definition or execution active |
| 32790 | Ctrl-v | |
| Pull-Down Menus | | |
| 264 | Menu Bar (Alt-=) | |
| 265 | File | |
| 273 | File,Text In | |
| 321 | File,Text In,Spreadsheet | |
| 272 | File,Text Out | |
| 293 | File>Password | |
| 377 | File.Setup | |
| 266 | Edit | |
| 270 | Edit,Append | Block on |
| 274 | Edit,Select | |

| Value | Keystrokes | Conditions |
|-------|---|------------|
| 322 | Edit, Comment | |
| 275 | Edit, Convert Case | |
| 287 | Search | |
| 277 | Search, Extended | |
| 268 | Layout | |
| 276 | Layout, Columns | |
| 378 | Layout, Tables | |
| 286 | Layout, Math | |
| 279 | Layout, Footnote | |
| 280 | Layout, Endnote | |
| 283 | Layout, Justify | |
| 290 | Layout, Align | |
| 376 | Mark | |
| 288 | Mark, Cross-Reference | |
| 289 | Mark, Table of Authorities | |
| 285 | Mark, Define | |
| 295 | Mark, Master Documents | |
| 294 | Mark, Document Compare | |
| 271 | Tools | |
| 281 | Tools, Macro | |
| 284 | Tools, Outline | |
| 282 | Tools, Merge Codes | |
| 269 | Font | |
| 278 | Font, Appearance | |
| 267 | Graphics | |
| 279 | Graphics, [Figure, Table, Text Box, User Box, Equation] | |
| 291 | Graphics, Line | |
| 292 | Help | |

The value returned corresponds to the original key pressed, not to the direction of the search. For example, if you press Shift-F2, then press ↓ to change the search to a forward search, {SYSTEM}Menu still returns 32813.

By Number

This table lists the menus by menu number (value returned by {SYSTEM}Menu). Use this table when you know the value but not the menu(s) at which that value is returned.

| Value | Keystrokes | Conditions |
|-------|--------------------------|-----------------------|
| 1 | Shift-F8,2 | |
| 2 | Alt-F5 | Block on |
| 3 | Ctrl-F7 | |
| 5 | Shift-F8,1 | |
| 6 | Shift-F8,2,3 | |
| 7 | Shift-F8,4,6 | |
| 9 | Shift-F8,2,6,4 | |
| 10 | Shift-F8,2,8 | |
| 11 | Shift-F1,4,8,5 | |
| | Shift-F8,3,4 | |
| 12 | Shift-F8,1,8 | |
| 16 | Alt-F7,1 | cursor not in a table |
| 17 | Ctrl-F2 | |
| 20 | Shift-F8,1,4 | |
| 21 | Shift-F7,s,3,2,[4,5,6,7] | |
| 23 | Ctrl-F4 | Block on |
| 24 | Shift-F5 | |
| 25 | Shift-F8,4 | |
| 28 | Alt-F7,3 | cursor not in a table |
| 30 | Ctrl-F4 | |
| 31 | Shift F3 | Block on |

| Value | Keystrokes | Conditions |
|-------|-----------------------------------|-----------------------|
| 32 | Ctrl-F4,[1,2,3] | Block on or off |
| 33 | Alt-F7,1,3 | cursor not in a table |
| 36 | Alt-F5,5,4,1 | |
| 38 | Ctrl-F3 | |
| 42 | Shift-F7,s,3,4 | |
| 44 | Shift-F1,3,2 | |
| 45 | Shift-F1,4 | |
| | Shift-F1,4,2 | |
| 46 | Shift-F7,s,3,2 | |
| 47 | Shift-F7,s,2,4 | |
| 49 | F1 | |
| 52 | Ctrl-F9,2,file,Enter,file,Enter | |
| 55 | Shift-F1,3,8 | |
| 56 | Shift-F1,6 | |
| 57 | Shift-F5,3 | |
| 59 | Alt-F8 | |
| 60 | Ctrl-F5 | |
| 61 | Ctrl-F9,2,file,Enter,file,Enter,3 | |
| 62 | Ctrl-F9 | |
| 63 | Ctrl-F8 | Block on |
| 64 | Ctrl-F2,[1,2,3] | word not found |
| 67 | Ctrl-F7,1,4 | |
| 68 | Shift-F1,3,1 | |
| 70 | Ctrl-F5,1 | |
| 74 | Shift-F7 | |
| 75 | Shift-F7,s,3,3 | |
| 76 | Shift-F7,s | |
| | Shift-F7,s,2 | |
| 78 | Alt-F5 | |
| 79 | Shift-F5,6 | |
| 84 | Shift-F8,4,6,2 | |
| 85 | Alt-F5,5,2,[1,3] | |
| 88 | Alt-F5,5,1 | |
| 89 | Alt-F5,5,1,3 | |
| 92 | Ctrl-F7,[1,2],4,5 | |
| 93 | Shift-F8,4,6,3 | |
| 101 | Shift-F1,3,7 | |
| 102 | Ctrl-F1 | under Shell |
| 103 | Ctrl-F1 | not under Shell |
| 105 | Shift-F1,4,8 | |
| 107 | Ctrl-F9,2,file,Enter,file,Enter,7 | |
| 108 | Ctrl-F9,2,file,Enter,file,Enter,6 | |
| 109 | Ctrl-F9,2,file,Enter,file,Enter,5 | |
| 110 | Alt-F1 | Select active |
| 112 | Ctrl-F4,4 | word found |
| 114 | F5,directory,Enter,6 | |
| 115 | Alt-F9,1,4,7,[1,2] | no document summary |
| 117 | Shift-F8,2,4 | Contents=Equation |
| 121 | Alt-F5,5 | |
| 122 | Shift-F8,1,5,y | |
| 125 | Shift-F8,3,5 | |
| 126 | Shift-F7 | running on a network |
| 132 | Ctrl-F8,4 | |
| | Shift-F7,s,3,5 | |
| | Shift-F8,3,3 | |
| 133 | Shift-F7,s,2,3 | |
| 134 | Shift-F7,s,3 | |
| 135 | Ctrl-F7,2,4 | |
| 136 | Alt-F9 | |
| 137 | Ctrl-F7,1 | |
| 138 | Alt-F9,[1,2,3,4,6],1 | |
| 139 | Alt-F9,1,1,4 | |

| Value | Keystrokes | Conditions |
|-------|---|---|
| 140 | Alt-F9,1,1,5 | anchor type=Page |
| | Alt-F9,5,2,2 | |
| 141 | Alt-F9,1,1,6 | anchor type=Page |
| 142 | Alt-F9,1,1,6 | anchor type=Paragraph |
| | Alt-F9,1,1,6,1 | anchor type=Page |
| | Alt-F9,1,1,6,2, <i>number of columns</i> ,Enter | anchor type=Page |
| 143 | Alt-F9,1,4,7 | Contents=Equation |
| 144 | Alt-F9,[1,2,3,4,6],4 | |
| 145 | Alt-F7,2,2,3,[1,2,3,4,5,6,7] | cursor not in a table; a table exists in document |
| | Alt-F7,3,[1,2,3,4,5,6,7] | cursor in a table |
| | Alt-F9,1,4,1 | |
| 146 | Ctrl-F8,5 | |
| 147 | Ctrl-F8 | |
| 148 | Alt-F7,2,[1,2],2,1 | cursor in a table |
| | Alt-F7,2,2,2,[1,2],2,1 | cursor not in a table; a table exists in document |
| | Ctrl-F8,1 | Block on or off |
| 149 | Alt-F7,2,[1,2],2,2 | cursor in a table |
| | Alt-F7,2,2,2,[1,2],2,2 | cursor not in a table; a table exists in document |
| | Ctrl-F8,2 | Block on or off |
| 150 | Shift-F8,4,1 | |
| 151 | Shift-F8,4,6,3,Enter | |
| 152 | Alt-F9,1,1,5 | anchor type=Character |
| 153 | Shift-F7,6 | |
| 154 | Alt-F5,1,[1,3] | |
| 155 | Shift-F8 | |
| 156 | Shift-F8,2,2 | |
| 159 | Alt-F5,1 | |
| 164 | Shift-F8,4,5 | |
| 169 | Shift-F8,3 | |
| 174 | Alt-F8,3,2 | |
| | Alt-F8,4,2 | not outline style |
| 176 | Alt-F5,6 | |
| 177 | Shift-F1 | |
| 181 | Shift-F1,3,3 | |
| 182 | Shift-F7,s,3,4,1 | cursor on Soft Fonts; soft fonts available |
| 183 | Alt-F9,1,4,7 | Contents=Equation |
| 184 | Shift-F7,s,3,2,[4,5,6,7],1 | |
| 190 | Shift-F1,2,1 | |
| 191 | Shift-F7 s,3,2,[4,5,6,7] 2 | |
| 199 | Shift-F1,4,8,4 | |
| | Shift-F7,1 | |
| 201 | Alt-F8,3,5,1 | paired styles only |
| | Alt-F8,4,4 | outline style |
| | Alt-F8,4,5 | not outline style; paired styles only |
| | Shift-F5,6,9,[2,3],4 | |
| 202 | Shift-F1,4,8,3 | |
| | Shift-F7,g | |
| 208 | Shift-F1,5 | |
| 209 | Shift-F1,5,7 | |
| 210 | Alt-F9,1,1,7 | |
| 217 | Shift-F7,4 | |
| 220 | Shift-F1,4,7 | |
| 221 | Shift-F1,2 | |
| | Shift-F1,2,[2,3] | |
| 222 | Ctrl-F10 | macro already defined |
| 223 | Ctrl-F7,1,4,7 | |
| 224 | Alt-F9,5,1 | |
| | Alt-F9,5,3 | |
| 225 | Alt-F9,1,1,9,Alt-F9 | Contents=Text |
| 226 | Alt-F9,5,1,2,2 | |
| | Alt-F9,5,[2,4] | |
| | Alt-F9,5,2,2,5 | |

| Value | Keystrokes | Conditions |
|-------|--|---|
| 227 | Alt-F9,5 | |
| 229 | Shift-F8,2,7,2 | |
| | Shift-F8,2,7,2,Enter,2 | form is not [ALL OTHERS] |
| | Shift-F8,2,7,5,2 | |
| 230 | Shift-F8,2,7,2,Enter,1 | form is not [ALL OTHERS] |
| | Shift-F8,2,7,5,1 | cursor in a table |
| 231 | Alt-F7,2,2,2 | cursor not in a table; a table exists in document |
| | Alt-F7,2,2,2,2 | |
| 232 | F5, <i>directory</i> , Enter, 9 | |
| 234 | Shift-F8,2,7,2,Enter,5 | form is [ALL OTHERS] |
| | Shift-F8,2,7,5,3 | form is not [ALL OTHERS] |
| | Shift-F8,2,7,5,5 | |
| 235 | Shift-F1,5,7,1 | |
| | Shift-F1,5,8,2 | |
| 236 | Ctrl-F10,2 | macro already defined |
| 239 | Alt-F9,1,4,[4,5] | |
| 240 | Shift-F8,2,[3,4],[1,2] | |
| 241 | F5, <i>directory</i> , Enter, 9, 5 | |
| 242 | Alt-F9,5,2,1 | |
| 243 | Alt-F9,5,1,1 | |
| 244 | Alt-F5,1,[1,3],5 | |
| 249 | Ctrl-F5,2 | |
| 250 | Ctrl-F5,4 | |
| 252 | Shift-F1,4,8,2,Enter | |
| | Shift-F7,u | |
| 253 | Ctrl-PgUp | macro definition or execution active |
| 255 | Alt-F7,1,3,1 | cursor not in a table |
| 256 | Shift-F5,6,9 | |
| 257 | Alt-F9,1,1,2 | |
| 259 | Ctrl-F5,5 | |
| 260 | Ctrl-F5,5,[1,2,3] | |
| 261 | Ctrl-F5,5,[1,2,3],3 | |
| 263 | Alt-F8,4 | outline style |
| 263 | Shift-F5,6,9,[2,3] | |
| 264 | (pull-down) Menu Bar (Alt--) | |
| 265 | (pull-down) F ile | |
| 266 | (pull-down) E dit | |
| 267 | (pull-down) G raphics | |
| 268 | (pull-down) L ayout | |
| 269 | (pull-down) F ont | |
| 270 | (pull-down) E dit, A ppend | Block on |
| 271 | (pull-down) T ools | |
| 272 | (pull-down) F ile, T ext O ut | |
| 273 | (pull-down) F ile, T ext I n | |
| 274 | (pull-down) E dit, S elect | |
| 275 | (pull-down) E dit, C onvert C ase | |
| 276 | (pull-down) L ayout, C olumns | |
| 277 | (pull-down) S earch, E xtended | |
| 278 | (pull-down) F ont, A ppearance | |
| 279 | (pull-down) G raphics, [E quation, F igure, T able, T ext B ox, U ser B ox] | |
| 279 | (pull-down) L ayout, F ootnote | |
| 280 | (pull-down) L ayout, E ndnote | |
| 281 | (pull-down) T ools, M acro | |
| 282 | (pull-down) T ools, M erge C odes | |
| 283 | (pull-down) L ayout, J ustify | |
| 284 | (pull-down) T ools, O utline | |
| 285 | (pull-down) M ark, D efine | |
| 286 | (pull-down) L ayout, M ath | |
| 287 | (pull-down) S earch | |
| 288 | (pull-down) M ark, C ross- R eference | |
| 289 | (pull-down) M ark, T able of A uthorities | |
| 290 | (pull-down) L ayout, A lign | |

| Value | Keystrokes | Conditions |
|-------|--|---|
| 291 | (pull-down) G raphics,Line | |
| 292 | (pull-down) H elp | |
| 293 | (pull-down) F ile,Password | |
| 294 | (pull-down) M ark,Document Compare | |
| 295 | (pull-down) M ark,Master Documents | |
| 296 | Alt-F7,2 | cursor not in a table |
| 296 | Alt-F7,2,1 | cursor not in a table |
| 297 | Alt-F7,2,2,5 | cursor not in a table; a table exists in document |
| | Alt-F7,5 | cursor in a table |
| 298 | Alt-F7 | cursor in a table |
| | Alt-F7,2,2 | cursor not in a table; a table exists in document |
| | Alt-F7,2,2,[4,7] | cursor not in a table; a table exists in document |
| | Alt-F7,[4,7] | cursor in a table |
| 299 | Alt-F7,2,2,3 | cursor not in a table; a table exists in document |
| | Alt-F7,3 | cursor in a table |
| 300 | Alt-F7,2,2,3,8 | cursor not in a table; a table exists in document |
| | Alt-F7,3,8 | cursor in a table |
| 301 | Alt-F7,2,2,Ins | cursor not in a table; a table exists in document |
| | Alt-F7,Ins | cursor in a table |
| 302 | Alt-F7,2,2,8 | cursor not in a table; a table exists in document |
| | Alt-F7,8 | cursor in a table |
| 303 | Shift-F1,5,8 | |
| 305 | F5, <i>directory</i> ,Enter | |
| 306 | Alt-F7,2,1 | cursor in a table |
| | Alt-F7,2,2,1 | cursor not in a table; a table exists in document |
| 307 | Alt-F7,2,2 | cursor in a table |
| | Alt-F7,2,2,1 | cursor in a table |
| | Alt-F7,2,2,2 | cursor not in a table; a table exists in document |
| | Alt-F7,2,2,2,1 | cursor not in a table; a table exists in document |
| 308 | Alt-F7,2,1,4 | cursor in a table |
| | Alt-F7,2,2,1,4 | cursor not in a table; a table exists in document |
| 309 | Alt-F7,2,2,Del | cursor not in a table; a table exists in document |
| | Alt-F7,Del | cursor in a table |
| 311 | Alt-F7,2 | cursor in a table |
| | Alt-F7,2,2 | cursor not in a table; a table exists in document |
| 312 | Alt-F7,2,2,3 | cursor not in a table; a table exists in document |
| | Alt-F7,2,3 | cursor in a table |
| 313 | Alt-F7,2,2,6 | cursor not in a table; a table exists in document |
| | Alt-F7,6 | cursor in a table |
| 314 | Ctrl-F5,5,4 | |
| 317 | Alt-F7,2,2,2,2,3 | cursor not in a table; a table exists in document |
| | Alt-F7,2,2,3 | cursor in a table |
| 318 | Alt-F7,2,2,Ctrl-F4 | cursor not in a table; a table exists in document |
| | Alt-F7,Ctrl-F4 | cursor in a table |
| 321 | (pull-down) F ile,Text In,Spreadsheet | |
| 322 | (pull-down) E dit,Comment | |
| 323 | Alt-F7,2,1,1 | cursor in a table |
| | Alt-F7,2,2,2,1,1 | cursor not in a table; a table exists in document |
| 324 | Alt-F7,2,2,Ctrl-F4,[1,2,3] | cursor not in a table; a table exists in document |
| | Alt-F7,Ctrl-F4,[1,2,3] | cursor in a table |
| 325 | Alt-F7,1 | cursor in a table |
| | Alt-F7,2,2,1 | cursor not in a table; a table exists in document |
| 326 | Shift-F8,2,7,5 | form is [ALL OTHERS] |
| 327 | Shift-F8,2,7,5 | form is not [ALL OTHERS] |
| 328 | Shift-F8,2,7 | |
| 329 | Shift-F8,2,7,2,Enter,9 | |
| | Shift-F8,2,7,5,4 | form is [ALL OTHERS] |
| | Shift-F8,2,7,5,9 | form is not [ALL OTHERS] |
| 330 | Shift-F8,2,7,2,Enter,3 | |
| | Shift-F8,2,7,5,3 | form is not [ALL OTHERS] |
| 331 | Shift-F8,1,3 | |
| 333 | Alt-F7,2,2,6,3 | cursor not in a table; a table exists in document |
| | Alt-F7,6,3 | cursor in a table |

| Value | Keystrokes | Conditions |
|-------|---|---|
| 334 | F5, <i>directory</i> , Enter, 6 | document summary |
| 335 | F5, <i>directory</i> , Enter, 6, 3 | |
| 339 | Alt F8, 4, 3 | outline style |
| | Shift-F5, 6, 9, [2, 3], 3 | |
| 340 | Shift-F9 | |
| | Shift-F9, 6 | |
| 341 | Shift-F1, 2, 4 | |
| | Shift-F1, 2, 4, [1, 2, 3, 5, 6], [1, 2] | |
| 342 | Shift-F1, 2, 5 | |
| 343 | Shift-F1, 2, 6 | |
| 344 | Shift-F1, 4, 1 | |
| 345 | Shift-F1, 4, 3 | |
| 346 | Shift-F1, 1 | |
| 347 | Shift-F1, 3 | |
| 348 | Shift-F1, 3, 4 | |
| 349 | Shift-F1, 3, 6 | |
| 350 | Shift-F5, 4 | |
| 351 | Alt-F7, 2, 1, 2 | cursor in a table |
| | Alt-F7, 2, 2, 2, 1, 2 | cursor not in a table; a table exists in document |
| 352 | Alt-F7, 2, 1, 3 | cursor in a table |
| | Alt-F7, 2, 2, 2, 1, 3 | cursor not in a table; a table exists in document |
| 353 | Alt-F7 | cursor not in a table |
| 356 | Shift-F8, 2, 7, 2, Enter, 8, y | |
| | Shift-F8, 2, 7, 5, 8, y | |
| 358 | Shift-F8, 1, 8, t | form is not [ALL OTHERS] |
| 359 | Shift-F8, 2, 7, 2, Enter, 7 | |
| | Shift-F8, 2, 7, 5, 7 | form is not [ALL OTHERS] |
| 361 | Alt-F9, 5, 1, 2 | |
| 362 | Ctrl-F3, 2 | |
| 363 | Shift-F1, 1, 2 | |
| 365 | Ctrl-F3, 2, 4 | |
| 366 | Alt-F7, 2, 2, 5, 3 | cursor not in a table; a table exists in document; formula exists in cell |
| 366 | Alt-F7, 5, 3 | cursor in a table; formula exists in cell |
| 367 | Alt-F7, 2, 1, 5 | cursor in a table |
| | Alt-F7, 2, 2, 2, 1, 5 | cursor not in a table; a table exists in document |
| 368 | Shift-F8, 2, 6 | |
| 369 | Shift-F1, 4, 8, 2, Enter | network |
| 370 | Alt-F8, 5 | |
| | Shift-F5, 6, 9, 4 | |
| 373 | Alt-F9, 1, 1, 9 | DrawPerfect not on Shell |
| 374 | Alt-F9, 1, 1, 9 | Contents=Equation: DrawPerfect on Shell |
| | Alt-F9, 1, 1, 9, Shift-F3 | Contents=Equation: editing window active |
| 375 | Alt-F9, 1, 1, 9, Shift-F3 | Contents=Equation: display window active |
| 376 | (pull-down) Mark | |
| 377 | (pull-down) File, Setup | |
| 378 | (pull-down) Layout, Tables | |
| 379 | Alt-F9, 1, 1, 9, Shift-F1, 3 | Contents=Equation |
| | Shift-F1, 4, 3, 3 | |
| 380 | Alt-F9, 1, 1, 9, Shift-F1, 4 | Contents=Equation |
| | Shift-F1, 4, 3, 4 | |
| 381 | Alt-F9, 1, 1, 9, Shift-F1 | Contents=Equation |
| 387 | Alt-F8, 3 | |
| | Alt-F8, 4 | not outline style |
| 388 | Ctrl-F5, 3 | |
| 389 | F5 | |
| 390 | Ctrl-F5, 5, [1, 2, 3], 1 | |
| | Ctrl-F9, 1 | |
| | Shift-F5, 6, 9, 6 | |
| 392 | Ctrl-F7, 2 | |
| 393 | Alt-F9, 1 | |
| 394 | Alt-F9, 2 | |
| 395 | Alt-F9, 3 | |

| Value | Keystrokes | Conditions |
|-------|---------------------------------|--|
| 396 | Alt-F9,4 | |
| 397 | Alt-F9,6 | |
| 399 | Alt-F9,1,1,9,F5 | Contents=Equation |
| | Alt-F9,1,1,9,Shift-F3 | Contents=Equation; equation palette active |
| 400 | Alt-F9,1,1,9,Shift-F1,2 | Contents=Equation |
| | Shift-F1,4,3,2 | |
| | Shift-F1,4,3,2,2 | |
| 401 | F5, <i>directory</i> , Enter, 5 | |
| 402 | Alt-F9,1,1,9 | Contents=Graphics; DrawPerfect on Shell |
| 403 | Shift-F8,4,8 | |
| 410 | Shift-F7,s,3,7,3 | |
| 32776 | Home | |
| | Home,F2 | |
| 32780 | Ctrl-PgDn | |
| 32790 | Ctrl-v | |
| 32795 | Esc | repeat value |
| 32801 | F2' | |
| 32802 | F3 | |
| 32806 | Alt-F5,4,Enter | Block on |
| | F7,y | Long Document Name=No |
| 32809 | F10 | Long Document Name=No |
| 32813 | Shift-F2' | |
| 32817 | Shift-F6 | Block on |
| 32818 | Shift-F7 | Block on |
| 32819 | Shift-F8 | Block on |
| 32821 | Shift-F10 | |
| 32824 | Alt-F1 | word not found |
| 32825 | Alt-F2 | |
| 32829 | Alt-F7 | Block on |
| 32833 | Alt-F10 | |
| 32840 | Ctrl-F5 | Block on |
| 32845 | Ctrl-F10 | |
| 32848 | Backspace | deleting code and Reveal Codes off |
| 32849 | Del | deleting code and Reveal Codes off |
| 32856 | Ctrl-Home | |
| 65027 | F10 | Long Document Name=Yes |
| 65027 | F7,y | Long Document Name=Yes |

'The value returned corresponds to the original key pressed, not to the direction of the search. For example, if you press Shift-F2, then press ↓ to change the search to a forward search, {SYSTEM}Menu~ still returns 32813.