Chapter 10

Glossary of Macro Commands/Routines

Links: The Chapter Name, above, moves to Release Notes. All page header links go to the contents menu. Within the chapter, Topic Titles returns here. Other Red Links are to other locations in this paper. Blue Links are to web sources. Click a letter in the header to move to a chronological entry.

Top	Contents	Glossary	Index	Release Notes
Chapter 1 Additional Resources	Chapter 2 Understanding Macros	Chapter 3 First Things	Chapter 4 Controlling Macro Flow	Chapter 5 Using the Dialog Editor
Chapter 6 Math Routines	Chapter 7 Date Routines	Chapter 8 DialogShow/Callbacks	Chapter 9 Macro Examples	Chapter 10 Glossary

Main Topics In This Chapter

General Notes	Math and Comparison Characters	System Variables	All The Rest

This section lumps a bunch of macro stuff together into a more or less alphabetical table – the "stuff" is not differentiated as to "WordPerfect Product Commands", "PerfectScript Macro Program Commands", "System Variables" or other "true" categories a real guru might use. "Common Persons" don't care about such things – they only want to know what commands to use, why, and how – so that their desired end result is obtained. For more specific identification and classification of the items listed here, consult one or more of the sources identified in the Additional Resources section of this paper. And, as said earlier, by no means are all macro commands listed and described here – consult the Additional Resources for many other macro commands and procedures. But, if you learn what you need to know from the following, you will be able to write both exceptionally simple and enormously complex macros, without learning anything else. And, note that the "syntax" descriptions are not necessarily the "preferred" method of macro writing – again, I'm interested in keeping things as simple as possible, and omitting any and all particulars that I've not found to be significant insofar as the "end result" is concerned. I don't see the point of using unnecessary keystrokes.

Important Note: No single line of macro code may exceed 512 keystrokes, spaces included. If more than 512 keystrokes are involved in a particular sequence, use HardReturns or other commands to break the component into a "legal" keystroke limit. In this regard, see this General Note, below.

The following table is organized as follows: 1) General Notes; 2) Math and Comparison Characters; 3) System Variables; and 4) All The Rest.

Command/Routine	Wp Ver	Syntax/Usage and Examples
General Notes These section explains various "general" items which are comprehensively applicable to understanding this manual, its limitations, and macro syntax I'll be using. Click the above Table Title to return to the start of this chapter. Click a following link to move to that part of this Chapter: Math and Comparison Characters, System Variables, or All The Rest. Other links: Top, Contents, Index.		

The term "Boolean Value"	All	"Boolean" is an adjective denoting a system of notation used to represent logical propositions by means of the binary digits 0 (false) and 1 (true). The term is named after English mathematician George Boole (d. 1864).
		A boolean value is *True* (for practical purposes, the same as 1) or *False* (for practical purposes, the same as 0).
		Sometimes, WordPerfect macros return a value with the non-string word *True* or *False*, and sometimes WordPerfect macros return a value with the numeric 1 or 0. That strikes me as kind of sloppy, but it means the same thing. If you're a compulsive type, try this code:
		x=QuickCorrectQuickBulletsQry // obsolete in Wp10 but it will compile
		Messagebox(;"x";x) // x will equal 1
		x=MacroIsCompiled("allfonts.wcm") //if Shipping Macros are installed

Command/Routine	Wp Ver	Syntax/Usage and Examples

		Messagebox(;"x";x) // x will equal *True*
		If(x=1) Messagebox(;"1 is same as True";"") EndIf

General Note About the := sign	Wp8 & higher	:= is the "official" way to assign a variable a value. Beginning with WordPerfect 6.1, a simple "equal" sign works just as well. For example, in the "multiplication" command which follows, the "official" way to use the same would be: var := numeric * numeric If you are writing macros for WordPerfect, I'm aware of no reason that you need to include the ":=" element ... just plain "=" works just fine. So, var=numeric*numeric works every bit as well as var := numeric*numeric. For this "Common User's Manual", I'm thinking that "less is more" and I've not used the preferred ":=" syntax/language. Certainly, no problems exist in doing equations in the ":=" structure. But, I'm aware of no benefits from doing so, either. So, I don't use ":=" in this manual.
General Note Rationale for including or excluding stuff in this book	All	This table is a subset of the various available macro commands. The Common Macro User won't need to know, or use, the full panoply of WordPerfect Macro commands. Those described here are (1) those used to create my own Grande Macros program for Oklahoma lawyers, including principles associated with the very complex child support computation macros; and (2) a few others. A Common Macro User doesn't need to know even 10% of the various macro commands to write very complex WordPerfect Macros. The commands/procedures described here should equip you well in that regard. If you want or need commands other than those described here, see Additional Resources in this manual.
General Note Parameters and Use of ()	All	() is used either to enclose an expression – e.g., If (a > b) ... EndIf – or to state any desired or required parameters (components) of a particular macro command – e.g., var=SubStr(var2;1;5). As noted below, the SubStr command extracts a substring from some other string, usually a variable which contains a string. 3 parameters are shown above: (1) the name of the variable from which the substring is to be extracted; (2) the beginning point of the string to be extracted; and (3) the ending point of the string to be extracted. In this manual, () is not used unless it is required. Not all macro commands have parameters. If a command either has no parameters, or has no required parameters, or has no permissive parameters that you want to use, no need exists to include () at the end of the particular macro command except to make clear to yourself and others that you are using a command as opposed to a variable or label.
General Note Use of "" for String Values	All	A pair of "regular" quotation marks the beginning and end of a text string. These marks must NOT be the "smart quotes" used in QuickCorrect's SmartQuotes feature. When you open a macro for editing using Tools, Macro..., Edit..., that action should turn off SmartQuotes. If it doesn't, you must turn SmartQuotes off manually: Tools, QuickCorrect..., SmartQuotes Tab, and uncheck all SmartQuotes options. A macro will not compile using SmartQuotes. Examples: Type("Old dogs can't learn new tricks.") ... if the quotation marks are regular, the macro will compile; if the quotation marks are SmartQuotes, the macro won't compile. FileOpen("C:\Grande4\CSupport.wpd") ... the identified file will open if regular quotation marks are used, but will result in a compilation error if SmartQuotes are used. See ?QuickCorrect and the various QuickCorrect items, below.

Command/Routine	Wp Ver	Syntax/Usage and Examples
General Note Don't Use "" for numeric, boolean or other non-string values	All	Don't use paired quotation marks (see above note) for non-string values in typing macros. The paired quotation marks indicates a string (regular text). No quotation marks indicates (as appropriate) a numeric, boolean or non-string parameter value of some sort. Example: Var="3 / 500" and Var=3 / 500 ... In the former, the value of Var is, literally, the text string shown. In the latter, the value of Var is the numeric quotient of the formula shown, 0.006.
General Note Use of VAR in this document	All	I've used "var" to indicate the name of a user-defined named variable in this chart. So, anytime you see something like, Var=?DateYear, the "Var" part means the name of the variable which could be any legally named Variable you determine. You can't use "Reserved Words" to name a variable, and other limitations are also present. See Reserved Words, Variable's name, and Variables, User Defined.
General Note Use of Upper/Lower Case	All	When you see commands such as SubStr, you may wonder: Is the use of upper/lower case important? Answer: No. Substr, SubSTR, substr, or any other combination of case would work just fine. The use of case gives a visual reminder of what the macro command does, but, in fact, when typing in the command's name, case doesn't matter.
General Note Length of Single Code Line	All	One single "line" may contain up to 512 characters before encountering a hard return, but not more. If a need exists that a variable contain more code than that, you can use concatenation to accomplish that, e.g., where variable "var" contains a string and variable "x" contains a string, var=var+x combines them both. Note: commands can break across lines.
General Note Reserved Words	All	You should NOT use a macro command's name or certain other "reserved" words to name a variable, a label, or in some other procedure that you may define. A list of such reserved words is in your on-line Macro Help file. In WordPerfect 10, these words are shown as "reserved": Address, And, Ansistring, Appactivate, Appexecute, Applocate, Application, Assert, Assertcancel, Asserterror, Assertnotfound, Assign, Beep, Bool, Break, Call, Cancel, Cancelcondition, Canceloff, Cancelon, Case, Caseof, Centimeters, Chain, Charlen, Charpos, Continue, Cton, Ddeexecute, Ddeexecuteext, Ddeinitiate, Ddepoke, Dderequest, Ddeterminate, Ddeterminatealldeclare, Default, Defaultunits, Digit, Discard, Div, Dllcall, Dllfree, Dllload, Dword, Else, Endapp, Endfor, Endfunc, Endif, Endifplatform, Endproc, Endprompt, Endswitch, Endwhile, Error, Errorcondition, Erroroff, Erroron, Errornumber, Exists, False, File, For, Foreach, Fornext, Fraction, Function, Getnumber, Getunits, Getstring, Global, Go, Hiword, If, Ifplatform, Inches, Indirect, Inputinteger, Label, Length, Letter, Local, Loword, Menu, Menulist, Millimeters, Mod, Nest, Next, Newdefault, Not, Notfound, Notfoundcondition, Notfoundoff, Notfoundon, Ntoc, Numstr, Oemstring, Off, On, Oncancel, Onddeadvise, Onerror, Onnotfound, Or, Pause, Persist, Persistall, Points, Position, Procedure, Prompt, Prototype, Quit, Realrepeat, Return, Returncancel, Returnerror, Returnnotfound, Run, Sendkeys, Speed, String, Strlen, Strnum, Strpos, Strunit, Subchar, Substr, Switch, Tolower, Toupper, True, Unitstr, Until, Use, Value, Varerrchk, Varerrchkoff, Varerrchkon, Void, Wait, While, Word, Wpstring, Wpunits, Wp1200Ths, Xyz, Xor. To avoid accidental use of a reserved word, use a character such as "v" to begin the name of a Variable or Label, e.g., vQuit or Label(vRun).

Command/Routine	Wp Ver	Syntax/Usage and Examples
Math and Comparison Characters This is not a comprehensive list of all possible math and/or comparison features. For more information than is shown here, consult your on-line Macro Help file or some other book or manual. I've only included items I consider likely to be essential, sooner or later. Click the above Table Title to return to the start of this chapter. Click a following link to move to that part of this Chapter: General Notes, System Variables, or All The Rest. Other links: Top, Contents, Index.		
*	All	An operator; Usage: var=numeric*numeric Assigns var the result of multiplication of 2 numbers; an error will result if one/both numbers are not really "numbers", or cannot be converted to numeric values via automatic data conversion, discussed in Chapter 6 (not available in WordPerfect 6.1). If using variables, each variable must contain a numeric value or be capable of being converted to a numeric value, e.g., if var1=2500 and var2=1.5, var3= var1 * var2 assigns var3 a value of 3750; but if var2 contains a string value which cannot be converted to a numeric value (e.g., var2="$1.5"), error will result since a "dollar sign" is not a number. Only +-1234567890 and "period" (decimal) are acceptable. If a variable contains a string, you may either rely upon automatic data conversion, discussed in Chapter 6, or create an error-trapping routines using OnError, then attempt to convert the string to a numeric value using StrNum. See Chapter 6, Math Routines, for examples. Also, see concatenation.
/	All	An operator; Usage: var=numeric / numeric Assigns var the result of division of 2 numbers; an error will result if one/both numbers are not numeric values or cannot be converted to numeric values by automatic data conversion discussed in Chapter 6 (not available in WordPerfect 6.1) or by an error-trapping routine which tests the value to insure that it can be converted to a numeric value, and, then does so if it is. If using variables, each variable must contain a numeric value or be capable of conversion to a numeric value, e.g., if var1=2500 and var2=1.5, var3= var1 / var2 assigns var3 a value of 1666.6666[to 15 decimals]; but if var2 contains a string value which cannot be converted to a numeric value (e.g., var2="$1.5"), error will result since a "dollar sign" is not a number. Only +-1234567890 and "period" (decimal) are acceptable. If a variable contains a string, you may either rely upon automatic data conversion, discussed in Chapter 6, or create an error-trapping routines using OnError, then attempt to convert the string to a numeric value using StrNum. See Chapter 6, Math Routines, for examples. Also, see concatenation.

+	All	An operator; Usage: (1) var=numeric + numeric or (2) var=string+string
		(1) Numeric: Assigns var the result of addition of 2 numbers; if all values are not numeric or cannot become numeric via automatic data conversion, discussed in Chapter 6 (not available in WordPerfect 6.1), or by some other conversion routine, concatenation will occur IF one value is a string which is not a number (see 3rd example, below). IF one value is numeric and the other is a string which IS a number, the string value will be treated as a number (see 2nd example, below). If it is intended that all values be treated as numbers, be sure that's so, perhaps using automatic data conversion or an error-trapping routine using OnError, then attempt to convert the string to a numeric value using StrNum. See Chapter 6, Math Routines, for examples. Only +-1234567890 and "period" (decimal) are acceptable.
		(2) String: The values will be combined. Also, see concatenation.
		Examples:
		x=500 y=1 z=x+y Result: z=501 (numeric value)
		x=500 y="1" z=x+y Result: z=501 (numeric value)
		x=500 y="$1" z=x+y Result: z=500$1 (string value)
		x="abc" y="def" z=x+y Result: z=abcdef (string value)

Command/Routine	Wp Ver	Syntax/Usage and Examples
-	All	An operator; Usage: (1) var=numeric - numeric or (2) var=string-string (1) Numeric: Assigns var the result of subtraction of 2 numbers; if all values are not numeric or cannot be converted to numeric values using automatic data conversion, discussed in Chapter 6 (not available in WordPerfect 6.1), or some other conversion means, a reduction will occur (see next row). If it is intended that all values be treated as numbers, be sure that's so, perhaps using automatic data conversion or an error-trapping routine using OnError, then attempt to convert the string to a numeric value using StrNum. See Chapter 6, Math Routines, for examples. Only +-1234567890 and "period" (decimal) are acceptable. Also, see Chapter 6, Floating Cell Problem. (2) String (reduction): the opposite of concatenation. The term "reduction" was introduced to me by J. Dan Broadhead in the context of string subtraction as being the "opposite of concatenation" where string values are used in corresponding variables. The term "reduction" is not used in any Macros Help file but the concept is described in PerfectScript macros help under "-" (minus sign). The context is var3=var1-var2. Reduction became available in WordPerfect 7.0. According to J. Dan: If either operand is a *true* string (not just a numeric value enclosed in "", but a true string value like "abc"), then the opposite of concatenation is used (called "reduction"). In string reduction, the result of removing the first occurrence of the second string from the first string is returned. var1="abcdef" var2="bcd" var3=var1-var2 will place "aef" into var3. But, if var3 = "xyz" instead of "bcd", var3-var1-var2 assigns var3 "abcdef" since "xyz" was not present in var1. See Contatenation, below, and examples of reduction in Chapter 6.
=	All	Usage: var=any = any Assigns var a boolean (true/false) value if two values (variables or otherwise) are exactly equal (including case, if string values are involved). Most often, the equal sign will be used otherwise than shown above. Instead, a common usage is: If(varA=varB) [do this] Else [do that] EndIf Either string, numeric, or Boolean values may be compared.
variable = or variable :=	All	Usage: as shown in the left column. The equals sign is commonly used to assign the value of a variable to some other value, which may be a string or numeric value. Examples: var=var5 var=5 var="Father" Note that such assignments are exact and, for string values, the assignment is case sensitive. So, if var="Father", in the statement: If(var="father") Call(MaleP) Endif, Label(maleP) wouldn't be called. See <>, below. See Reserved Words, above, and/or Variables, below.
<> or !=	All	Usage: var=any <> any Assigns var a boolean (true/false) value if two values (variables or otherwise) are NOT exactly equal – and for string values, case is sensitive. Most often, the not equal sign will be used otherwise than shown above. Instead, a common usage is: If(varA <> varB) [do this] Else [do that] EndIf Either strings or numerical values may be compared. Examples if var1 = "Father" and var2 = "father":

Command/Routine	Wp Ver	Syntax/Usage and Examples
		If(var1=var2) MessageBox(;"Values are the same";"") Else MessageBox(;"Values are not the same";"") EndIf The 2nd message box will appear since the values are not identical. But, the following code effectively renders the compared values to be the same: If(ToLower(var1) = ToLower(var2)) MessageBox(;"Values are the same";"") Else MessageBox(;"Values are not the same";"") EndIf
<	All	Usage: var=any < any Assigns var a boolean (true/false) value if the 1st (left) item is less than the value of the 2nd (right) item. Most often, the less than sign will be used otherwise than shown above. Instead, a common usage is: If(varA < varB) [do this] Else [do that] EndIf Either string or numeric values may be compared. If strings are compared, True is returned if the left value alphabetically precedes the right value.
<=	All	Usage: var=any <= any Assigns var a boolean (true/false) value if the 1st (left) item is less than or equal to the value of the 2nd (right) item. Most often, the less than or equal to sign will be used otherwise than shown above. Instead, a common usage is: If(varA <= varB) [do this] Else [do that] EndIf Either string or numeric values may be compared. If strings are compared, True is returned if the left value alphabetically precedes or is identical (including case) the right value.
>	All	Usage: var=any > any Assigns var a boolean (true/false) value if the 1st (left) item is greater than the value of the 2nd (right) item. Most often, the greater than sign will be used otherwise than shown above. Instead, a common usage is: If(varA > varB) [do this] Else [do that] EndIf Either string or numeric values may be compared. If strings are compared, True is returned if the right value alphabetically precedes the left value, e.g., in c="b">"a" Messagebox(;"c";c) //c=True
>=	All	Usage: var=any >= any Assigns var a boolean (true/false) value if the 1st (left) item is greater than or equal to the value of the 2nd (right) item. Most often, the greater than or equal to sign will be used otherwise than shown above. Instead, a common usage is: If(varA >= varB) [do this] Else [do that] EndIf Either string or numeric values may be compared. If strings are compared, True is returned if the right value is equal to or alphabetically precedes the left value, including case, e.g., c="b">"a" Messagebox(;"c";c) //c=True.

Command/Routine	Wp Ver	Syntax/Usage and Examples
System Variables System variables report various types of information to you about your "system", from date and time information, to the particular state that WordPerfect is in within some particular context. Those which follow are a small subset of the full schedule of System Variables. See your on-line Macros Help or more comprehensive resources than this manual for such a list. Click a following link to move to that part of this Chapter: General Notes, Math & Comparison Characters, or All The Rest. Other links: Top, Contents, Index.		
?DateDay	All	Usage: var=?DateDay Assigns var the numeric value of the current day, based on your computer's clock. See other ?Date... commands immediately following and various Date... commands, and Chapter 7, Date Routines.
?DateMonth	All	Usage: var=?DateMonth Assigns var the numeric value of the current month, based on your computer's clock. Click here for the 1st ?Date... variable; and, see various Date... commands, and Chapter 7, Date Routines.
?DateWeekDay	Wp 8 & higher	Usage: var=?DateWeekday Assigns var the numeric value of the current day of the week. Click here for the 1st ?Date... variable; and, see various Date... commands, and Chapter 7, Date Routines.
?DateYear	All	Usage: var=?DateYear Assigns var the numeric value of the current year, based on your computer's clock. Click here for the 1st ?Date... variable; and, see various Date... commands, and Chapter 7, Date Routines.
?DisplayMode	All	Usage: var=?DisplayMode Assigns var the numeric value of the current display mode: 2 (Draft view); 3 (Page view); 4 (2 page view). Also, See DisplayMode() and ?Zoom, Zoom and DisplayZoom(), below.
?DocBlank	All	Usage: var=?DocBlank Assigns a Boolean True/False value to var whether the currently active WordPerfect document is completely "blank" or not. Note that a space, even if deleted, in an open document makes it "not" blank". This command is useful in knowing whether or not a new blank document might need to be opened before begin to write to the document – i.e., you don't want to start writing a new letter or court document in a file that already has something in it. So, if the above command results in var having a value of False, you might want to follow with a FileNew command, assuming that can be done (only 9 WordPerfect documents can be open at the same time). See ?NumberOpenDocuments and FileNew, below.
?EndNote	All	Usage: var=?EndNote Assigns a numeric value to var, that being the number of the endnote to the left of the insertion point. Also, see ?Footnote, below.
?Font	All	Usage: var=?Font Assigns a "string" value to var, that being the currently active font at the location of the insertion point. If, for one reason or another, you have been changing fonts in a document being written, this command can instruct the macro to change the font if the font isn't what you want at the insertion point. In that event, follow this command with some other Font() command, below. See ?FontSize and FontSize, below.
?FontSize	All	Usage: var=?FontSize Assigns a measurement value in WordPerfect units (1200ths of an inch) to var of the font size at the insertion point; if the size isn't what you want, you'd want to follow up this command with a FontSize() command. Also, see ?Font, above, and Font, below.

Command/Routine	Wp Ver	Syntax/Usage and Examples
?Footnote	All	Usage: var=?Footnote Assigns a numeric value to var of the footnote that immediately precedes the insertion point. See Numeric Values. This may be useful in performing various operations on a document that contains footnotes. Also, see ?EndNote, above.
?InTable	All	Usage: var=?InTable Assigns a Boolean (True/False) value whether the insertion point is in a table or not.
?Justification	All	Usage: var=?Justification Assigns var the numeric value of the justification at the insertion point: 0 (Left), 1 (Full), 2 (Center), 3 (Right), 4(Full all), 5 (Decimal). Also, see Justification(), below.
?LeftChar	All	Usage: var=?LeftChar Assigns var the string value of the character immediately to the left of the insertion point.
?LeftCode	All	Usage: var=?LeftCode Assigns var the numeric value of the code to the left of the insertion point. See ?RightCode.
?LineSpacing	All	Usage: var=?LineSpacing Assigns var the numeric value representing the distance between lines of text at the insertion point. Also, see LineSpacing(), below.
?MajorVersion	All	Usage: var=?MajorVersion Assigns var the numeric value of the major version number of the WordPerfect program you are using. Wp6.1 (6), Wp7 (7), Wp8 (8), Wp9 (9), Wp10 (10), Wp11(11), Wp12(12), but, NOTE: In the initial release of WordPerfect 10, this command erroneously returns a value of 9. It was fixed in WordPerfect 10's Service Pack 2. Also, see VersionInfo() and MacroInfo, below.
?Name	All	Usage: var=?Name Assigns var the string value of the active filename.
?NumberOpenDocuments	All	Usage: var=?NumberOpenDocuments Assigns var the numeric value of the number of currently open WordPerfect documents. This command is useful when testing to determine if another document can be opened, as follows: x=?NumberOpenDocuments If(x=9) [dialog or message box informing that another document cannot be opened] Else [FileNew] [FileOpen()] EndIf
?Path Commands	All	Various ?Path... commands exist to identify the string value of various WordPerfect default and other directories. Some are identified below, but others exist. Consult your on-line macro help file for more.
?Path	All	Usage: var=?Path Assigns var the string value of the path of the currently active document without the filename, if it has been saved. If the active document is new and has not been saved, the value is returned as empty, "". The variable can be used with various commands, such as FileOpen() and FileInsert().
?PathCurrent	All	Usage: var=?PathCurrent Assigns var the string value of the current directory. The variable can be used with various commands, such as FileOpen() and FileInsert().

Command/Routine	Wp Ver	Syntax/Usage and Examples
?PathDocument	All	Usage: var=?PathDocument Assigns var the string value of the default documents directory, as set up in Tools | Settings | Files (Wp8 & higher) or Edit | Preferences | Files (Wp6.1 or 7.0). The variable can be used with various commands, such as FileOpen() and FileInsert().
?PathMacros	All	Usage: var=?PathMacros Assigns var the string value of the default macro directory, as set up in Tools | Settings | Files (Wp8 & higher) or Edit | Preferences | Files (Wp6.1 or 7.0). The variable can be used with various commands, such as FileOpen(), FileInsert(), Run().
?PathMacrosSupplemental	All	Usage: var=?PathMacrosSupplemental Assigns var the string value of the default supplemental macro directory, as set up in Tools | Settings Files (Wp8 & higher) or Edit | Preferences (Wp6.1 or 7.0). The variable can be used with various commands, such as FileOpen(), FileInsert(), Run().
?PathSpreadsheet	All	Usage: var=?PathSpreadsheet Assigns var the string falue of the default spreadsheet directory, as set up in Tools | Settings | Files (Wp8 & higher) or Edit | Preferences | Files (Wp6.1 or 7.0). The variable can be used with various commands, such as FileOpen() and FileInsert().
?QuickCorrect	All	Usage: var=?QuickCorrect Assigns var the Boolean (True/False) value as to whether QuickCorrect is turned on or off. See various QuickCorrect items, below.
?RightChar	All	Usage: var=?RightChar Assigns var the string value of the character to the right of the insertion point. See example in Chapter 9.
?RightCode	All	Usage: var=?RightCode Assigns var the numeric value of the code to the right of the insertion point. See ?LeftCode & J. Jeppson's list of ?RightCodes on the Internet. See example in Chapter 9.
?Row	All	Usage: var=?Row Assigns var the numeric value of the current row number in a table.
?SelectedText	All	Usage: var=?SelectedText Assigns var the string value of the currently selected text, or no value if no text is selected. See various Select... commands, below.
?UnderlineSpaces	All	Determines whether UnderlineSpaces is currently on or off. See UnderlineSpaces(), below. Usage: var=?UnderlineSpaces assigns var the boolean value of 1 if it is on or 0 if it is not on. See UnderlineSpaces(), below.
?UnderlineTabs	All	Determines whether UnderlineTabs is currently on or off. See UnderlineTabs(), below. Usage: var=?UnderlineTabs assigns var the boolean value of 1 if it is on or 0 if it is not on. See UnderlineTabs(), below.
?Zoom	All	Returns the current zoom as a numeric percentage between 25-400. Usage: var=?Zoom See ?DisplayMode, above, and DisplayZoom and DisplayMode, below

Command/Routine	Wp Ver	Syntax/Usage and Examples
All The Rest In this section of the table, I'm dumping every other command, etc., that I intend to include that hasn't already been included above. As with the System Variables section, this is but a small subset of all the possible commands and procedures you can use. Note that the following table does NOT differentiate between "PerfectScript" and "WordPerfect Product" commands – a "Common Macro Person" wouldn't care about or be aware of the distinction but would tend to think that ALL of WordPerfect's available macro commands are simply, "WordPerfect". You should consult your on-line Macro Help file or some other resource for a listing and description of other commands not listed. Click a following link to move to that part of this Chapter: General Notes, Math & Comparison Characters, System Variables. Other links: Top, Contents, Index.		
/* <comment text> */	Wp 8 & higher	Usage: as shown in left column; the /* marks the beginning of a non-executing comment you decide to place in a macro and the */ marks the end of the comment. Unlike "regular" comments, below, a hard return does not end the comment and complete sections of a macro can be "commented out" using this approach.
//	All	Usage: as shown in the left column; the // at the beginning of a comment continues until the next hard return, except that this type of comment cannot exceed 512 characters, as is true for any single line of code.
AND	All	Usage: var=boolean AND boolean Assigns var the boolean (True/False) value if all the compared expressions are True, False if not. Commonly, the AND coupling statement is not used as shown above. Usually, the purpose is to add conjunctive statements, all of which must be True, before commands which follow will be executed. For example, if both statements in the following are True, then the [do this] commands will be executed; but, if not, the [do that] statements will be executed: If(var1>500 AND var2=50) [do this] Else [do that] EndIf
OR	All	Usage: var=boolean OR boolean Assigns var the True boolean (True/False) value if any of the of the compared expressions are True, or, if not, the False boolean value. Commonly, the OR coupling statement is not used as shown above. Usually, the purpose is to add alternative conjunctive statements, at least one of which must be True before commands which follow will be executed. For example, if any one statement in the following is True, then the [do this] commands will be executed; but, if not, the [do that] statements will be executed: If(var1>500 OR var2=50 OR var3=25) [do this] Else [do that] EndIf
Appearance attributes	All	For appearance attributes, e.g., Bold, see Attribute commands, below.
Application()	All	When you create a keyboard macro, the macro created will always begin with the "Application" statement identifying for the macro compiler the application which will use the macro. Even though the "Application" statement is not required, I recommend that you always use it and locate it at the very top of your macro. For WordPerfect 6.1 and 7.0 USA editions, the Application statement would read (if you created a macro from your keyboard): Application(A1; "WordPerfect"; Default; "US") For WordPerfect 8.0 & higher, if you create a macro from the keyboard, the statement would read: Application (WordPerfect; "WordPerfect"; Default!; "EN") With WordPerfect 8, Corel abandoned separate "English" versions for various English-speaking versions and opted for a single English version – hence the change from US to EN as the last element of the statement. By way of further explanation, J. Dan Broadhead notes that: The "A1" thing is a bit misleading. The first parameter can be anything you want it to be. It is a prefix that you can use later in the macro in front of commands (followed by

Command/Routine	Wp Ver	Syntax/Usage and Examples
		a ".") for that application. In the old macro recorder, it just recorded this prefix as A1, then A2, then A3, etc., for every application in the macro. Starting in 8 (??), the macro recorder used more readable names like WordPerfect, but A1, A2, etc., can still be used if you want in 8-10. Version 6 (and I believe 7) limited this first parameter to 2 characters, but you could put WP instead of A1 if you wanted to (WP7 may have allowed the longer names like 8-10). The "!" following Default is optional in 8, 9 and 10. You can either leave it off or specify just Default, or include it and specify Default!. Either is fine. So the only item that won't work the same in all versions, is the "US" vs. "EN" thing. And, from at least WP7 on, this last parameter is optional and can be left off (I don't have the WP6 manuals here to check). So you could do this and it would work in at least 7-10: Application (WP; "WordPerfect"; Default) In fact, even Default is optional, so this would work too: Application (WP; "WordPerfect")
Arrays	All	See Declare(), below.
Assert()	All	Assert creates, or simulates, a condition, such as a Cancel condition. This manual does not develop the Assert command particularly, but see examples in Chapter 4 and Chapter 8. Usage: Assert(enum), e.g., Assert(CancelCondition!) or Assert(ErrorCondition!) or Assert(NotFoundCondition!). See WordPerfect Macro Help for more information. Also, see OnCancel(), OnError(), and OnNotFound(), below.
AttributeAppearanceOff () AttributeAppearanceOn () AttributeAppearanceToggle ()	All	Turns off/on the following format codes, which you'd include within the (): Bold!; DoubleUnderline!; Every!; Italics!; Outline!; Redline!; Shadow!; SmallCaps!; Strikeout!; Underline! Separate multiple attributes with semicolon(s): AttributeAppearanceOff(Bold!) AttributeAppearanceOn(Bold!;Italics!) The "...Toggle" command is the same, except that it works like a light switch – turns lights on, if already off; turns lights off, if already on. Example: AttributeAppearanceToggle (Underline!)
AttributeNormal	All	Usage: As shown in the left column; turns off all current font attributes except color.

AttributePosition()
AttributePositionToggle()
	All	Use as shown in the left column, adding one of the following:
		NormalPosition! or Subscript! or Superscript!
		Example: AttributePosition(Superscript!)
		The "...Toggle" command is the same, except that it works like a light switch – turns lights on, if already off; turns lights off, if already on. Example: AttributePositionToggle (Normal!)

AttributeRelativeSize()
AttributeRelativeSizeToggle()
	All	Use as shown in the left column, adding one of the following:
		ExtraLarge! or Fine! or Large! or NormalSize! or Small! or VeryLarge!
		The "...Toggle" command is the same, except that it works like a light switch – turns lights on, if already off; turns lights off, if already on. Example: AttributeRelativeSizeToggle (Large!)

Command/Routine	Wp Ver	Syntax/Usage and Examples
Average()	Wp 8 & higher	Usage: var=Average (numeric values separated by semicolons) Assigns var the average numeric value of numeric items contained within the expression. If one of the averaged values is not a numeric value, an error will occur. Example: var=Average (2; 4; 6; 8) – the value of var is 5.0 Example: var=Average (32.53333; 59.122334567; 33.33; 49.22) – the value of var is 43.55141614175 You can use other commands to round the result if you wish, e.g., var=Average (32.53333; 59.122334567; 33.33; 49.22) var=NumStr(var;2) – the string value of var is 43.55; note that you can reconvert this value to numeric by var=StrNum(var).
Beep	All	Usage: As shown in left column, unless you wish to add parameters to identify the "sound" of the "beep" (as matching such sounds in your Windows, Control Panel, Sounds defaults), in which case, the command could include one of the following optional parameters: Default! (used if no parameter is specified) or Question! or Asterisk! or Exclamation! or CriticalStop! So, with a parameter, the statement might read: Beep(Question!); without a parameter, just use Beep, without the (). Commonly, the Beep command is used to call a user's attention to something important, such as an error running the macro or the completion of its running, or whatever.
BlockProtect()	All	Usage: BlockProtect(On!) and/or BlockProtect(Off!) If you are writing a document with a macro and you want to insure that particular text remains on the same page or column, use BlockProtect (On!) to mark the beginning of such text and BlockProtect (Off!) to mark its end.
BookmarkCreate(string)	All	Usage: Same as shown in left column <another parameter, selected – Yes! No! – is also available but I've found no practical reason to use it>. This command creates a named bookmark in a document. In "document assembly" type macros, it can be useful to add a bookmark you can later "go to" in the macro and perform some operation after the bookmark. You may or may not want to delete a bookmark after it has served its purpose by using BookmarkDelete. For an example, see ConvertFE.wcm.
BookmarkDelete(string)	All	Usage: Same as shown in left column. This deletes a previously defined Bookmark, where "string" is the name of the bookmark.
BookmarkFind(string)	All	Usage: Same as shown in the left column. This moves the insertion point to immediately following the named bookmark.
Box Commands	All	No Box commands are described in this paper, but are generally noted here just so you'll know that several such commands exist. Consult your on-line Macro Help file or other resources.
Call(Label) [same for all versions]		Usage: Same as shown in the left column. This one's a "biggie", one which you must master and use as needed. As noted earlier, a macro's movement is "lineal", top to bottom of the macro document's layout, unless or until some other command interrupts that flow. This is one of such commands, one you will use frequently. A "Call" statement sort of "sucks up" the commands contained in the "called" Label before proceeding with its general lineal movement. The Label "called" needs a Return command at its end, meaning, the "call" is done – go back from whence you were called. *If a Return command is not present,* the macro flow will NOT "go back" but will instead proceed lineally at and after whatever is contained in the "called" Label. Here's a simple example: -------------- Type("This line of type is being typed into a new document.") Call(H) Type("This is the second line of type being typed into the same document.") Call(H) Quit

Command/Routine	Wp Ver	Syntax/Usage and Examples
	Label(H) HardReturn HardReturn Return Type("Donald Duck is dumb.") ------------- In this very simple example, after executing the 1st "Type" statement wherein a line of text was "typed", the Call(H) command called (sucked up) the commands contained at Label(H), in this case, 2 hard returns. The "Return" command was encountered, which ended the "call". Macro operation then proceeded normally (lineally) and macro commands immediately following the "Call" command were then executed: first, the 2nd "Type statement shown above, was done; second, another Call(H) statement was encountered, so that, after the 2nd "typing" was done, the 2 HardReturns ordered at Label(H) were again executed. Note: If the "Return" command did not follow the 2 HardReturn commands, the Type command immediately following the 2 HardReturns would have been typed, "Donald Duck is dumb." The point: include the Return command at the end of the called Labels. Otherwise, the macro flow will not be "returned" to the point of the "call". Note: When using a "call" command, it is not necessary to literally type "Call(H)" (for example). Simply typing H would have done the same thing. So that ... Type("This line of type is being typed into a new document.") H Type("This is the second line of type being typed into the same document.") H ... would produce identical results to the more descriptive statements included above. Note: Various forms of the "call" methodology/principle appear in many different macro routines, most notably including "Callback" routines discussed in Chapter 8. It is essential to your writing macros with any complexity at all that you "understand" the principles stated here. If you don't, you'll simply not be able to write macros which are as complex as you'd like them to be. Understanding "call" principles is of very great importance. See Chapter 4. Note: Call routines can call macro code associated with "Procedures" or "Functions", as well as "Labels". "Procedures" are beyond the scope of this manual and "Functions" are developed only slightly in Function-EndFunc, below, and in a few examples referenced there.	
Callback function	All	Callback functions, which are effectively "loop" routines which continue until some control or command stops the loop from executing, may be included in See DialogShow() [not using CallbackWait CallbackResume], DialogShow() [using CallbackWait CallbackResume] and/or Chapter 8, Callback Routines.
CallbackWait CallbackResume	Wp7.0 & higher	These commands are applicable to DialogShow() using a Callback function in WordPerfect 7.0 & higher. For some discussion about their use, see DialogShow() [using CallbackWait and CallbackResume, below. And, see Chapter 8, Callback Routines.
Case Manipulation	All	See ToLower(), ToInitialCaps(), and ToUpper(), below.
CaseOf	All	Conditional statement used in Switch() CaseOf EndSwitch, below.
Center	All	Inserts a [HdCenteronMarg] code at the insertion point, centering the current line of text, e.g., Type("Chapter One") Center Type("Part B") FlushRight DateCode HardReturn
Chain()	All	Chain("MacroName") starts another macro once the parent macro ends and macro flow does not return to the parent macro once the chained macro is done when a Return command is encountered in the chained macro. The Chain() command may exist anywhere in the parent macro but it does not execute until the parent macro ends. On-line macro help says that, "If the current macro contains more than one Chain command, only the macro file in the last Chain command is executed when the current macro ends." I don't use the Chain command since I've never found a good reason to do so and I use Run() instead. If you want more information about Chain, see on-line macro help or Additional Resources. See Run() for complete syntax information, which is the same as it is for Chain.

Command/Routine	Wp Ver	Syntax/Usage and Examples
CloseNoSave	All	Closes the current document without saving. You can add a No! or Yes! parameter to prompt a user to be sure, if you want. If no parameter is specified, no prompt is presented. Examples: CloseNoSave or CloseNoSave(Yes!)
Column Commands	All	No Column commands are described in this paper, but are generally noted here just so you'll know that several such commands exist. Consult your on-line Macro Help file or other resources.
Comment Commands	All	No Comment commands (creating, editing, etc., comments in a regular WordPerfect document) are described in this paper, but are generally noted here just so you'll know that several such commands exist. Consult your on-line Macro Help file or other resources.
Concatenation	All	A technique, not a command. Concatenation joins elements to make a combined value. See use of +, above, and Type(), below, and discussion/examples in Chapters 5 and 6, working with decimals. Also, see Run() and Chain(). Also, see Reduction.
Copy	All	Copies selected text to the clipboard. See EditCopy, below
DateCode DateText DateFormat(string)	All	DateCode inserts a date "code" at the insertion point, e.g., January 1, 2002 (or however you may have defined the code with DateFormat). The "date" is updated based on the computer clock. DateText inserts a static "text" date at the insertion point. DateFormat is used to change the presentation of the date and would normally be used before a DateCode or DateText command. Numerous possibilities exist, but, for example, to make text you are writing to a document read, "Now, on this __ day of January, 2002," where January is the current month and 2002 is the current year (based on your computer's clock), do this: DateFormat ("___ day of Month, Year(4)#") NOTE: What appears above for "Month" and "Year(4)#" is NOT typed in text. Rather, it is code inserted into the macro using the Codes... button located on the Macro Toolbar. See Macro Toolbar Codes button in Chapter 5 for information on using this feature. Also, see various ?Date... system variables, above, and Chapter 7, Date Routines.
Date... Commands	Wp7 & higher	Other Date... commands are available in WordPerfect 7.0 & higher, as noted below, but are not particularly covered in this manual: DateAddDays (WordPerfect 8.0 & higher) (see examples in Math.wcm) DateAddMonths (WordPerfect 8.0 & higher) (see examples in Math.wcm) DateAddWeeks (WordPerfect 8.0 & higher) (see examples in Math.wcm) DateAddYears (WordPerfect 8.0 & higher) (see examples in Math.wcm) DateAndTime (WordPerfect 7.0 & higher) (see examples in Math.wcm) DateDay (WordPerfect 7.0 & higher) DateDayOfYear (WordPerfect 8.0 & higher) DateDaysInMonth (WordPerfect 8.0 & higher) DateDaysInYear (WordPerfect 8.0 & higher) DateIsLeapYear (WordPerfect 8.0 & higher) DateMonth (WordPerfect 7.0 & higher) DateMonthName (WordPerfect 7.0 & higher) DateOfMonthEnd (WordPerfect 8.0 & higher) DateOfNthDay (WordPerfect 8.0 & higher) DateOfNthWeek (WordPerfect 8.0 & higher) DateOfNthWeekday (WordPerfect 8.0 & higher)

Command/Routine	Wp Ver	Syntax/Usage and Examples
		DatePart (WordPerfect 8.0 & higher) DateString (WordPerfect 7.0 & higher) DateWeekday (WordPerfect 7.0 & higher) DateWeekdayName (WordPerfect 7.0 & higher) DateWeekOfYear (WordPerfect 8.0 & higher) DateYear (WordPerfect 7.0 & higher) See Chapter 7, Date Routines.
DDE Commands	All	DDE (Dynamic Data Exchange) commands are not included in this paper and are noted here so you'll know they exist. Consult your on-line Macro Help file or other resources.
Declare var[num]	All	Declare is used to create a local variable or an array. As a practical matter, when you create a variable, you "declare" it and Declare is unnecessary to create a variable in the usual context. Declare is discussed here in the context of creating "arrays". The discussion here is limited to one or two dimensional arrays, even though ten dimensions are possible. Essentially, declare creates a local variable with multiple elements, an array, rather like a table of values, horizontal rows and vertical columns. The number of elements in the array = the number of rows x the number of columns. The number of elements in each dimension can't exceed 32,767. Each element in the array can be assigned a value, and an element's value is "undefined" unless you do so. In the following, "var" is the variable's name. Usage if a one dimensional array: Declare var[num] Usage if a two dimensional array: Declare var[num1;num2] If an array is one dimensional, the number of rows (1) is implied – Declare var[1;3] is the same thing as Declare var[3], although it took me awhile to figure that out (Wp's on-line help about Declare is rather obtuse to me). Some examples are: Declare vCounty[5] – the number of elements is 5 (1 x 5) and the variables to assign values to are vCounty[1], vCounty[2], vCounty[3], vCounty[4] and vCounty[5], e.g., Oklahoma, Canadian, Cleveland, Pottawatomie and Logan counties. Declare vCounty[2;5] – the number of elements is 10 (2 x 5) and the variables to assign values to are vCounty[1;1], vCounty[1;2], vCounty[1;3], vCounty[1;4], vCounty[1;5], vCounty[2;1], vCounty[2;2], vCounty[2;3], vCounty[2;4] and vCounty[2;5]. In this example, the named counties could be in the 1st row, the corresponding county seat cities in the 2nd row. Don't be mislead by this example – an element can be assigned numeric values, formulas, etc., even though strings (text) is used here. Declare is useful in creating lists which will be used in dialogs in which selections will be made. Each element in the array has a value, empty if not particularly assigned, but each element can be assigned a value. See an example in Chapter 9.
DeleteCharNext DeleteCharPrevous DeleteToEndOfLine DeleteWord	All	Deletes the character (or, if something is selected, the selection – SelectDelete does the same) to the right of the insertion point. Deletes the character (or, if something is selected, the selection) to the left of the insertion point. Also, see SelectDelete, below. Deletes text and codes from the insertion point to the end of the line. Deletes the current word or space if the insertion point is not in a word.

Command/Routine	Wp Ver	Syntax/Usage and Examples
DialogAdd...	All	Numerous DialogAdd... commands are available to add controls to dialogs created by using DialogDefine() directly from your keyboard or to dialogs made in the Dialog Editor. This manual covers none of such commands except by a few unexplained examples. This manual completely focuses on adding controls within the Dialog Editor. See Chapter 5. Consult your WordPerfect on-line Macro help for information about DialogDefine() and the various DialogAdd... statements which are possible in that context. For examples, see DialogShow w/CallbackWait, MacroCompile, Copying a Dialog to Text in Chapter 5, and Sample RegionGet Code in Chapter 8.
DialogDefine()	All	DialogDefine() is used to create dialogs with literal macro code you enter from your keyboard. This manual develops no commands in that regard, including but not limited to DialogDefine(), but relies totally upon use of the Dialog Editor to create dialogs. For a simple and unexplained example, see DialogShow() [using CallbackWait], below. And, see Copying A Dialog To Text in Chapter 5 for another example. Consult WordPerfect on-line Macro help for information about this and related commands. To make dialogs using the Dialog Editor, see Chapter 5, and see 2 examples in Chapter 8: Example 1; Example 2.
DialogDestroy()	Wp7.0 & higher	Removes a Dialog from memory. The following comments relate to DialogDestoy being used with a Dialog Editor dialog. The use of DialogDestroy in DialogDefine dialogs is not covered in this manual, except in J. Dan Broadhead's comments in Chapter 8. WordPerfect 6.1: Don't use the command. WordPerfect 7.0: Don't use the command for non-callback dialogs; but, if DialogShow() includes a callback routine, you can use it as described below in "If using a dialog with a callback function". WordPerfect 8 & higher: Use the command when you are through with a dialog AND want it removed from memory. However, see Chapter 8 for much more discussion. Although WordPerfect's Macro help says that a dialog created with the Dialog Editor without a Callback is removed from memory when any button is pushed, that's not so. If an occasion arises that you WANT a dialog to be removed from memory, DialogDestroy() will do that. In Wp8.0 & higher, inclusion of a DialogDestroy() command following DialogShow() does not produce error. Usage. After the DialogShow() command, (a) if you need to know the value of the control which closes the dialog (e.g., "OKBttn", "CancelBttn", "OtherBttn") use the MacroDialogResult command, below, before using the DialogDestroy() command, or (b) if only one control (i.e., only one button can close the dialog such as an "OKBttn" control), you don't need to bother with using MacroDialogResult since its value doesn't matter. Example: DialogShow("vTest";"WordPerfect") x=MacroDialogResult DialogDestroy("vTest"); variable x will be the value of the button that closed the dialog (1 for an "OKBttn"; 2 for a "CancelBttn" or a Windows "Close" (x) button, or, otherwise, the string name of the control, e.g., "HelpBttn"). Then, you could write: If(x=1) [do this] Else [do that] EndIf. Unless the value of the control which closes the dialog is 2, if variables are associated with controls in the dialog (e.g., the dialog contains a list box control "B1" which is associated in the dialog with variable vB1), those variables remain declared (or updated) with the value of the control (e.g., "Yes" or "No" if the "B1" control includes "Yes" and "No" in its list). If using a dialog with a callback function: Same as above, except use the command afterDialogDismiss(). You may or may not have need to remove a dialog from memory – see this substantial discussion in Chapter 8 of this and related topics for more discussion.
DialogDismiss()	Varies	This command is used to dismiss (hide) a dialog but not destroy (remove) it from memory. My experience is that its use varies between WordPerfect versions. Consider the following for what it may be worth, but be sure to

Command/Routine	Wp Ver	Syntax/Usage and Examples
		read the substantial discussion in Chapter 8 before deciding what to do.

If a DialogShow() command does NOT include a Callback function:

WordPerfect 6.1: Although contrary to orthodoxy, and, so, not recommended, I found that DialogDismiss("Help1";1) did no harm.

WordPerfect 7.0 & higher: Don't use the command; but, in WordPerfect 8.0 & higher, you can use a DialogDestroy() command to eliminate the dialog from memory.

All Versions, if a DialogShow() command includes a Callback routine: Use DialogDismiss() either after or within the Callback function itself.

In the past, I used as my model what I'd seen others do without giving much thought to it. Commonly, this method would be seen and used:

```
vLoop=False vQuit=False
DialogShow("vTest";"WordPerfect";cbTest)
Repeat Until(vLoop)
DialogDismiss("vTest";1)
If(vQuit) Quit EndIf
//[Do something else]

Label(cbTest) Switch(cbTest[3])
Caseof "OKBttn";1: vLoop=True
Caseof "CancelBttn";2: vQuit=True vLoop=True
EndSwitch
Return
```

What's the effect of using the 1 in DialogDismiss("vDialog";1)? I really didn't know, or care! It just worked, and I just did it.

The above method had worked for me for years, so why consider actually "learning" something new? Given the information received from J. Dan Broadhead, I could either ignore it or reconsider what I'd been doing and pass it on to you. I chose the latter course, but only after an appropriate amount of kicking and screaming.

Here's the better way (shown more completely in Example 1, Chapter 8):

```
vLoop=False
DialogShow("vTest";"WordPerfect";cbTest)
Repeat Until(vLoop)
If(x=2) Quit EndIf // could be Return instead of Quit
//[Do something else]

Label(cbTest) Switch(cbTest[3])
Caseof "OKBttn";1:  vClose
Caseof "CancelBttn";2: vClose
EndSwitch
Return

Label(vClose)
DialogDismiss("vTest";cbTest[3]) x=MacroDialogResult  vLoop=True
Return
```

So, what's the difference, and what was that 1 in the 1st method which started this comparison?

The 1 in DialogDismiss("vTest";1) forces and assigns the internal variable MacroDialogResult to have a value of 1, i.e., the value assigned if an "OKBttn" closes the dialog. Hence, the vQuit=False statement preceded DialogShow(), and, in the callback label, it was necessary to assign vQuit a value of True should the "CancelBttn" be clicked, so that, after the callback, the macro would know that the "CancelBttn" had been clicked (remember, the DialogDismiss("vTest";1) assigned MacroDialogResult the value of 1, not 2).

But, in the new (immediately above) model, the value assigned to MacroDialogResult IS the value of the control which closes the dialog. That's what happens in Label(vClose), above. I used variable x to assign the value of MacroDialogResult since x is a lot "shorter" to type than

<table>
<tr><th>Command/Routine</th><th>Wp Ver</th><th>Syntax/Usage and Examples</th></tr>
<tr><td></td><td></td><td>MacroDialogResult would be when used following the callback. That was a pragmatic decision, not a necessary one. The statement, If(x=2) Quit EndIf, could just as well have been, If(MacroDialogResult=2) Quit EndIf. Either way, the use of vQuit is no longer necessary and DilaogDismiss() no longer uses an arbitrary, if not fictitious, value to hide the dialog.

Perhaps more importantly, variables associated with controls in the dialog are neither declared nor updated using the above method, which is what you'd probably expect would happen if a "CancelBttn" control is clicked. In the first (old) method, they would be and for no good reason, and, perhaps with a bad consequence since if variables were updated (from variables previously declared) that could affect other code in the macro (unless you intended the macro to actually stop (Quit) a the end of the callback if vQuit=False, which is not necessarily always the case).

See the discussion in DialogDestroy(), above. Also, see DialogShow() [not using CallbackWait] and DialogShow() [using CallbackWait] and this substantial discussion in Chapter 8 of this and related topics.</td></tr>
<tr><td>DialogDisplay()</td><td>All</td><td>DialogDisplay() is an obsolete command. Use DialogShow() [not using CallbackWait] or DialogShow() [using CallbackWait].</td></tr>
<tr><td>DialogShow()

[not using CallbackWait and CallbackResume]

For All Wp Versions</td><td colspan="2">This discussion is not applicable if you want to use the CallbackWait Callback Resume commands present in WordPerfect 7 & higher. For that, see DialogShow() [using CallbackWait], below. The discussion here applies to all WordPerfect versions, 6.1 & higher and should be used if the macro is intended to be compatible in all WordPerfect versions, 6.1 & higher.

This command displays a dialog (in Wp6.1, a Dialog Editor dialog at the least, and, perhaps – I don't know for sure – a DialogDefine dialog; in Wp7.0 & higher, either a Dialog Editor or a DialogDefine dialog) and is of fundamental importance in making macros work the way you want them to. Optionally, the command may include a Callback function which enormously expands flexibility in manipulating and getting information from dialogs. Usage will vary, depending on whether you include a Callback function in the statement or not. Usage variations are:

No Callback: DialogShow("vDialog";"WordPerfect")

Concerning use of DialogShow with or without callbacks, see Chapter 8.

Concerning statements after/before the DialogShow() command, see DialogDismiss and MacroDialogResult and the substantial discussion about such matters in Chapter 8.

With Callback: DialogShow("vDialog";"WordPerfect";cbvDialog), where vDialog is the name of the dialog – the Callback Label name need not match the name of the dialog, even though the called Label matched here for illustration). The last parameter of the statement is the name of the called (callback) label. That label's commands will continue until the label's contents encounter one/more commands which terminate the Callback routine.

In this regard, when a callback routine is used, the DialogShow() command is typically preceded by one or both of these commands: vLoop=False vQuit=False, but see this better approach in DialogDismiss(), above. In the callback label, at the point that vLoop is redefined as vLoop=True, the callback loop is flagged to stop. If vQuit has been redefined as vQuit=False, that information will be shortly be used to stop the macro, or, at least, to proceed to some other routine in the macro, as is shown in the example below or in the better example in DialogDismiss(), above.

After the DialogShow() statement, a Repeat Until(vLoop) statement is commonly used which instructs the macro how long the loop will last. Notice the Repeat Until(vLoop) statement below. The Repeat statement can be preceded by other statements relating to the looping dialog, e.g.,

If(Exists(var9)) RegionShowWindow("vDialog.B4";Show!) Else
RegionShowWindow("vDialog.B4";Hide!) EndIf

In this regard, see Commands At The Beginning of Dialog Show in Chapter 8.

When the Until (vLoop) condition is met, macro execution continues at that point. One or more statements dismissing the dialog then follow, if not already contained in the callback label, as show in DialogDismiss(), above. Also, see DialogDestroy(), above. Then, additional statements instructing the macro's action are made. As an example:

vLoop=False vQuit=False
DialogShow("vDialog";"WordPerfect";cbvDialog)</td></tr>
</table>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		Repeat Until(vLoop) DialogDismiss("vDialog";1) [in WordPerfect 8 & higher, followed by] DialogDestroy("vDialog")] If(vQuit) Quit [or other statement] EndIf [other statements as to what to do next] Label(cbvDialog) Switch(cbvDialog[3]) Caseof "OKBttn": Get1 If(exists(q)) Discard(q) Else vLoop=True EndIf Caseof "CancelBttn": vQuit=True vLoop=True EndSwitch Return Label(Get1) xx=RegionGetWindowText("vDialog.B1") If(xx="") q=1 Messagebox(;"You Didn't Type Something";"Please type something into the edit box.") EndIf Return But, again, see the "better" way described in DialogDismiss(), above. See Chapter 8 for a more thorough discussion, and, for other examples, see Chapter 9, MacroCompile, and MacroDialogResult.

DialogShow()
[using CallbackWait and CallbackResume]
For WordPerfect 7.0 & higher
		Since I write macros intended to be compatible with WordPerfect 6.1 & higher, I don't use the CallbackWait and CallbackResume commands as a rule. So, I obviously know less about the use of such routines than has been described in DialogShow() [not using ...] and I'll just say a little about these commands. If you don't care about WordPerfect 6.1 compatibility, you'll want to explore using callbacks using CallbackWait and CallbackResume. Note that this discussion only applies to using callback loops as part of DialogShow(). Otherwise, see DialogShow [not using ...], above.
		
		WordPerfect's Macro help says, "CallbackWait and CallbackResume are easier to use and more efficient than loops." The principal structure is this:
		
		DialogShow("vDialog";"WordPerfect";cbLabel)
		CallbackWait
		[... other statements which execute after the loop stops...]
		Label(cbLabel)
		[desired statements and including at least one CallbackResume command and, at the end of the Label, a Return command]
		
		Below is an example you can play with. Note For WordPerfect 7.0 Users: You will need to modify the 1st line to replace "US" for "EN" in the Application statement. And, WordPerfect 7.0 is notoriously bad about displaying message boxes – if you don't see the message boxes indicated, press Alt+Tab to see if you can restore focus to it.
		
		Application(A1; "WordPerfect"; Default; "EN")
		Label(Begin)
		Call(vDialog)
		DialogShow("vTest";"WordPerfect";cb)
		CallbackWait
		Messagebox(;"The Button You Clicked and The Value of the Edit Box";x+" -- "+xx)
		Switch(x)
		    Caseof "OK": Discard(xx) Go(Begin)
		    Caseof "Cancel": Quit
		    Caseof "Back": Discard(xx) Go(Begin) EndSwitch
		//Normally, Return, Quit or another command would be here; not needed in this example
		Label(cb) Switch(cb[3])
		Caseof "OKBttn": x="OK" Get1
		    If(exists(q)) Discard(q) Else DialogDestroy("vTest") CallbackResume EndIf
		Caseof "CancelBttn": x="Cancel" xx="No text gotten" DialogDestroy("vTest")
		CallbackResume
		Caseof "BackBttn": x="Back" Get1
		    If(Exists(q)) Discard(q) Else DialogDestroy("vTest") CallbackResume EndIf
		EndSwitch
		Return
		
		Label(Get1)
		xx=RegionGetWindowText("vTest.B1")
		If(xx="") q=1 Messagebox(;"You Didn't Type Something";"Please type something into the edit box.") EndIf Return
		
		Label(vDialog)
		DialogDefine ("vTest"; 50; 65; 200; 64; Percent!+NoCloseBox!; Caption:"Testing

Command/Routine	Wp Ver	Syntax/Usage and Examples
	CallbackWait and CallBackResume") DialogAddEditBox ("vTest"; "B1"; 78; 11; 83; 14; Left!+AutoHScroll!; MacroVar:var0; LimitText:100) DialogAddPushButton ("vTest"; "OKBttn"; 133; 40; 50; 14; OKBttn!; "OK") DialogAddPushButton ("vTest"; "CancelBttn"; 75; 40; 50; 14; CancelBttn!; "Cancel") DialogAddPushButton ("vTest"; "BackBttn"; 17; 40; 50; 14; 0; "Back") DialogAddText ("vTest"; "Static6"; 8; 14; 65; 10; Right!; Text:"Type something:") Return For more discussion about DialogShow and Callback Routines, see Chapter 8.	

DirectoryCreate(string)
DirectoryExists(var;string)
	All	Creates a new directory. Include the full pathname in quotation marks, e.g., DirectoryCreate("C:\Form Files")
		Returns a True/False value if a specified directory exists. For example: DirectoryExists(var;"C:\Form Files") will assign var a value of True if the directory exists, False if it does not.

Discard(var)	All	Destroys a variable from memory; can be used with While/EndWhile statement to destroy different levels of a variable (local, global, persist), e.g., While(Exists(var)) Discard(var) EndWhile. Also, see Repeat, below.
Display()	All	Parameters are On! or Off! During macro playback, this command turns on or off the "display" as the macro executes. In the absence of a Display command, display is off. Macros play faster and video is more stable if display is "off". Usage: Display(On!) or Display(Off!)
DisplayMode(enum)	All	Display the current document in draft or page mode. Enumerations for all WordPerfect versions are Text!, Graphics!, FullPage!. In WordPerfect 8.0 & higher, WebPage! is also possible. Usage: DisplayMode(Text!), etc. See ?DisplayMode, above
DisplayZoom(numeric)	All	Specifies the percentage of the current zoom setting. Enter a number from 25 to 400, e.g., DisplayZoom(75). See Zoom... below, and ?DisplayMode, ?Zoom, above.
DocInitialFont()	All	See Font, below.
DoubleSmartQuote()	All	Turns on/off Double SmartQuotes with the On! or Off! parameter. With an optional numeric parameter, you can specify the numeric equivalent of the character you want to use, though little apparent reason exists to do so. See SingleSmartQuote() and Quick... Commands, below. Usage: DoubleSmartQuote(On!) or DoubleSmartQuote(Off!)
DropCap Commands	All	Numerous DropCap commands exist, none of which are described in this manual. See on-line Macro Help or other references.
EditCopy EditCut EditPaste EditPasteSimple	All	Copies a selection to the clipboard. Copies a selection to the clipboard and deletes it at its current location. Pastes the clipboard contents at the insertion point. Parameters are possible but are not described here. See on-line Windows Help or other references. Same as above, except that text attributes (e.g., bold, paragraph formatting, etc.) are not pasted.
End...	All	Some macro commands use paired starting and ending commands, the ending command being, for example, EndIf. See If() EndIf, Function EndFunc, Procedure EndProc, Switch EndSwitch, Procedure EndProc, While EndWhile, below.
EndNote Commands	All	Numerous EndNote commands exist, none of which are described in this manual. See on-line Macro Help or other references.

Command/Routine	Wp Ver	Syntax/Usage and Examples
Else	All	The Else command is used with the IF conditional statement, e.g., IF ... Else ... ENDIF. If the first statement is True, then commands (if any) following the IF() statement will execute, but if the first statement is False, then commands (if any) following the ELSE statement will execute. See If-Else-EndIf.
Exists(var; enum)	All	Used to determine whether a variable exists. A boolean or numeric value is returned, depending upon whether 1 of 3 items in the possible "pool" of enumerations is used or not. Common usage: x=Exists(var) Other usages: x=Exists(var;Local!) or x=Exists(var;Global!) or x=Exists(var;Persistent!) Examples: x=Exists(var) Result: If variable "var" exists, depending on whether it is a local, global, or persistent variable, x=1 (local), 2 (global) or 3 (persistent), but if the variable does not exist, x=0 (same as NotFound!) (numeric). x=Exists(var;Local!) or x=Exists(var;Global!) or x=Exists(var;Persistent!) Result: If the type of variable enumerated exists, x=True, but, if not, x=False (boolean) Most often, an Exists() statement is combined with an If - EndIf statement or an If - Else - EndIf statement, e.g., If(Exists(var)) [do this] Else [do that] EndIf Also, see Repeat/Until and While/Endwhile, below.
FileExists(var;string)	All	Assigns a boolean True/False value to var as to whether a specific file exists or not. Use the full path of the file. For example: FileExists(var;"C:\Form Files\Petition.wpd") will assign var a value of True if it exits, False if it does not. See FileInsert, FileOpen, below.
FileInsert(string; enum; enum)	All	Inserts a file into the active WordPerfect document at the insertion point. Parameters are: string pathname of file; optionally, whether the file's format should be automatically detected, No! or Yes!; optionally, whether to be prompted to all the file's insertion, Insert! or Prompt!. In WordPerfect 9 and 10, I've found that if the latter 2 parameters are not specified, the implied defaults are Yes! and Insert!. In WordPerfect 8, the implied defaults seem to be Yes! and Prompt!. I'm unsure of defaults in Wp 6.1 or 7.0. The failsafe method is to use the parameters you actually intend. If the file is in a default WordPerfect directory, you can use a ?Path... system command to identify the path, e.g., ?PathCurrent (the current file path), ?PathDocument (default document directory, ?PathMacros, ?PathSpreadsheet. You can also predetermine a variable to represent the path, e.g., vPath="C:\Grande5\". If the specified file does not exist in the specified directory, an error will occur. Use FileExists() to determine that fact before the FileInsert() command is used, or use an error trapping routine - OnError() before using the command. Usage: FileInsert(?PathDocument+"vfile.wpd") or FileInsert("C:\Grande5\vfile.wpd") or FileInsert(vPath+"vfile.wpd") or FileInsert("C:\Grande5\vfile.wpd"; Yes!; Insert!) or FileInsert("C:\Grande5\vfile.wpd"; Yes!; Prompt!). Also, see FileOpen(), below, and ?NumberOpenDocuments, above.
FileNew	All	Creates a new document based on the default template. If 9 documents are already open, an error occurs. See OnError, below, and ?NumberOpenDocuments, above.

Command/Routine	Wp Ver	Syntax/Usage and Examples
FileOpen(string; enum)	All	Opens a specified file into a new WordPerfect document. An optional parameter enumerating the file format type can be used (e.g., Html!), but if you're intending to open a WordPerfect "wpd" document or probably any other file format capable of being opened in WordPerfect, no need exists to use the parameter. See on-line macros help for more information about that. The full pathname should be used. If the file is in a default WordPerfect directory, you can use a ?Path... system command to identify the path, e.g., ?PathCurrent (the current file path), ?PathDocument (default document directory, ?PathMacros, ?PathSpreadsheet. You can also predetermine a variable to represent the path, e.g., vPath="C:\Grande5\". If the specified file does not exist in the specified directory, or if 9 WordPerect documents are already open, an error will occur. Use FileExists() to determine that fact before the FileInsert() command is used, or use an error trapping routine - OnError() before using the command. Usage: FileOpen(?PathDocument+"vfile.wpd") or FileOpen("C:\Grande5\vfile.wpd") or FileOpen(vPath+"vfile.wpd") Also, see ?NumberOpenDocuments and FileInsert(), above.
Find and Find/Replace	All	See Search commands, below
FloatingCell Commands		Several FloatingCell commands exist, none of which are described in this manual. See on-line Macro Help or other references.
FlushRight	All	Aligns text at the right margin. Two consecutive FlushRight commands produces a dot leader from the insertion point to the right margin.
Font()	All	This sets font options at the insertion point. 7 parameters exist: (Name: string; Family: enumeration; Attributes: enumeration; Weight: enumeration; Width: enumeration; Source: enumeration; Type: enumeration; CharacterSet: enumeration), none of which will be elaborated on here – see on-line Macro Help or other references. Also, see Appearance/Attribute commands, above, ?Font, ?FontSize, above, and FontSize(), below. The simplest way to set font options, including the document initial font, is to use the Macro Toolbar's Record button, described in Chapter 5.
FontSize()	All	Sets font size at the insertion point. Use inch measurements. For example: FontSize (0.194") is 14p FontSize (0.167") is 12p FontSize (0.138") is 10p The simplest way to set font size is to use the Macro Toolbar's Record button, described in Chapter 5.
Font - Others	All	Numerous additional font commands are available but are not covered in this manual. See on-line Macro Help or other references.
Footnote Commands	All	Numerous footnote commands are available but are not covered in this manual. See on-line Macro Help or other references.
Function EndFunc	All	Use of "Functions" remains one of the topics that Labels me a "Common Person" instead of a "Macro Guru"! I've not mastered the Function command, even though I can and have "copycatted" from others' macros to use the routines shown in Chapters 6, 7 and 9 in my own macros. So, while I'm not able to explain "why" they work, I can assure you that they do! But, generally, a Function is a macro subroutine containing statements that execute when the Function is called. The Function statements shown in Chapters 6, 7 and 9 test a variable to insure that it matches the content type necessary for the variable to be able to be used to "do" what is needed elsewhere in the macro. For example, if an edit box in a dialog asks the user to enter a "number" which will be used later for math addition, subtraction, division or multiplication, but the user enters an "l" (L) instead of a 1 or an O (Oh) instead of a 0 (zero), subsequently

Command/Routine	Wp Ver	Syntax/Usage and Examples
		attempted math computations using the variable will produce error and the macro will fail. So, during the Callback routine, you will want to test the user's input to be sure it only contains numerals, and, if not, prompt the user to enter a "correct" value. The "Functions" referenced below do this sort of thing. Otherwise, this command is not further developed in this manual. See examples in Chapter 6, Chapter 7, and Chapter 9.
GetNumber()	All	Displays a dialog that lets you input a number in an edit box. The syntax is: GetNumber(var;prompt;title) where "var" is the variable to assign the result to, "prompt" is the string telling the user what to do, and "title" is the string name of the dialog. Probably as of WordPerfect 8.0, the command displays the prior content of the variable, if any, in the dialog. Earlier, the dialog always came up with a blank value in it. See, by analogy, the GetString example in Chapter 4.
GetString()	All	Displays a dialog that lets you input a string in an edit box. The syntax is: GetString(var;prompt;title) where "var" is the variable to assign the result to, "prompt" is the string telling the user what to do, and "title" is the string name of the dialog. Probably as of WordPerfect 8.0, the command displays the prior content of the variable, if any, in the dialog. Earlier, the dialog always came up with a blank value in it. See an example Chapter 4 for an example.
Go()	All	This command instructs macro operation to "go to" a particular Label in the macro. See Chapter 4.
GraphicsLineCreate	All	Creates a graphics line at the insertion point, using one of two possible parameters: GraphicsLineCreate(HorizontalLine!) or GraphicsLineCreate(VerticalLine!) Alternatively, use HLineCreate or VLineCreate.
Graphics Commands	All	Numerous other graphics commands are available but are not covered in this manual. See on-line Macro Help or other references.
HardPageBreak	All	Inserts a hard page break at the insertion point.
HardReturn	All	Inserts a hard return [HRt] at the insertion point. See NTOC, below.
HardSpace	All	Inserts a hard space [HSpace] at the insertion point, having the effect of holding adjacent words/characters together on the same line.
Header Commands	All	Numerous header commands are available but are not covered in this manual. See on-line Macro Help or other references.
Help Commands	All	Numerous "help" commands are available but are not covered in this manual. See on-line Macro Help or other references. Also, see Push Button in Chapter 5.
HLineCreate	All	Inserts a horizontal line from the left margin to the right margin. Also, see VLineCreate, below.
Hypertext Commands	All	Numerous hypertext commands are available but are not covered in this manual. See on-line Macro Help or other references.
Hyphen	All	Inserts a hyphen at the insertion point. Note: For some purposes, e.g., Search/Replace, understand that the hyphen "code" is not the same as the hyphen "character". The latter is, literally, the "minus sign". While the two appear to be the same, they are not. Type a hyphen in your document and turn on Reveal Codes. You will see this code: -Hyphen. In practice, the hyphen "code" will almost always be used – in fact, it's actually awkward

Command/Routine	Wp Ver	Syntax/Usage and Examples
		to enter the hyphen "character" – using WordPerfect's DOS keyboard, Home, - enters the character; using the CUA keyboard, Format, Line, Other Codes..., Hyphen Character, does so. The distinction between the Hyphen "code" and "character" may never present a problem ... but, if a macro routine uses a hyphen code or character and something's not working, explore the distinction in looking for a possible solution.
If - Else - EndIf	All	This is a must-know macro command. Here are some basic rules: (1) an IF statement must ALWAYS end with an ENDIF statement; (2) you can include an ELSE statement if you want; and (3) You can include multiple IF statements within the overall IF statement if you want, but, remember that EACH IF statement must end with an ENDIF statement. The better illustrations of IF - ELSE - ENDIF statements are included in Macro Examples herein, but, here are some simple illustrations: If(var=1) Go(LabelX) EndIf If(var=1) Go(LabelX) Else Go(LabelY) EndIf Also, see Else and Indentation practices in Chapter 5.
IfPlatform() - ElseIfPlatform - EndIfPlatform	All	A conditional statement that specifies a platform (Windows or WordPerfect version) for which subsequent statements are compiled. If the current version of WordPerfect matches the specified condition (WordPerfect version), statements between IfPlatform and EndIfPlatform are compiled and executed, but otherwise not. This provides a means of commands NOT being compiled or BEING compiled for different versions of WordPerfect, when the macro is run in different versions of WordPerfect. If the condition is not met, statements between IfPlatform and EndIfPlatform are not attempted to be compiled. This can be handy if a particular macro command is not included in a particular WordPerfect version in which the macro is being run – otherwise, during macro compilation, an error message would occur and the macro would not compile. The parameter may be a Windows version (e.g., WIN!, Win32!, Win95!, Win98!, WinNT!, Win2000!, WinXP!) or a WordPerfect version (_Version6!; _Version7!; _Version8!, _Version9!; or Version10!). Some subsets for WordPerfect versions (e.g., Professional version) are also possible. I'm not well-versed with this command, and, so, I'll avoid saying more about it here, except to say that a wholly reliable illustration appears in Chapter 8's Version Control discussion in an example furnished by J. Dan Broadhead, and that another example is in Chapter 9's wp9select.wcm. Other than the practical example given in Chapter 8's Version Control, these commands are not developed in this manual. For more, see http://www.jdan.com/perfectscript/macros/ch11_gl.htm#_VPID_11_273 and http://www.jdan.com/perfectscript/macros/appa_2.htm#_VPID_AA_8057.
Import Commands	All	Numerous import commands are available but are not covered in this manual. See on-line Macro Help or other references.
Indent IndentLeftRight	All	Indents a paragraph from the left margin. Indents both sides of a paragraph equally from left and right margins.
Integer()	All	This math command only works with "numeric" values. The value returned is the "whole number" of the identified variable or value. So, x=Integer(1.09) results in x having a value of 1, the whole number. Similarly, x=Integer(x), where the value of x before the command is 1.09, results in x having a value of 1. The numeric range of the Integer command is -2,147,483,648 to 2,147,483,647, inclusive. If the integer is beyond that range, x=Integer(num) will produce zero (0) as the result. See Chapter 6, Working With Integers.

Command/Routine	Wp Ver	Syntax/Usage and Examples
Justification() JustifyAll, JustifyCenter, JustifyFull, JustifyLeft, JustifyRight	All	Sets justification with one of 6 parameters: Center!, DecAlign!, Full!, FullAll!, Left! or Right!. Example: Justification(Full!) Alternatives to the Justification() command. See ?Justification, above.
Label()	All	A "Label" is a specific location within a macro, the place-name being within the (). The name must begin with a letter but can contain letters, numbers, or "_" after the initial letter. Though you can use upper/lower case for ease of reading, Label names are not case sensitive. Don't use Reserved Words for a Label's name. Use up to 30 characters. Following the Label(vLabel) statement, various commands are usually stated to be executed at that point in the macro. If the Label is a "called" Label, be sure to end the Label's statements with a Return command. If it is intended that the macro stop all operations when the Label's statements are executed, use the Quit command. See Chapter 4.
Labels Commands	All	Several Labels commands are available to create "Labels", e.g., mailing Labels, but are not covered in this manual. See on-line Macro Help or other references.
LineSpacing()	All	Sets line spacing for the current and following paragraphs. Example: LineSpacing(1.5) Also, see ?LineSpacing, above.
Line Commands	All	Numerous other Line commands are available but are not covered in this manual. See on-line Macro Help or other references.
List Commands	All	Numerous List commands are available but are not covered in this manual. See on-line Macro Help or other references.

MacroCompile()
For Wp8 & higher
	Compiles a macro. Macros compiled in one "major version" (6, 7, 8, 9, 10, 11, 12 – see ?MajorVersion) or, sometimes, a specific "minor version", need to be recompiled when run in a different WordPerfect version. While this is done automatically the 1st time that a macro is run in the different WordPerfect version, very long and code-intensive macros may take awhile for the recompilation to occur, depending on the computer. You may want to use this command in conjunction with MacroIsCompiled, below. This command presents a dialog so that a user knows what is happening and, therefore, the user doesn't think the computer and/or WordPerfect has locked up. If the referenced macro does not exist, a NotFound error condition occurs, so you may want to include an OnNotFound command, as well.
	Syntax: MacroCompile(string value macro of the macro file; [optional parameters: Wait! or Don'tWait!; Debug!; NoPrompts!; ShowProgress!; GenerateListing!; ForceRecompile!; ShowIcon!). These parameters are not developed here. See on-line macro help.
	I'm not fond of the prompt command since it allows a user to cancel the compilation. The routine I prefer to use is set out below. It assumes that a macro is assigned to variable "vMac" before Label(chkMac) is called:
	Label(chkMac)
	vMac="c:\mymacros\test1.wcm")
	x=MacroIsCompiled(vMac)
	If(x=true) Return EndIf
	Call(vDialog)
	RegionSetWindowText("vCompile";vmac+" Must Be Recompiled")
	DialogShow("vCompile";"WordPerfect") DialogDestroy("vCompile")
	Msg1="Recompilation of "+vmac+" is in progress" Msg2="Please wait until it's done"
	WaitMsg
	MacroCompile(vMac;NoPrompts!|ShowProgress!)
	x=MacroIsCompiled(vMac)
	Switch(x)
	    Caseof True: vMsg="Compilation is done. The macro will now run"
	    Caseof False: vMsg="Compilation was interrupted. The macro will not run"
	    EndSwitch
	Messagebox(;"Compilation Report";vMsg)
	If(x=False) Quit Else Return EndIf	

<table>
<tr><th>Command/Routine</th><th>Wp Ver</th><th>Syntax/Usage and Examples</th></tr>
<tr><td></td><td colspan="2">Label(vDialog)
DialogDefine (Dialog:"vCompile"; Left:50; Top:60; Width:222; Height:107; Style:Percent!+NoCloseBox!; Caption:"The Macro Must Be Recompiled")
DialogSetProperties (Dialog:"vCompile"; FontName:"Arial"; FontSize:9p)
DialogAddPushButton (Dialog:"vCompile"; Control:"OKBttn"; Left:67; Top:89; Width:85; Height:12; Style:OKBttn!; ButtonText:"Begin Recompilation")
DialogAddText (Dialog:"vCompile"; Control:"S1"; Left:6; Top:6; Width:209; Height:20; Style:Left!; Text:"The specific WP release version of the compilation is different than what the macro is presently precompiled for.")
DialogAddText (Dialog:"vCompile"; Control:"S2"; Left:7; Top:30; Width:206; Height:19; Style:ShadowBox!+Left!; Text:"Recompilation only happens once and should take just a few seconds.")
DialogAddText (Dialog:"vCompile"; Control:"S3"; Left:7; Top:53; Width:206; Height:30; Style:ShadowBox!+Left!; Text:"Recompilation starts when you click the Begin button. When recompilation is done, you will be advised and the program will run.") Return
Label(WaitMsg)
DialogDefine (Dialog:"vWait"; Left:50; Top:80; Width:136; Height:59; Style:Sizeable!+Percent!+NoCloseBox!; Caption:"Let's Wait")
DialogSetProperties (Dialog:"vWait"; FontName:"Arial"; FontSize:9p)
DialogAddText (Dialog:"vWait"; Control:"Static1"; Left:6; Top:6; Width:124; Height:25; Style:ShadowBox!+Center!; Text:"")
DialogAddPushButton (Dialog:"vWait"; Control:"CancelBttn"; Left:3; Top:3; Width:0; Height:0; Style:CancelBttn!; ButtonText:"Cancel")
DialogAddText (Dialog:"vWait"; Control:"Static2"; Left:4; Top:38; Width:127; Height:17; Style:ShadowBox!+Center!; Text:"") Return
Label(WaitMsg) DialogShow("vWait";"WordPerfect";cbWait) Return
Label(cbWait) If(cbWait[3]="CancelBttn") Assert(CancelCondition!) EndIf Return
Label(KillMsg) DialogDismiss("vWait";2) DialogDestroy("vWait") Return</td></tr>
<tr><td>MacroDialogResult</td><td>All</td><td>The discussion here is a practical, not technical, discussion of the MacroDialogResult variable. What you need to know is that when any dialog is closed, a special variable named MacroDialogResult is always present and is assigned a value. That value remains present in memory as variable MacroDialogResult until any other Dialog... command is called (e.g., DialogDestroy), at which time MacroDialogResult's prior value is updated.
If it is desired to save the MacroDialogResult value, do so by var=MacroDialogResult.
While typically not needed in dialogs using Callback functions, it may be useful for Dialogs without Callback functions which have multiple means of closing the dialog, i.e., more than one button may close the dialog.
Most importantly, see the substantial discussion about this and related matters in Chapter 8, and the discussion in DialogDismiss(), above.
In WordPerfect 8 & higher, if you use a DialogDestroy command at all, be sure to include the command before a DialogDestroy() command and then tell the macro what to do based upon the value of MacroDialogResult, e.g.:
If(MacroDialogResult=1) [do this] Else [do that] EndIf
If used without a callback routine, possible values assigned to MacroDialogResult will be:
If an OKBttn closes the dialog, MacroDialogResult = 1;
If a CancelBttn closes the dialog, MacroDialogResult =2;
If a user-defined Button closes the dilaog, MacroDialogResult = the RegionName (a case sensitive string, in quotes (e.g., "TipsBttn", "BackBttn"), exactly as you have defined the button's name in the dialog.
Usage: var=MacroDialogResult, e.g., x=MacroDialogResult.</td></tr>
</table>

Command/Routine	Wp Ver	Syntax/Usage and Examples
MacroFilePlay()	All	Plays a macro, but WordPerfect's macro help doesn't say much about it. The string parameter is the filename of the macro to play. It's better to use full pathname if the macro is not in the default or supplemental macro directory. Example: MacroFilePlay("C:\Grande5\Grande.wcm"). See Run(), below. J. Dan Broadhead advises: Be careful with this one. This is a WordPerfect command, and it will suspend the current macro while the new macro runs. From the macro system's perspective, a complete new macro is started. The first macro is essentially suspended, because WordPerfect never returns to the macro system from the MacroFilePlay command until after WordPerfect has asked the macro system to load and play another separate macro file. In this case, the macro system loads an entire new macro context, which shares nothing with the prior macro. In addition, WordPerfect itself may reset some of its own internal context/state information and some information may not be reset. And there are some restrictions as to when you can call this command and when you cannot. Calling the Run or Nest command is much safer and predictable. These commands are like calling the Call command for a Label, but operate on a whole macro. These commands are implemented in the macro system, and not in WordPerfect, and the macro system uses the same existing context/state for the new macro. And since WordPerfect does not know this is happening, it also retains and shares the same existing context and state for the new macro as well. The current macro is suspended until the named macro file completes. However, if the named macro does a Quit command, then the prior/suspended macro is also ended rather than resumed.

MacroInfo()	All	Returns information about the computer, operating system, macro state and other possible information. This command is not developed in this manual beyond what's stated here - see on-line macro help for the numerous possibilities. The type of value returned (boolean, numeric, string) depends on the information requested. A few examples are shown below:
		x=MacroInfo(ComputerName!) [string]
		x=MacroInfo(UserName!) [string]
		x=MacroInfo(PlatformVersion!) [string name, abbreviated]
		x=MacroInfo(PlatformVersionString!) [full string name]
		Also, see VersionInfo.

MacroIsCompiled()	Wp8 & higher	Determines whether a named macro is compiled or not. If so, a boolean value of *True* is returned; if not, *False*. Syntax: x=MacroIsCompiled("C:\MyMacs\test.wcm") or if the macro file name has been assigned to a variable, x=MacroIsCompiled(vMac). See MacroCompile, above, for an example.
MacroPause	All	Pauses playback of a macro until you press Enter or select Pause on the Macro Menu. During the pause, you can type stuff in the document, until you press the Enter key, which resumes macro playback. See PauseKey, below.
MarginBottom(), MarginLeft(), MarginRight() and MarginTop()	All	Sets the respective margins in inches. Example: MarginTop(0.75")
MatchExtendSelection	All	Used with Find and Replace (Search); see Search Commands, below.
MatchPositionAfter	All	Used with Find and Replace (Search); see Search Commands, below.
MatchPositionBefore	All	Used with Find and Replace (Search); see Search Commands, below.
MatchSelection	All	Used with Find and Replace (Search); see Search Commands, below.

Command/Routine	Wp Ver	Syntax/Usage and Examples
Merge Commands	All	Numerous merge commands are available but are not covered in this manual. See on-line Macro Help or other references.
MessageBox()	All	MessageBox() is used to display a message. While you have less control over the content and appearance, a Message Box is a simpler to make than a regular dialog. But, my experience is that, sometimes, the Message Box gets "lost" (hidden, not visible), particularly if your macro is moving between different WordPerfect documents as it does its stuff. When that happens, the message box is really "there" but the user can't see it ... so the macro is waiting for a button to be pushed in the message box which the user can't see and so nothing happens, except that the user gets frustrated and closes the macro, or even WordPerfect, while that message box is waiting for a button to be pushed. This is particularly a problem in WordPerfect 7 and 8. Dialogs don't seem to have that problem, and, as they are so easy to make in the Dialog Editor, and since they "look" better, I've pretty well stopped using the MessageBox() command. Additionally, consider J. Dan Broadhead's comments: The problem in the MessageBox gettting lost is a well know problem that affects version 7 and 8. Version 6 did not suffer as much from this problem (but it was a 16 bit Windows 3.x product, not a Windows 95/98/ME/NT/2000/XP product, and Windows 3.x behaves differently). The ultimate problem is for a dialog box, you can specify the parent window for the dialog box. Prior to version 9, you could not specify the parent for any of the other prompts, message boxes, etc., that can be used in a macro. But starting in version 9, all those commands (including dialog boxes) now use a specific macro system internal method to determine the parent window. This parent window is initially set to the process that started the macro (usually WordPerfect), but this parent window can be explicitly specified by using the SetDefaultParent command in WP9 & higher. Dialog boxes are still the only commands where a specific parent can be specified when the DialogShow command is called, but all the other commands use the contents of the values speficied by the SetDefaultParent command. Despite its limitations, the use of Message Boxes in a macro's code is a useful tool to troubleshoot a macro – such as by inserting a Message Box around the suspected point of a macro's failure to do what you want it to. Here's how to use this command: MessageBox(var;"Title of Message";"Text of message.";enum). Use of "var" and "enum" are optional and, if no user interaction is needed, they are superfluous. In its simplest form, the command can be: MessageBox(;"This Is The End of the Macro";"Happy Trails, Bud!"). In this form, an OK button is inserted in the box automatically. But, if user interaction is needed with the message, such as a Yes, No, Cancel response to the message, include the variable and the enumeration you want to close the box. Some, but not all, possibilities are: MessageBox(x;"Are You Sure You Want To Stop The Macro?";"Your prior choice will stop the macro. Are you sure that's what you want to do?";YesNo!). A Yes and No button will appear in the box and the value of the button pushed will be assigned to the variable "x". Self-explanatory closing enumerations are: OkCancel! YesNo! YesNoCancel! AbortRetryIgnore! RetryCancel! Values returned to the variable for the possible buttons are: OK=1 Cancel=2 Abort=3 Retry=4 Ignore=5 Yes=6 No=7 Or, instead of testing for the above numeric values, you can do as J. Dan Broadhead recommends: I would encourage the use of the enumeration constants that this command returns, that can ease in the understanding of the macro code.

Command/Routine	Wp Ver	Syntax/Usage and Examples
		As with all enumerations returned from commands, the enumerations must be prefixed by the name of the command they are returned from. So instead of checking for the value 6 to mean Yes, check for MessageBox.YesButton!: x = MessageBox (.....) If (x = MessageBox.YesButton!) [do this] EndIf This will work back to at least version 7, and possibly version 6 as well. You can also combine the above enumerations with icon enumerations, if you want. Separate multiple enumerations by a "pipe" character: IconAsterisk! | IconExclamation! | IconHand! | IconQuestion! So, with an icon enumeration, the command might be: MessageBox(x;"An Error Has Occurred In The Macro!";"Something dreadful has happened and your hard disk has just dissolved. SO sorry! Do you want me to fix it?";YesNo! | IconQuestion!) Other variations are possible. See MessageBox in Macro help. And, see NTOC(), below, for adding hard returns in your message lines, and SetDefaultParent, generally.
Nest()	All	Use Run(), instead. Used to run a macro, included for DOS compatibility.
NTOC()	All	While other Number To Character (NTOC) commands exist, without doubt the most common is NTOC(0F90Ah), a "hard return". So, if you define a variable, e.g., "H", as H=NTOC(0F90Ah), you can use the variable H to insert a hard return, such as in a variable, e.g.: MsgTxt="This is line one of the message."+H+H+"This is line two of the message."+H+"This is line three of the message." The variable MsgTxt will include two hard returns after the first line and one after the second line. Also, see HardReturn, above.
NumStr()	All	Converts a numeric value to a string value. If the value is not numeric, error will result. The 1st parameter is the numeric value or variable containing one. An optional 2nd parameter is the number of decimals to return if the numeric value is not a whole number. If the 2nd parameter is omitted, WordPerfect 10's on-line help says that "the default is 6", but I've not found that to be so in WordPerfect 10 - all decimal values are returned up the maximum if the 2nd parameter is omitted. The maximum value of the 2nd parameter in WordPerfect 10 is 16. I don't know if that's different in other WordPerfect versions. Values after the 2nd parameter are rounded up or down, 5 being the round-up point. So var=NumStr(12345.5;0) results in var equaling 123456, and var=NumStr(12345.55;1) SHOULD result in var equaling 12345.6, even though in my WinXp machine, in Wp9 and Wp10, WinXP that the result is 12345.5. The problem isn't with WordPerfect but is a Windows 2000 & higher issue. See this discussion at WordPerfect Universe and here in Chapter 6. Consequently, great care should be taken when using NumStr() in Windows 2000 or later, and it might be better to parse pre-and-post decimal parts, work on them, and then join them back together. But, if you are a Win2000 or later user, it might be good if you'd take a close look at any NumStr() commands in your WordPerfect macros. Usage: var=NumStr(200.05), or var=NumStr(var), where var is initially a numeric value, or var=NumStr(200.05;0) or var=NumStr(var;16). See StrNum and Chapter 6, and, particularly, working with decimals.
OLE Commands	All	Numerous OLE commands are available but none of them are covered in this manual. See on-line Macro Help or other references.

Command/Routine	Wp Ver	Syntax/Usage and Examples
OnCancel(Label)	All	A macro will stop running when a cancel condition occurs unless it is preceded by an OnCancel statement. OnCancel(Label) tells the macro where to go if a cancel condition occurs and macro statements in that Label will be executed. You can have different OnCancel statements at different places in a macro, but if you don't reset the OnCancel statement at the end of a subroutine the secondary OnCancel statement remains effective. Also, see Assert(), above. Generally, see Chapter 4.
OnCancel Call(Label)	All	Same as OnCancel(Label) except that the referenced Label (and its contents) are "called", just as in a regular Call statement. "Return" at the end of the Called Label returns macro execution to the 1st statement after the cancel condition. As a practical matter, most often you will probably want to use OnCancel(Label) instead. Also, see Assert(), above. Generally, see Chapter 4.
OnError(Label)	All	If an error (or error condition) occurs during a macro's operation, a WordPerfect PerfectScript message will appear with some information about the error after which the macro will stop. If you establish an OnError(Label) command before the error condition occurs, macro execution will instead be directed to the Label specified and the contents of that Label will be executed. You can have different OnError statements at different places in a macro, but if you don't reset the OnError statement at the end of a subroutine the secondary OnError statement remains effective. Also, see Assert(), above. Generally, see Chapter 4.
OnError Call(Label)	All	Same as OnError(Label) except that the referenced Label (and its contents) are "called", just as in a regular Call statement. "Return" at the end of the Called Label returns macro execution to the 1st statement after the error condition. As a practical matter, most often you will probably want to use OnError(Label) instead. Also, see Assert(), above. Generally, see Chapter 4.
OnNotFound(Label)	All	If a Not Found condition occurs during a macro's execution, a WordPerfect PerfectScript message will appear with some information in it after which the macro will stop. If you establish an OnNotFound(Label) command before the Not Found condition occurs, macro execution will instead be directed to the Label specified and the contents of that Label will be executed. You can have different OnNotFound statements at different places in a macro, but if you don't reset the OnNotFound statement at the end of a subroutine the secondary OnNotFound statement remains effective. Also, see Assert() and MacroCompile, above, and Search Commands, below. Generally, see Chapter 4.
OnNotFound Call(Label)	All	Same as OnNotFound(Label) except that the referenced Label (and its contents) are "called", just as in a regular Call statement. "Return" at the end of the Called Label returns macro execution to the 1st statement after the Not Found condition. As a practical matter, most often you will probably want to use OnNotFound(Label) instead. Also, see Assert(), above, and Search Commands, below. Generally, see Chapter 4.
Outline Commands	All	Numerous outline commands are available but are not covered in this manual. See on-line Macro Help or other references.
PageNumber(num)	All	Sets a new page number for the current page, useful if you want to restart page numbering on a particular page. The value is numeric, so don't use quotation marks. Example: PageNumber(1) or PageNumber(4).
PageNumber Commands	All	Many PageNumber... commands are available but are not covered in this manual, except PageNumber(), above. See on-line Macro Help or other references for more.
Paragraph Commands	All	Many Paragraph... commands are available but are not covered in this manual, except ParagraphSpacing(), below. See on-line Macro Help or other references for more.

Command/Routine	Wp Ver	Syntax/Usage and Examples
ParagraphSpacing()	Varies	Sets spacing between paragraphs. In Wp9 and 10, 2 possible parameters: numeric lines (e.g., 1.5) and numeric distance in points. If you're going to use the latter, it's easier to use the Macro Toolbar's Record button, Chapter 5. In Wp6.1 through 8, only use 1 parameter, the numeric lines between paragraphs. Example: ParagraphSpacing(1.5)
PauseKey()	All	Pauses a macro and Identifies the key to be used to resume macro playback. 2 parameters: 1 of these: Any!, Cancel!, Character!, Close! or Enter!; and a case-sensitive character (if Character! is the first parameter). Example: PauseKey(Character!;"~") See MacroPause, above. Other Pause commands, e.g., Pause, MacroPause, PauseCommand(), are not described in this manual. See on-line Macro help or other references.
Position Commands	All	While many Position commands are described below, many others are not. See on-line Macro help for other position commands. And, see the various commands beginning with "Pos", below. For differences in Wp10's (& higher) text selection method which can affect various PosWord... macro commands, see BMT and wp9select.wcm.
PosCharacter	All	Moves the insertion point forward to a specified case-sensitive character (2,000 character movement limit), positioning the insertion point after that character. Example: PosCharacter("~")
PosCharNext	All	Moves the insertion point to the next character to the right.
PosCharPrevous	All	Moves the insertion point to the previous character.
Pos Column Commands	All	Several position commands for use in columns are available but are not covered in this manual. See on-line Macro Help or other references.
PosDocBottom	All	Moves the insertion point to the very end of the document.
PosDocTop PosDocVeryTop	All	Moves the insertion point to the top of the document, after any initial codes. Moves the insertion point to the very top of the document, before any initial codes.
PosLineBeg PosLineVeryBeg	All	Moves the insertion point to the beginning of the current line, after codes. Moves the insertion point to the beginning of the current line, before codes.
PosLineDown	All	Moves the insertion point down one line.
PosLineEnd PosLineVeryEnd	All	Moves the insertion point to the end of the current line, before any ending codes. Moves the insertion point to the end of the current line, after any ending codes.
PosLineUp	All	Moves the insertion point up one line.
Position in table commands	All	Several, but not all, are covered below. Also, See Table Commands, below.
PosTableBegin	All	Moves the insertion point to the first cell of a table.

Command/Routine	Wp Ver	Syntax/Usage and Examples
PosTableCell()	All	Moves the insertion point to a specified cell. To move to a specific table, use the table's Name, e.g., "E". If multiple tables are in a document and the insertion point is not already in the table you want, use the full name of the table, e.g., PosTableCell("Table A") to move to the 1st cell in the table that is not a protected table cell. In-Table Example: PosTableCell("A5")
PosTableCellBottom	All	Moves the insertion point to the beginning of the last line in a cell.
PosTableCellDown	All	Moves the insertion point down one table row to the beginning of the first line in the cell. If the insertion point is not in a table, an error message will occur.
PosTableCellNext	All	Moves the insertion point to the beginning of the next cell.
PosTableCellPrevious	All	Moves the insertion point to the beginning of the previous cell.
PosTableCellTop	All	Moves the insertion point to the beginning of the cell.
PosTableCellUp	All	Moves the insertion point up one table cell.
PosTableEnd	All	Moves the insertion point to the beginning of the last table cell.
PosTableRowBegin	All	Moves the insertion point to the beginning of the current row.
PosTableRowEnd	All	Moves the insertion point to the beginning of the last cell in the current row.
PosWordNext	All	Moves insertion point to the beginning of the next word. Hyphenated words are treated as one word. Wp10 & higher: BMT; wp9select.wcm
PosWordPrevious	All	Moves the insertion point to the beginning of the previous word, unless the insertion point is located within a word, in which event the movement is to the beginning of the current word. Wp10 & higher users: see BMT.
Pos Commands	All	Many other "Pos" (position insertion point) commands are also available. See on-line Macro Help or other references. Wp10 & higher users: see BMT; wp9select.wcm
Pref Commands	All?	Numerous "Preferences" commands are available, e.g., default file locations, etc., but none are shown here. See on-line Macro Help or other references.
Print()	All	Prints the current document. If used without any parameters, the document will be printed using the previously set print option. Or, use an "action" parameter: AdvancedMultiplePages!, CurrentPage!, DocumentOnDisk!, DocumentSummary!, FullDocument!, MultiplePages!, or SelectedText!. The on-line Macro Help notes, "The option in Print should be "MultiplePages!" instead of "AdvancedMultiplePages!". The "Advanced" thing is really used to set a range of pages, like: PrintRangeFrom(2) PrintRangeTo(3) Print(AdvancedMultiplePages!). Example: Print – without a parameter, prints according to then existing print settings, which would normally be Full Document. Print(FullDocument!) would print the full document, regardless of then existing print settings. Many other "Print..." commands are also available. See on-line Macro Help or other references.
Procedure EndProc	All	The Procedure command is used to identify a macro subroutine that can receive one or more values from a calling statement. Procedures contain one or more statements that execute when the procedure is called. This command is not further developed in this manual. See Function, above.
Prompt() EndPrompt	All	I don't use Prompt(), Pause, EndPrompt and mention them only so you'll know they're there. See on-Line Macro help or other references.
ProofReadAsYouGoOff	Wp 8 & higher	Turns off Spell-As-You-Go and Grammar-As-You-Go

Command/Routine	Wp Ver	Syntax/Usage and Examples
QuickCorrect and Quick... Commands	Various	Other QuickCorrect and other Quick... [this or that] control commands are available than are described below. See on-line Macros Help or other resources. And, see DoubleSmartQuote(), above, and SingleSmartQuote(), below.
QuickCorrect()	All	Turns QuickCorrect off or on by using the Off! or On! parameter. Examples: QuickCorrect(Off!) or QuickCorrect(On!) See ?QuickCorrect, above, and the various other QuickCorrect... commands below.
QuickCorrectQuickBulletsQry	Wp7; obsolete in Wp9, Sp4 & higher, but works	Returns a boolean True/False value. Although included in Wp10-12's on-line Macro Help, use of this command produces an "obsolete" warning message during macro compilation in WordPerfect 9, Service Pack 4 and WordPerfect 10-12, even though it appears to function properly. The purpose of the command is to determine whether or not QuickCorrerctQuickBullets is turned "on" or "off", upon which information you could then use QuickCorrectQuickBulletsSet(On!) or QuickCorrectQuickBulletsSet(Off!) depending on what you wanted. Example: var=QuickCorrectQuickBulletsQry. Var will be assigned a value of True if QuickCorrectQuickBullets is turned on, or False if it isn't.
QuickCorrectQuickBulletsSet()	Wp7 & higher	Turns QuickCorrectQuickBullets off or on by using the Off! or On! parameter. When writing a document with a macro, unless you want to use automatic paragraph numbering, you will probably want to use this command to turn the WordPerfect feature off. Ditto the QuickCorrectQuickIndent feature, unless you like indented paragraphs. If you want precise control over paragraph numbering and if you want paragraph structure to be like this: [Left Tab]3., followed by another tab or a couple of spaces, with text wrapping to the left margin instead of being indented, use the following to accomplish that result: QuickCorrectQuickBulletsSet(Off!) QuickCorrectQuickIndentSet(Off!) Note that you can use a variable for the actual paragraph number.
QuickCorrectQuickIndentQry	Wp7; obsolete in Wp9, Sp4 & higher, but works	Returns a boolean True/False value. Although included in Wp10-12's on-line Macro Help, use of this command produces an "obsolete" warning message during macro compilation in WordPerfect 9, Service Pack 4 and WordPerfect 10-12, even though it appears to function properly. The purpose of the command is to determine whether or not QuickCorrerctQuickIndent is turned "on" or "off", upon which information you could then use QuickCorrectQuickIndentSet(On!) or QuickCorrectQuickIndentSet(Off!) depending on what you wanted. Example: var=QuickCorrectQuickIndentQry. Var will be assigned a value of True if QuickCorrectQuickIndent is turned on, or False if it isn't.
QuickCorrectQuickIndentSet()	Wp7 & higher	Turns QuickCorrectQuickIndent off or on by using the Off! or On! parameter. See notes under QuickCorrectQuickBulletsSet(), above. Example: QuickCorrectQuickIndentSet(Off!) turns off QuickCorrectQuickIndent.
Quit	All	Stops all macro action. See Chapter 4.
Region Name Syntax	All	This isn't a command, it's just a note describing the correct syntax when a a Region Name is used in a Region... command such as RegionGetWindowText(). A Region Name consists of (1) The dialog name; (2) a period; (3) the Control Name within the named dialog; and, (4), all of which is surrounded by a pair of quotation marks. All parts of a Region Name are case sensitive. See the various Region... commands, below. So, for a dialog named "Test", it will contain various "controls". One such region may be a list box named "B1". You determine the control's name. Wp 6.1 Example: RegionGetSelectedText(var;"Test.B1") Wp7.0 & higher: var=RegionGetSelectedText("Test.B1") See Chapter 8 for more detail.

Command/Routine	Wp Ver	Syntax/Usage and Examples
RegionAddListItem()	All	Adds items to a dialog's list box, combination box or popup button; it will commonly follow a RegionResetList() command. See Region Name Syntax, above. The second parameter can be repeating, separating items by semi-colons. Usage: RegionAddListItem("RegionName";"item"), e.g., RegionAddListItem("Test.B1";"bananas") or RegionAddListItem("Test.B1";"bananas";"apples";"pears"). See Chapter 8 for more detail.
RegionGet ...() commands, generally	All	A group of "RegionGet..." commands are of great value in making/using a macro containing a dialog in which DialogShow() will include a "callback" routine. All "RegionGet..." commands described below assign a variable the value of a specified Dialog Control. See Region Name Syntax, above, and in Chapter 8 for much more detail. The syntax varies in Wp6.1 and in all later Wp versions. In Wp6.1, the variable's name is a parameter of the RegionGet statement. In Wp7 & higher preferred syntax, it is not, and the variable name precedes the statement followed by a equal(=) sign. See Chapter 8 for much more detail. If the older form is used in WordPerfect 9.0 (Service Pack 4) or later, a warning message will occur during compilation advising you that you are using obsolete code – but the macro will still compile if you continue compilation.
RegionGetCheck()	All	In a callback routine, this gets the enumeration value of a dialog checkbox - Checked! or Unchecked!. The variable is assigned that value. Location of the variable's name in the statement differs between WordPerfect versions. See RegionGet ...(), above, and Region Name Syntax, above. Wp6.1 syntax: RegionGetCheck(var;"RegionName"), e.g., RegionGetCheck(var;"Test.B1") Wp7 & higher preferred syntax: var=RegionGetCheck("RegionName"), e.g., var=RegionGetCheck("Test.B1"), but note the following. If the older form is used in WordPerfect 9.0 (Service Pack 4) or later, a warning message will occur during compilation advising you that you are using obsolete code – but the macro will still compile if you continue compilation. See Chapter 8 for more detail.
RegionGetSelectedText()	All	In a callback routine, this gets the text selected in a dialog's list box, combination box, or counter. If no selection is made, the return value is "" (empty); if multiple selections are made, the selections are separated by semi-colons. Location of the variable's name in the statement differs between WordPerfect versions. See RegionGet ...(), above, and Region Name Syntax, above. Wp6.1 syntax: RegionGetSelectedText(var;"RegionName"), e.g., RegionGetSelectedText(var;"Test.B1") Wp7 & higher preferred syntax: var=RegionGetSelectedText("RegionName"), e.g., var=RegionGetSelectedText("Test.B1"), but note the following. If the older form is used in WordPerfect 9.0 (Service Pack 4) or later, a warning message will occur during compilation advising you that you are using obsolete code – but the macro will still compile if you continue compilation. See Chapter 8 for more detail.

Command/Routine	Wp Ver	Syntax/Usage and Examples
RegionGetWindowText()	All	In a callback routine, this gets a dialog's caption bar text, static text, or edit box text – i.e., text items where no choices are present. See RegionGet ...(), above, and Region Name Syntax, above. Wp6.1 syntax: RegionGetWindowText(var;"RegionName"), e.g., RegionGetWindow(var;"Test.B1") Wp7 & higher preferred syntax: var=RegionGetWindowText("RegionName"), e.g., var=RegionGetWindowText("Test.B1"), but note the following. If the older form is used in WordPerfect 9.0 (Service Pack 4) or later, a warning message will occur during compilation advising you that you are using obsolete code – but the macro will still compile if you continue compilation. See Chapter 8 for more detail.
RegionRemoveListItem()	All	In a callback routine, this command can be used to remove item(s) from a dialog's list box which are inappropriate choice(s) based upon the value of some other item in the dialog or some other reason occurring before the dialog opens. See Region Name Syntax, above. Usage: RegionRemoveListItem("RegionName";"item"), e.g., RegionRemoveListItem("Test.B1";"bananas"). See Chapter 8 for more detail.
RegionResetList()	All	In a callback routine, this clears the list in a dialog's list box. It will commonly be followed by a RegionAddListItem() command, above. See Region Name Syntax, above. Usage: RegionResetList("RegionName"), e.g., RegionResetList("Test.B1"). See Chapter 8 for more detail.
RegionSelectListItem()	All	In a callback routine, this selects specific item in a List Box, Combo Box, or Counter Box control in a dialog. See Region Name Syntax, above. An optional ending parameter is possible, Select!, Unselect! or Extend!. Select! selects the item and unselects all others in the list; Unselect! unselects the item; Extend! selects an item and adds it to any others which may already be selected. Usage: RegionSelectListItem("RegionName";"item") e.g., RegionSelectListItem("Test.B1";"No"). See Chapter 8 for more detail.
RegionSetCheck()	All	In a callback routine, this sets the value of a check or radio control box as Unchecked! (empty) or Checked! (checked). See Region Name Syntax, above. Usage: RegionSetCheck("RegionName";numeric) e.g., RegionSetCheck("Test.B1";Unchecked!) or RegionSetCheck("Test.B1";Checked!). See Chapter 8 for more detail.
RegionSetFocus()	All	In a callback routine, this sets the active input position within a dialog. See Region Name Syntax, above. Usage: RegionSetFocus("RegionName"), e.g., RegionSetFocus("Test.B1"). See Chapter 8 for more detail.
RegionSetWindowText()	All	In a callback routine, this sets the text of a control within a dialog. See Region Name Syntax, above. Usage: RegionSetWindowText("RegionName";"item"), e.g., RegionSetWindowText("Test.B1";"2/2/2002"), and, for the dialog's title bar, RegionSetWindowText("Test";"About This Dialog"). See Chapter 8 for more detail.

Command/Routine	Wp Ver	Syntax/Usage and Examples
RegionShowWindow()	All	In a callback routine, this hides or makes visible a particular control within a dialog. Although other enumerations are available, as a practical matter, the enumerations Show! (makes visible) or Hide! (makes hidden) are all you probably need to know. See Region Name Syntax, above. Before reading J. Dan Broadhead's comments on this point, I'd become accustomed to using numeric values for Show! (1) and Hide! (0). However, he noted to me that: People should be encouraged to always use the enumeration names. Using the numeric values can lead to macro code that is difficult to read and make sense of later on when read by themselves or others. There have also been cases when the internal numeric values for the enumerations have had to change from one version to the next, and using the numeric values would cause the code to not work properly after such an internal macro system change. Usage: RegionShowWindow("RegionName";enum), e.g., RegionShowWindow("Test.B1";Show!) or RegionShowWindow("Test.B1";Hide!). See Chapter 8 for more detail.

Repeat (statement) Until(test)	All	This is a loop command. The statement portion continuously executes until the test portion is the boolean value False.
		For example, this routine uses the command to strip commas out of a string:
		x=StrPos(Num;",")
		    If(x > 0)
		    Repeat y=Substr(Num;1;x-1) z=Substr(Num;x+1;Strlen(Num))
		    Num=y+""+z x=StrPos(Num;",")
		    Until (x=0) EndIf
		When used with a dialog's Callback function and when not using CallbackWait, the (statement) portion isn't used – the DialogShow()'s Callback Label is the action repeated. The DialogShow() command has been preceded by a vLoop=False statement. So, in that context, the statement reads: Repeat Until(vLoop). In the Callback Label, vLoop will have been defined to equal True when some button is pressed in the dialog, thus meeting the test condition and causing the loop to end. Also, see While/EndWhile, below.

RepeatValue(numeric)	All	Sets the numeric value (number of times) the action which follows the command will be repeated. Examples: RepeatValue(5) Tab ... inserts 5 tabs at the insertion point. RepeatValue(33) Type("_") makes a line with the underscore key 33 characters long (that's usually about the right length for a signature line).
ReplaceAll()	All	Used with search and replace operation to replace all occurrences of the search and replace operation. Possible parameters are Extended! or Regular! Use of ReplaceAll without any parameter is a "regular" search. To include footnotes and/or endnotes in the search and replace operation, use the Extended! parameter. This command must be used in conjunction with other Search and Replace commands. See Search Commands, below. Example: ReplaceAll(Extended!) includes footnotes in the search and replace operation.
ReplaceBackward()	All	Replaces specified text or codes from the insertion point backward to the beginning of the document. See Search/Replace, below, and ReplaceAll() above for additional information. Optional parameters are Extended! or Regular! – if no parameter is used, Regular! (which does not include footnotes, text boxes, endnotes) is presumed. See Search Commands, below. Example: ReplaceBackward(Extended!) will search backward through regular text, footnotes, endnotes, text boxes. ReplaceBackward will search backward through regular text.

Command/Routine	Wp Ver	Syntax/Usage and Examples
ReplaceCurrent	All	Replaces a matched word, code or phrase. Precede with commands such as SearchString and SearchNext. Search Commands, below.
ReplaceForward()	All	Replaces specified text or codes from the insertion point to the end of the document. See Search/Replace, below, and ReplaceAll() for additional information. Optional parameters are Extended! or Regular! – if no parameter is used, Regular! (which does not include footnotes, text boxes, endnotes) is presumed. Search Commands, below. Example: ReplaceForward(Extended!) will search forward through regular text, footnotes, endnotes, text boxes. ReplaceForward without a parameter will search forward through regular text.
ReplaceString()	All	Identifies the replacement string and/or codes when a search string is found in a search/replace operation. Unless the replacement item is a variable, the replacement string is between a pair of quotation marks. If a code is included in the replacement string, use the Codes... button on the Macro Toolbar to identify and insert into the replacement string. A search or replacement code cannot be "typed" in, it must be inserted using the Codes... button on the Macro Toolbar, just as you would do with a regular Search/Replace in ordinary WordPerfect routines. If the replacement string is nothing at all, use a pair of quotation marks with nothing in between. See Macro Toolbar Codes button in Chapter 5 and Search Commands, below. Examples: ReplaceString("apples") or ReplaceString("[HRt]"). In the latter, the HardReturn code was inserted using the Codes... button on the Macro Toolbar by selecting it and clicking the Insert button in that dialog. ReplaceString("") deletes the found string. Don't forget to add the pair of quotes in using the ReplaceString() command.
Return	All	Ends a routine; if at the end of a Call Statement, action returns to the point of the call; if at the end of a macro but not in a Call Statement, the macro stops and the command is the same as Quit; if at the end of a macro being run from another macro, action returns to the parent macro. See Chapter 4 and Run().
RevealCodes()	All	Turns on/off Reveal Codes with the On! or Off! parameter. Usage: RevealCodes(On!) or RevealCodes(Off!)
Run()	This applies to Wp6.1 & higher. Run("MacroName.wcm") runs (nests) another macro from a "parent" macro – the macro containing the Run command – and the parent macro continues to run (remain active) at the same time. Its purpose is to "run" another macro while the parent macro is still running – ordinarily it would be expected that macro flow would or could return to the parent macro after the "child" macro finishes running. If you intend for that to happen, it is essential that you know how to use the "Run" command. It is not important if you intend that each macro be "stand-alone", so to speak. If you intend that variables already defined in the parent macro be recognized in the called (run) macro, make Global variables to cause a variable and its value to be recognized in the called macro before using the Run command. Unless you do (or unless you make a Persist variable), although Local variables will be recognized in the parent macro once macro flow returns to it, the called macro will neither recognize the existence of the variable nor its value, and, in the called macro, unless you've included a VarErrChk(Off!) command in the called macro, macro error will occur and the macro(s) will stop. You may or may not want to define Global variables within the called macro so that the parent macro will recognize the existence and value of such variables once macro flow returns to the parent macro. Syntax: Run("vmacro.wcm") where "vmacro.wcm" is the string name of the macro you want to run. Macro/PathName: If the macro you want to run is in the default or supplemental macro directory (as defined in Settings ... Wp6.1 and 7.0: Edit | Preferences | Files | Merge/Macro ... Wp8.0 & higher, Tools | Settings | Files | Merge/Macro), the full pathname is probably not necessary. Or, if it is and the macro you want to run is in the default or supplemental macro directory, you can use the ?PathMacros command and use concatenation to show	

Command/Routine	Wp Ver	Syntax/Usage and Examples
	the macro's path like this: Run(?PathMacros+"vmacro.wcm"). If the macro is not in the default or supplementary macro directory, use the full pathname, e.g., Run("C:\WpMacros\vmacro.wcm"). At the conclusion of the called (run) macro, if you intend macro flow to be returned to the parent macro, a Return command should be included at the point that you intend for that to occur. A "Quit" command in the called macro prevents that from happening. Otherwise, just as with a Call() command, macro flow will then resume in the parent macro at the point immediately following the Run command. Also, see Chain().	
Search Commands	All	These are links to various items used in writing search and/or replace routines, listed here for convenience. MatchExtendSelection MatchPositionAfter MatchPositionBefore MatchSelection ReplaceAll ReplaceBackward ReplaceCurrent ReplaceForward ReplaceString SearchCaseSensitive SearchNext SearchPrevious SearchString If you want the search to include footnotes/endnotes/headers/footers, use the Extended! parameter in SearchNext (Extended!) ReplaceForward (Extended!), ReplaceAll (Extended!). If you don't want an "extended" search, no need exists to include the Regular! parameter, e.g., in SearchNext (Regular!) the Regular! parameter is unnecessary. It will often be desirable to combine search routines with an OnNotFound() or OnNotFound Call() routine to avoid macro error if the searched for item is not found. See OnNotFound(), OnNotFound Call() and the discussion in Chapter 4. Examples: PosDocTop SearchString(var1) ReplaceString(var2) ReplaceForward SearchString(var1) MatchPositionAfter SearchNext SearchString("book") ReplaceString("pamphlet") MatchSelection SearchNext ReplaceCurrent SelectMode(On!) SearchString("[HPg]") MatchExtendSelection SearchNext PosCharPrevious EditCopy SelectMode(Off!) (where the [HPg] code is inserted using the Codes... button on the macro toolbar - see ReplaceString, above). Also, see search routines in ConvertFE.wcm.
SearchCaseSensitive()	All	Makes a search/replace case sensitive or not with the No! or Yes! parameter. If you don't use this command, searches will not be case sensitive unless you have done an earlier search which turned on the Search Case feature. See Search Commands, above. Usage: SearchCaseSensitive(Yes!)
SearchNext()	All	Searches forward to the next occurrence of the last search performed. Use the Extended! parameter to include footnotes, text boxes, endnotes, in the search. Use the Regular! parameter to exclude such areas in the search. If no parameter is specified, e.g., just SearchNext, a regular search is performed. See Search Commands, above.
SearchPrevious()	All	Same as SearchNext(), except that the search is backward. See Search Commands, above.
SearchString()	All	Identifies the search string and/or codes to search for. Unless the search item is a variable, the search string is between a pair of quotation marks. If a code is included in the search string, use the Codes... button on the Macro Toolbar to identify and insert into the search string. A search or replacement code cannot be "typed" in, it must be inserted using the Codes... button on the Macro Toolbar. See Macro Toolbar Codes button in Chapter 5, and Search Commands, above, and search routines in ConvertFE.wcm. Examples: SearchString("bananas") and or SearchString("[Left Tab]"). In the latter, the Left Tab code was inserted using the Codes... button on the

Command/Routine	Wp Ver	Syntax/Usage and Examples
		Macro Toolbar by selecting it and clicking the Insert button in that dialog. Don't forget to add the pair of quotes.
Select Commands	All	Many Select... commands are described below. See on-line Macro help for more. For differences in Wp10's (& higher) text selection method which can affect various Select... macro commands, see BMT; wp9select.wcm
SelectAll	All	Selects all text and graphics in the current document.
SelectCell	All	Selects the current cell. Not valid for floating cells (not covered in this manual).
SelectCellDown	All	Selects from the insertion point down one row.
SelectCellDownArrow	All	Selects the current cell and one cell down. Not valid for floating cells (not covered in this manual).
SelectCellLeft	All	Selects the current cell (cell containing the insertion point) and extend the selection to the previous cell. If the current cell is the first cell in the first row (A1), select the current cell. If the current cell is in the first column (column A) and not in the first row, select the current cell, extend the selection up one row, then continue to extend the selection to the rightmost column. The resulting selection includes the current row and the previous row with the insertion point in the last column of the previous row.
SelectCellRight	All	Selects the current cell and the next cell to the right. In the farthest-right cell, select the entire current row and the row below.
SelectCellUp	All	Selects the current cell and one cell above. In the top row of a table, does not select text or codes above the table.
SelectCellUpArrow	All	Selects the current cell and one cell above. In the top row of a table, select from the insertion point to the line above the table.
SelectCharNext	All	Selects the text or code one character position to the right of the insertion point.
SelectCharPrevious	All	Selects the text or code one character position to the left of the insertion point.
SelectColumnBottom	All	Selects the text and codes from the insertion point to the bottom of the current column.
SelectColumnNext	All	Selects the text and codes from the insertion point to the right one column. In the farthest-right column, selects the text and codes from the insertion point to the beginning of the current line.
SelectColumnPrevious	All	Selects the text and codes from the insertion point to the top of the current column, and from the bottom up to and including the corresponding line in the column to the left. In the farthest-left column, selects the text and codes from the insertion point to the beginning of the current line.
SelectColumnTop	All	Selects the text and codes from the insertion point to the top of the current column.
SelectDelete	All	Deletes selected text. Also, see DeleteCharNext.
SelectDocBottom	All	Selects the text and codes from the insertion point to the end of the document.
SelectDocTop	All	Selects the text and codes from the insertion point to the beginning of the document.
SelectDocVeryTop	All	Selects the text and codes from the insertion point to the very beginning of the document, before all codes.

Command/Routine	Wp Ver	Syntax/Usage and Examples
SelectLineBegin	All	Selects the text and code from the insertion point to the beginning of the current line, after any beginning codes.
SelectLineDown	All	Selects the text and codes from the insertion point down one line.
SelectLineEnd	All	Selects the text and codes from the insertion point to the end of the current line, before any ending codes.
SelectLineUp	All	Selects the text and codes from the insertion point backward to the corresponding position on the line above.
SelectLineVeryBegin	All	Selects the text and code from the insertion point to the beginning of the current line, before any beginning codes.
SelectLineVeryEnd	All	Selects the text and code from the insertion point to the end of the current line, after any beginning codes.
SelectMode()	All	Turns Select on or off with the On! or Off! parameter. Wp10 & higher users: see BMT; wp9select.wcm Usage: SelectMode(On!) or SelectMode(Off!)
SelectPage	All	Selects all text on the current page.
SelectPageNext	All	Selects the text and codes from the insertion point to the beginning of the next page. If used on the last page, selection will be to the end of the document.
SelectPagePrevious	All	Selects the text and codes from the insertion point to the beginning of the previous page. On the first page, selects the text and codes from the insertion point to the beginning of the document.
SelectParagraph	All	Selects the current paragraph and the subsequent codes until the text resumes. Wp10 & higher users: see BMT; wp9select.wcm
SelectParagraphNext	All	Selects the text and codes from the insertion point to the beginning of the next paragraph. In the last paragraph, selects the text and codes from the insertion point to the end. Wp10 & higher: see BMT; wp9select.wcm
SelectParagraphPrevious	All	Selects the text and codes from the insertion point to the beginning of the current paragraph. At the beginning of a paragraph, selects the text and codes from the insertion point to the beginning of the preceding paragraph. Wp10 & higher users: see BMT; wp9select.wcm
SelectSentence	All	Selects the current sentence. Wp10 & higher: see BMT; wp9select.wcm
SelectSentenceNext	All	Selects the text and codes from the insertion point to the beginning of the next sentence. Wp10 & higher users: see BMT; wp9select.wcm
SelectSentencePrevious	All	Selects the text and codes from the insertion point to the end of the previous sentence. Wp10 & higher users: see BMT; wp9select.wcm
SelectTable	All	Selects all cells in the current table. If the insertion point is not in a table, the macro ends or goes to the OnError Label, if one exists.
SelectTableColumn	All	Selects all cells in the current column of a table.
SelectTableColumnExtendLeft	All	Selects the text and codes from the insertion point to the beginning of the current row.
SelectTableColumnExtendRight	All	Selects the text and codes from the insertion point to the beginning of the last column.
SelectTableRow	All	Selects all cells in the current row.
SelectWord	All	Selects the current word. Wp10 & higher: see BMT; wp9select.wcm
SelectWordNext	All	Selects the text and codes from the insertion point to the beginning of the next word. Wp10 & higher users: see BMT; wp9select.wcm

Command/Routine	Wp Ver	Syntax/Usage and Examples
SelectWordPrevious	All	Selects the text and codes from the insertion point to the end of the previous word. Wp10 & higher users: see BMT; wp9select.wcm
SetDefaultParent	Wp9 & higher	It's best just to state J. Dan Broadhead's comments here: Prior to version 9, you could not specify the parent for any of the other prompts, message boxes, etc., that can be used in a macro. But starting in version 9, all those commands (including dialog boxes) now use a specific macro system internal method to determine the parent window. This parent window is initially set to the process that started the macro (usually WP), but this parent window can be explicitly specified by using the SetDefaultParent command in WP9 and WP10. Dialog boxes are still the only commands where a specific parent can be specified when the DialogShow command is called, but all the other commands use the contents of the values specified by the SetDefaultParent command. Also, see MessageBox and, very broadly, the DialogShow reference in Chapter 5.
SGML Commands	Wp8 & higher	None of the numerous SGML (Standard Government Markup Language) commands are included in this manual.
SingleSmartQuote()	All	Turns on/off Single Smart Quotes with the On! or Off! parameter. See DoubleSmartQuote() and Quick... Commands, above. Usage: SingleSmartQuote(On!) or SingleSmartQuote(Off!)
SingleSpaceInSentence	All	Changes double spaces in a sentence to single spaces (or not) with the On! or Off! parameter. Usage: SingleSpaceInSentence(On!) or SingleSpaceInSentence(Off!)
Sort Commands	All	None of the several Sort commands are included in this manual.
SoundClip Commands	All	None of the several SoundClip commands are included in this manual.
SpellAsYouGo	Wp7 & higher	Turns on/off SpellAsYouGo with the On! or Off! parameter. Or, use the command to determine whether SpellAsYouGo is on or off, in which event a true/false boolean value is assigned to a variable. Examples: SpellAsYouGo(Off!) or var=SpellAsYouGo. In the latter, var will be assigned a boolean true or false value.
Style Commands	All	Except for the item below, none of the several Style commands are included in this manual.
StyleSystemOn()	All	In this vastly oversimplified an incomplete description of the use of this command, for common usage, use StyleSystemOn(stylename) to turn on a WordPerfect or user defined style. In my personal use, I use a "letterhead" style for my office letterhead, and the style used is named "Doug". Using this command in a new blank document inserts the letterhead style at the top of the new document, and this is extremely useful in making office or home correspondence. The Style name is not in quotes. If a named style does not exist, nothing will occur. Example: StyleSystemOn(Doug) [no quotation marks]
Str... Commands	Wp7 & higher	Several Str... commands are covered in this manual, but others are not. Covered: StrFill; StrInsert; StrLen; StrNum; StrPos; StrReverse; StrToChars; StrTransform; StrTrim; SubStr Not Covered: Except as noted, see Wp macro help in WordPerfect 7.0 or higher. StrFraction; StrIsChar; StrLeft; StrMakeList; StrPad; StrParseList; StrRight; StrScan; StrUnit (6.1 & higher)

Command/Routine	Wp Ver	Syntax/Usage and Examples

StrFill()	Wp7 & higher	Replicates a string a specified number of times.
		Usage: vString=(n;"string"), e.g.,
		x=StrFill(3;"Doug") Messagebox(;"x";x) //x="DougDougDoug"
		The 2nd parameter is optional. If not included, a "space" would be replicated.

StrInsert()	Wp7 & higher	Inserts characters into a string, replaces characters in a string, or removes characters in a string.
		Usage: vString=StrInsert(string; substring [optional]; begin at [n] [optional]; # of characters [n] [optional]).
		Paraphrasing WordPerfect's macro help:
		1st parameter: string (the original string)
		2nd parameter: substring: (optional – the string to insert; if missing, characters are removed from String
		3rd parameter: numeric beginning point (optional – the starting position for inserting the substring, or for removing characters from the string; if missing, the starting position is the end of the string; if less than zero, the starting position is the absolute value taken from the end of the string)
		4th parameter: number of characters (optional – the number of characters replaced by substring; if substring is missing, the number of characters to remove from the string; if missing or 0, no characters are removed; if less than zero, all characters from the starting position to the right are either replaced by substring or removed)
		Examples:
		x=StrInsert("Doug"; "las") Messagebox(;"x";x) // x="Douglas"
		x=StrInsert("Doug"; "las";5) Messagebox(;"x";x) // x="Douglas"
		x=StrInsert("Doug"; "las";1) Messagebox(;"x";x) // x="lasDoug"
		x=StrInsert("Doug"; ;-2;1) Messagebox(;"x";x) // x="Dog"
		x=StrInsert("Doug"; ;-3;1) Messagebox(;"x";x) // x="Dug"

StrLen(string)	All	Returns the numeric length of characters in a value (numeric or string) or a variable, including spaces, e.g., x=StrLen("Oklahoma"), x will = 8; x=StrLen(var), x will = the total characters, including spaces, in the "var" variable. See numerous macro examples in this paper, particularly Chapter 6, working with decimals.
StrNum()	All	Converts a string containing these characters (+-0123456789 and "period" (decimal)) to a numeric value. If the 1st character in the string is not a number, or is not a number preceded by a minus (-) sign, a plus (+) sign or a decimal (.), error will result. Commas are not numeric characters. If the string begins with a legal character but is followed by non-legal characters, everything after the legal character is ignored. So, error is produced by all of the following: num=StrNum("abc20"), num=StrNum("-abc20"), num=StrNum(".abc20"). And, while not producing error, exemplary results mixing numbers and characters are as follows: num=StrNum("20abc30") results in num equaling 20; num=StrNum("200,000") results in num equaling 200; num=StrNum("200.005.008") results in num equaling 200.005 Usage: num=StrNum("105.5") or num=StrNum(var), where the variable contains a numeric value. See NumStr and Chapter 6, working with decimals.
StrPos()	All	Returns the 1st position of a specified string in a string, numeric value, or variable. If the string is not present, the value 0 is returned. Usage: num=StrPos(string [or var];substring), e.g., x=StrPos(var;"p") – if var is "oppressed", x will = 2 but if var is "kindly", x will = 0. See numerous macro examples in this paper, particularly Chapter 6, working with decimals and Automatic Data Conversion.

Command/Routine	Wp Ver	Syntax/Usage and Examples
StrReverse()	Wp7 & later	Reverses the order of characters in a string. I can't think of an occasion to use this unless just fooling around, but maybe you can. vStr=StrReverse("You are such a jerk, I just wish you would go away") Messagebox(;"yStr";yStr) // vStr="yawa og dluow uoy hsiw tsuj I ,krej a hcus era uoY"

StrToChars()	Wp7 & later	Converts one string value to another by keeping or removing characters from a string. The 1st parameter is the string value to be converted; the 2nd parameter (optional) is an enumeration, Keep! or Remove! (if missing, Keep! is implied); and the 3rd parameter (optional) is either the actual "string" value OR the enumeration value for predefined "character sets" to be kept or removed - Alphabetic!, AlphaNumeric!, Numeric!, Punctuation!, WhiteSpace!, UpperCase! and LowerCase!. In the 3rd parameter, for predefined character sets, enumerations can be combined by a "pipe" (|) symbol.
		Usage: x=StrToChars(vString; enum; Str or enum)
		Example: x=StToChars("12345.406.505";Keep!;".") //x=".." since only decimals have been kept
		Example: x=StrToChars("0.05%";Remove!;"%") //x="0.05" since "%" has been removed.
		Example: x=StrToChars(" 123,456.789";Remove!;Punctuation! | WhiteSpace!) //x="123456789"
		See an example in Math.wcm.

StrTransform()	Wp7 & later	Converts a string of characters into other characters.
		Syntax: x=StrTransform(String: string; FromChars: string; [ToChars: string]; [Options: enumeration])
		The 1st parameter is the string to be transformed. The 2nd parameter identifies the string within the string to be transformed. The 3rd parameter [ToChars] is optional and it identifies the "replacement" string. WordPerfect help says, "If ToChars is missing or shorter than FromChars, all characters are removed from String that match characters in FromChars but have no corresponding character in ToChars. The 4th parameter (optional) controls the interpretation of FromChars and ToChars and how many transformations are performed. Possible enumerations are: Characters!; Strings! FirstOnly!; All! (the default). Unless the Strings! enumeration is used, only exactly matching FromChars and ToChars are affected (see last example).
		Example: var=StrTransform(var;"%";"") or var=StrTransform(var;"%") //both remove all "%" signs from var
		Example: var=StrTransform(var;",";";") //replaces all commas with semi-colons
		Example: If var="50.78 %",
		var=StrTransform(var;"%";"percent";Strings!) //var="50.78 percent"
		var=StrTransform(var;"%";"percent";Characters!) //var="50.78 p")
		var=StrTransform(var;"%";"percent";All!) //var="50.78 p"
		var=StrTransform(var;"%";"percent";Strings!) //var="50.78 percent"
		See examples in Math.wcm: Example 1; Example 2

StrTrim()	Wp7 & Later	Removes characters from a string. Syntax: x=StrTrim(String: string; [Length: numeric]; [Option: enumeration]; TrimChars: string or enumeration) The 1st parameter is the string to be "trimmed". The 2nd parameter (optional) is the final length of the trimmed string. If the length is not less than the original string's length, no trimming is done. If the 2nd parameter is missing, the string is trimmed until all matching characters are removed at the locations identified in the 3rd paramater [if any]), but not more than to the length specified in the 2nd parameter. The 3rd parameter (optional) enumerations are TrimRight! (the default); TrimLeft!; TrimEnds!; TrimWords!. The 4th parameter is the string to be removed or an optional enumeration: Alpahbetic!; AlphaNumeric! Numric!; Punctuation!;

Command/Routine	Wp Ver	Syntax/Usage and Examples

		WhiteSpace!; UpperCase!; LowerCase!. 4th parameter enumerations can be combined using the "pipe" (|) symbol, but not with a literal string. It does not appear that a "period" (.) can be trimmed *if it is the only character identified in the 4th parameter* or by using the Punctuation! enumeration.
		Example: var=StrTrim(var;;TrimLeft!;"0") // would eliminate all leading zeros from var, if any are present
		Example: If var="%50.78%",
		var=StrTrim(var;;;"%") // var="%50.78" since Right! enum is implied
		var=StrTrim(var;;TrimLeft!;"%") // var="50.78%"
		var=StrTrim(var;;TrimWords!;"%") // var="50.78"
		var=StrTrim(var;;TrimWords!;Punctuation!) // var="50.78"
		See example in Math.wcm: Example

SubStr()	All	Returns a portion of an existing string or variable; works with numeric values as well; can be used with StrPos and StrLen; the parameters are (a) the string, which can be a variable, (b) starting point, and (c) ending point. Usage: x=SubStr(var;starting point; number of characters), e.g., if var="Oklahoma", x=SubStr(var;1;1), x = "O"; x=SubStr(var;2;1), x = "k"; x=SubStr(var;5;4), x="homa"; x=SubStr(var;StrLen(var);1), x="a". See numerous macro examples in this paper, particularly Chapter 6, working with decimals and Automatic Data Conversion.
SubstructureExit()	All	If you are in a substructure (e.g., a footnote), this command can be used to exit that substructure and, with the Next! or Previous! optional parameter, it can open an existing substructure of the same type. Usage: SubstructureExit or SubstructureExit(Next!)
Suppress()	All	Turns off specified page formatting items for the current page. Parameters: FooterA!, FooterB!, HeaderA!, HeaderB!, PageNumberBottomCenter!, PageNumbering!, WatermarkA! and WatermarkB! (the latter 2 only being available in Wp 8 or later). Example: Suppress(PageNumbering!) suppresses whatever page numbering you've got going on the current page.
Switch(var) CaseOf EndSwitch	All	An extremely useful tool, this set of commands is used to test the value of a variable and execute whatever commands you give if the condition is true. A Switch command ALWAYS must end with an EndSwitch command. It has more clarity than a bunch of If - Else - EndIf statements containing multiple "If" commands. If the variable is a string value, use quotation marks in the CaseOf elements. If the variable is a numeric value, use numbers and no quotation marks. If the variable is a True/False boolean value, use the words, without quotes, True, False. Usage: Switch(var) CaseOf (value): [Do This] CaseOf (value): [Do That] Default: [Do The Other] – optional element if none of the CaseOf items are true. EndSwitch Example, where var="blue": If(Exists(var)) Switch(var) CaseOf "blue": MessageBox(;"The color is blue";"") Go (vNext) CaseOf "red": MessageBox(;"The color is red";"") Go (vNext2) CaseOf "": MessageBox(;"Var has no value";"Start over") Go

Command/Routine	Wp Ver	Syntax/Usage and Examples
		(Begin) CaseOf "orange"; "purple": MessageBox(;"The color is "+var;"") Go(vNext3) Default: MessageBox(;"No valid color was selected";"Start over") Go(Begin) EndSwitch EndIf Notice that it is possible to use multiple values in CaseOf, separated by semi-colons, as for "orange" and "purple", above. Explanation: In this example, the Switch command was preceded by If(Exists(var)), which conditioned the entire Switch routine upon the existence of a variable named "var". If "var" had not been declared, the contents of the Switch statement would be skipped, picking up after the EndIf command. Without the If(Exists(var)) EndIf commands, if var did not exist, error would result in the Switch routine since nothing would exist to "switch". But, since var="blue", the message box shown will execute after which macro flow will be sent to Label (vNext). If var had equaled "red" the message shown for "red" would have executed and the macro flow would have been sent to Label (vNext2). If var exists but had not been assigned any value, the stated message box would be displayed and macro flow would go to Label (Begin). The optional Default picks up everything else, e.g., if var DID have a value but which was not equal to the specifically stated CaseOf components, the commands stated would be executed. The above presents a simple example – the routines following a CaseOf statement can be as complex as needed. See examples in Chapter 6, Chapter 7, and numerous examples in Chapter 8. Note: while I tend to avoid the use of the MessageBox command, it was used above for simplicity in this description.
SwitchDoc(num)	All	Switches between currently open document. The parameter is numeric, 1 through 9. To avoid having to "know" what the document number is that you want to switch to, use the ?DocNumber command to identify a current document when you're in it, e.g., Doc1=?DocNumber. Then when you're in a different document and want to switch back to the earlier identified document, use SwitchDoc(Doc1) to switch to the document you previously identified as Doc1. Usage: SwitchDoc(number), either a number or a variable containing a numeric value of 1 ~ 9.
Tab Commands	All	Several Tab commands other than the few shown below are available but they are not covered in this manual.
Tab	All	Inserts a tab. In ordinary circumstances, Left Tabs will have been set in your default template, so the Tab command will insert a Left Tab at the insertion point. The generic Tab command utilizes the current tab settings at the insertion point. If you want to insure you use a Left Tab, use TabLeft, below. Example: Tab Type("1.") Tab Type("My dog is good.")
TabLeft()	All	Inserts a hart left tab at the insertion point. Use the DotLeader! parameter if you want the tab preceded by a dot leader. Usage: TabLeft or TabLeft(DotLeader!) or TabLeft(Normal!)
TabDecimal()	All	Inserts a decimal tab at the insertion point and is useful for creating a "column" of numbers which are properly aligned on the decimal point. See the example, below, which results in a column of numbers with the text properly aligned on the decimal. Optional parameters are DotLeader! (to precede the tab with a dot leader) or Normal! (unnecessary to use since it is the default). HardReturn Tab Type("Books") DecimalTab DecimalTab Type("$350.00") HardReturn Tab Type("Clothing") DecimalTab DecimalTab Type("75.00") HardReturn

Command/Routine	Wp Ver	Syntax/Usage and Examples
		Tab Type("Total") DecimalTab DecimalTab Type("$425.00")
Table Commands	All	Numerous Table commands are available but none are included in this manual (other than various table positioning and copying commands). See Position In Table commands, above, and a few more, below.
TablePosNextTable	All	Moves the insertion point to the beginning of the next table.
TablePosPreviousTable	All	Moves the insertion point to the beginning of the previous table.
Template Commands	All	Numerous Template commands are available but none are included in this manual.
TOA and TOC Commands	All	Numerous Table of Authorities (TOA) and Table of Contents (TOC) commands are available but none are included in this manual.
ToInitialCaps(var;enum)	Wp 7.0 & higher	This command is available in WordPerfect 7.0 & higher but not in WordPerfect 6.1. It converts either the first character of a string, or the first character of each word in a string, to upper case. Usage: var=ToInitialCaps(var;enumeration). Use of an enumeration is optional. They are FirstCharOnly! And FirstOfEachWord!, e.g., var=ToInitialCaps(var;FirstOfEachWord!). If an enumeration is not specified, FirstOfEachWord! will be implied. Also, see ToUpper(var) and ToLower(var), below.
ToLower(var)	All	Converts the contents of a string to all lower case; can be used to assign value to a new variable, e.g., x=ToLower(var), or can be used in conjunction with the Type command, e.g., Type("What's happening, "+ToLower(vName)+"?"). See ToUpper(var), below and ToInitialCaps(var), above.
ToUpper(var)	All	Converts the contents of a string to all upper case; can be used to assign value to a new variable, e.g., x=ToUpper(var), or can be used in conjunction with the Type command, e.g., Type("IN THE DISTRICT COURT OF "+ToUpper(County)+" COUNTY". See ToLower(var), above.
Type(string)	All	Type is the basic command to "type" something and a pair of parentheses mark the beginning and end of that which is to be typed. If raw text is to be typed, use quotation marks at the beginning and end of the parenthetical expression. If a variable is to be typed, don't use quotation marks for it. You can combine string text and variables by using the "+" joiner (called concatenation) as shown in an example below. If using Type() to type literal text, e.g., Type("abc"), observe the line limit. Use multiple Type() commands to break larger segments of text into more manageable pieces or assign variables to segments of what you want to type. See the following examples and the use of +, above. Examples: Type(Var), where "Var" is a predefined variable containing something. Type("This is some stuff.") or Type(var1) Type(var2) Type(var3) Type("Comes now the Plaintiff,"+PlaintiffName+", and for "+PPosPro+" Motion To Dismiss, "+PNomPro+" states as follows:"), where PlaintiffName is a previously defined variable, the value of which is the Plaintiff's name; and where PPosPro is a previously defined variable, the value of which is the appropriate possessive pronoun for the Plaintiff (his or her); and where PNomPro is a previously defined variable, the value of which is the appropriate nominative pronoun for the Plaintiff (he or she). Numerous examples are in this manual, e.g., Chapter 7, Date Routines.
UnderlineSpaces()	All	Turns on/off UnderlineSpaces with the Yes! or No! parameter. With UnderlineSpaces turned on or off, spaces between words will be or will not be underlined when the Underline command is invoked. Use ?UnderlineSpaces to determine whether UnderlineSpaces is currently turned on or off. See ?UnderlineSpaces, above. The "toggle" command, below, assumes that Underline is off when the code begins. Also, see UnderlineTabs(), below, and ?UnderlineSpaces, above. Example: UnderlineSpaces(Off!) AttibuteAppearanceToggle(Underline!)

Command/Routine	Wp Ver	Syntax/Usage and Examples
		Type("My dog has fleas.") - Text, when typed, will appear as My dog has fleas. UnderlineSpaces(On!) AttributeAppearanceToggle(Underline!) Type("My dog has fleas.") - Text, when typed, will appear as My dog has fleas.
UnderlineTabs()	All	Turns on/off UnderlineTabs with the Yes! or No! parameter. With UnderlineTabs turned on or off, tabs (and indents) between words will be or will not be underlined when the Underline command is invoked. Use ?UnderlineTabs to determine whether UnderlineTabs is currently turned on or off. See ?UnderlineTabs, above. The "toggle" switch command, below, assumes that Underline is off when the code begins. Also, see UnderlineSpaces() and ?UnderlineTabs, above. Example: UnderlineTabs(On!) AttributeAppearanceToggle(Underline!) Tab Type("My dog has fleas.") - Text, when typed, will appear as My dog has fleas., where a Tab precedes "My dog ..." and UnderlineSpaces is On. UnderlineTabs(Off!) AttributeAppearanceToggle(Underline!) Tab Type("My dog has fleas.") - Text, when typed, will appear as My dog has fleas., where a Tab precedes "My dog ..." and UnderlineSpaces is On.
VarErrChk()	All	Determines how a macro responds to a macro not containing a variable (or a declared variable with no value) but which nonetheless references it. By default, such error checking is "On". VarErrChk(Off!) turns checking off and VarErrChk(On!) turns it on. See Run() and Chain().
Variables, User Defined The variable's name Duration of a variable Local variables Global variables Persist variables PersistAll	All	A "variable" isn't a command, it's a concept. Think of a variable as a "place in memory" which has a name you define and the contents of which you specify. Without "variables", macros would be little more than player pianos, always playing the same tune. If you want that "tune" to vary depending upon particular input you make which creates personalized and variable documents, you must get a good understanding of what a "variable" is. Note that this discussion has to do with user-defined variables, not the System Variables, above, 2, and not the special variable MacroDialogResult, above. The Variable's Name: the name can be up to 30 characters long but it makes for good sense and order to keep it short. While a variable can include numbers, it can't begin with a number. Only use alphabetical characters, numbers and the underscore character, not spaces or other keyboard characters. While you can use differing "case" for ease of reading, case is unimportant during a macro's execution. Duration of A Variable: Variables can be Local, Global or Persist. While a variable exists, it uses memory. "Local" variables are created by default. So, var="2001" creates a local variable named "var" the contents of which are the string 2001 (without the quotation marks, this would be a numeric value 2001). Local variables only remain in memory while the macro which defines them is running. So, if you have defined a local variable, then run another macro which attempts to use that variable and its value, error will occur (unless you've included a VarErrChk(Off!) command in the 2nd macro) and the variable's value will not exist. When a Return command is encountered in the 2nd macro and macro flow returns to the 1st macro, the variable and its value will still be recognized in that macro. See Run() and Chain(). "Global" variables stay in memory until discarded or until all macros involved in a particular macro stop. If you want a variable and its value to be recognized when other macros are run from the parent macro, use the Global command to make that happen. Usage: Global var=var See Run() and Chain(). "Persist" variables stay in memory until discarded or until the WordPerfect session ends. As a practical matter, this will seldom be used. It is also possible to construct macros which retain variables in memory using the PersistAll command. That command is not developed in this manual. See Declare, above, for creating local variables with multiple elements,

Command/Routine	Wp Ver	Syntax/Usage and Examples
Arrays		used in making Arrays.
VersionInfo()	Wp8 & higher	Syntax: var=VersionInfo() VersionInfo assigns to a variable the value of particular items of information associated with the WordPerfect version you are then running. WordPerfect 10's on-line Macro Help does not correctly state some of or all the parameters, so I'll not parrot that information here - but see on-line macro help for additional uses than are shown here, but with eyes-wide-open. About the only reason I'm including this command in this manual is that it provides a work-around for the broken ?MajorVersion command in WordPerfect 10's initial release (but fixed in WordPerfect 10's Service Pack 2). var=?MajorVersion should assign var the value of 6, 7, 8, 9 or 10. It does that in Wp6.1, 7.0, 8.0 and 9.0. In Wp10 (initial release), var is returned the value of 9, obviously incorrect. If you have a need to cause a macro operate differently in WordPerfect 6.1, 7.0, 8.0, 9.0, or 10.0 (for any number of possible reasons), var=?MajorVersion, combined with a Switch - EndSwitch (or If - EndIf) sequence can sometimes but not always be a helpful tool in writing macros. But, since the same command in WordPerfect 10's initial release returns a value of 9, even though fixed in WordPerfect 10's Service Pack 2, you may want to use this alternative. A work-around for the initial WordPerfect 10 release is: var=VersionInfo(ProductMajorVersion!). In WordPerfect 10, var would be assigned the numeric value 10, which is what ?MajorVersion should do. Also, see MacroInfo and IfPlatform - EndIfPlatform.
VLineCreate	All	Creates a vertical line at the insertion point between the top and bottom margin. Also, see HLineCreate, above.
Web Commands	Wp8 & higher	More than 60 "Web" commands are present in WordPerfect 8, 9 and 10, none of which are covered here. Some commands are described as obsolete in WordPerfect 10.
While / EndWhile	All	While / EndWhile is a loop statement that executes while the expression at the top of the loop is true. The loop does not execute the first time unless Test is true. When Test is false, the first statement after EndWhile is executed. EndWhile must close the statement. Usage: While [test] [statement block executes] EndWhile. Example: While(Exists(var)) Discard(var) EndWhile. See Repeat and Exists, above, and examples in Chapter 6, working with decimals.
Zoom Commands	All	Some, but not all, Zoom commands are listed below. See on-line Macro help for more information.
ZoomToFullPage	All	Zooms the current document to a full page view on screen. See DisplayZoom(), ?DisplayMode, and ?Zoom, above.
ZoomToMarginWidth	All	Zooms the current document to see the area between left/right margins. See DisplayZoom(), ?DisplayMode, and ?Zoom, above.
ZoomToPageWidth	All	Zooms the current document to see the area between the left and right edges of the page. See DisplayZoom(), ?DisplayMode, and ?Zoom, above.

Top of Chapter Index Release Notes

NOTES

NOTES

NOTES

NOTES